VIETNAMESE - GERMAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

and

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Computer Science

Heuristic methods for Target Coverage Problem in Mobile Air Quality Monitoring Systems

Bui Xuan Phuoc

VIETNAMESE - GERMAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

and

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Computer Science

Heuristic methods for Target Coverage Problem in Mobile Air Quality Monitoring Systems

Author:	Bui Xuan Phuoc
Supervisor:	Dr. Tran Thi Thu Huong
Advisor:	Assoc. Prof. Do Phan Thuan
Submission Date:	24.09.2021

I hereby declare that this paper is my own work and has not been submitted to any institution for assessment purposes. Further, I have cited and acknowledged all sources and materials used in this paper.

Ho Chi Minh City, Vietnam, 24.10.2021 Bui Xuan Phuoc

Acknowledgments

I would like to thank my supervisors, Dr. Tran Thi Thu Huong and Assoc. Prof. Do Phan Thuan for their patience and support. I would have not been able to finish this thesis without their guidance and the resource they have allocated for me. I would also want to express my appreciation towards Dr. Michel Toulouse for having inspired me to love Computer Science and not to give up on this path.

Abstract

In this paper, a heuristic algorithm for the optimal trajectories and the scheduling of air quality measuring sensors is proposed. In order to be able to monitor the air quality of large urban areas, an adequate system of sensors must first be deployed. The objective is to be able to monitor as many important areas as possible with the sensors network with limitations to the amount of sensors and their capabilities. To achieve this goal, we proposed an effective genetic algorithm and multiple strategies to generate initial populations for the process. Together, the genetic algorithm with different initial populations are tested using real-world bus routes of Hanoi and Ho Chi Minh City which resulted in statistically significant better results when compared to previous approaches to the problem.
Keywords: sensor on bus problem, genetic algorithm, two-layers encoding, target coverage problem.

Contents

1 Introduction

One the greatest problem of our era is air pollution, which not only impact public's health but also play a large role in climate change. The problem is most severe in big and crowded cities, due to industrialization, increasing traffic transportation, and the heavy use of fossil fuel. As such, a large-scale and extensive air monitoring system is essential to enhance the public's health and to help the government in their administration and planning.

The initial step to monitor the air quality of an area is to have sensors deployed at the locations near or in the target areas. A conventional but naive approach is to have air quality sensors installed at fixed locations. However, this method restricts the sensor from monitoring outside its bounded area. Moreover, operating a station for this purpose alone might prove to be very expensive, especially when a large amount of stations is needed to be deployed. For instance, Hanoi has a total of less than 50 stations for air monitoring while its area is $3329km^2$ and Ho Chi Minh City only has 22 stations for air monitoring while having an area of $2061km^2$ [1].

To this goal, a novel solution has been developed in recent research, namely the vehicle-based mobile air quality monitoring system [11]. In this approach, air sensors are installed on vehicles, specifically buses, to expand the controlled area. In order to effectively deploy such a system, the problems of choosing which buses to have the sensors mounted on and when to perform the measurement have to be solved, or at least an effective solution must be found. This problem was proposed and named the "Sensor on bus problem (SOBP)" by Nguyen et al. [18]. An attempt to tackle the problem with an approximation greedy algorithm was made in the same paper, which was tested on the real-world Hanoi's bus routes and yielded better performance than the approximation.

The problem of sensor scheduling and placement optimization in static sensor networks has been researched extensively in the current literature. In [2], the authors investigated an evolutionary approach named Cuckoo search to solve the target coverage problem with limits to the amount of sensors in underwater wireless networks. F. Senel et al. [21] describe a method to tackle to target coverage problem by splitting the problem into smaller problems with only three sensors. In [19], the performance of many heuristics, including binary integer linear programming and evolutionary algorithms, for the target coverage problem was studied. However, all of the research mentioned earlier only took into account static sensors.

Regarding the target coverage problem in mobile wireless networks, there are very few relevant researches. Nguyen et al. [17] suggested a scheduling algorithm to move sensors in order to maximize the total weighted value of covered targets. The authors in [4] utilize heuristic approaches to solve the problem of minimization of the travel distance of sensors to cover all the required targets. Different from these problem instances, the SOBP concerns both the course of the sensors and the time to perform the measurement.

Our contributions are as follows:

- We provide a possible candidate solution model for the SOBP. Since the SOBP possesses both the problem of deciding the trajectory and the scheduling of the sensors, we created a two-layers genotype to represent a solution to the SOBP. Following this modeling and the constraints given by the problem, suitable heuristic operators were derived specifically for them. Additionally, we also included a version of Mixed Integer Programming model to try and achieve the optimal solution for the said problem.

- We propose a genetic algorithm (GA) that results in statistically better solutions than what the greedy algorithm can provide. We also provide different strategies to generate the initial population for the proposed GA. These strategies are also tested in real-world Hanoi's and Ho Chi Minh City's bus maps using statistical methods to have them fully examined.

The remainder of the paper is structured as follows. The next section describes the problem, its mathematical formulation, the modeling of candidate solutions in our paper and the procedure necessary to transform real-world data to our mathematical model. Section 3 presents possible approaches to the problem, including a brief review of Nguyen et al.'s greedy approach [18], our proposed GA in detail, and our proposed MIP model to compute the optimal solution for each problem instance. Section 4 includes the experiment details, the result of the experiments, the statistical testing of the results and their highlights. Section 5 describes the possible threat to the validity of our model. Finally, section 6 provides a concise overview and conclusions that were drawn from our research.

2 Problem formulation and hardness of Sensor On Bus Problem

2.1 Sensor On Bus Problem formulation

The Sensor On Bus Problem (SOBP) in our experiment can be described as follows. A grid of $p \times q$ (p and q are predefined integers) squares with each square marked as *critical* or *non-critical*, where the critical squares represent the areas that need to be monitored. A map of n bus routes is scaled and modeled on the grid, each of these routes is represented by an ordered list of poly-lines. There are m ($m \leq n$) air quality monitoring sensors. These sensors are to be attached to buses where each bus can have at most one sensor installed. In this problem, we assume that every bus route has only one bus.

The SOBP asks to choose m bus routes to install sensors on and to choose the combinations of turn-on points for each sensor installed to maximize the number of observable critical squares.

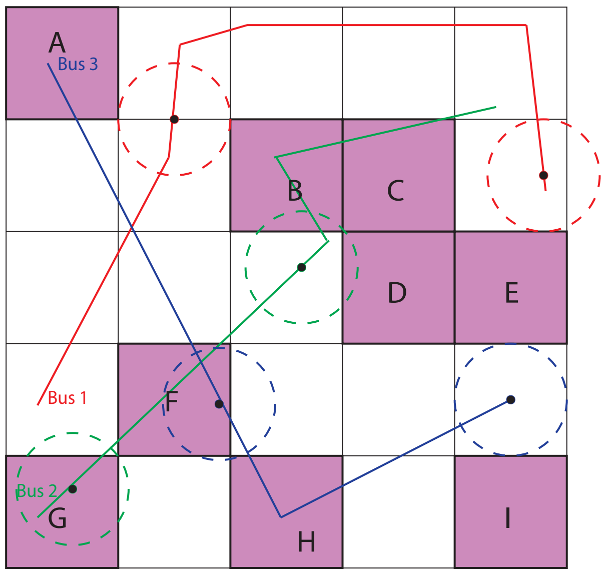

Figure 2.1: A map of size 5×5, with 3 bus routes and 9 critical squares. When $k = 2$, with such selected positions, if a sensor is installed on Bus 1, that sensor can observe 3 critical squares A, B, and E. With $m = 3$ and $k = 2$, every critical square in this instance can be covered as described in the figure.

Nguyen et al. [18] have proven that the well-known $\mathcal{NP}$-hard maximum coverage problem [8] can be reduced to the SOBP. Thus, making the SOBP a problem in the $\mathcal{NP}$-hard class. As for any $\mathcal{NP}$-hard problems, it is vital to develop and research on possible heuristic or approximation approaches to find solutions for them. In this paper, a heuristic approach is introduced and examined, namely the genetic algorithm.

2.2 Turn-on points and critical points

For any critical square A and bus route b, if A can be observed from b from one of the turn-on points of the sensor installed on b, then that turn-on point must be within a closed interval $[\ell_{Ab}, r_{Ab}]$. In order to calculate these intervals, we must recognize that for an arbitrary point p, if $d(p, A) \leq r$ then p must be in the area of squircle such that every point on the perimeter of this squircle has its distance to A equals to r. Our strategy to calculate these intervals is to loop through every pair of the bus route and critical squares, perform simple geometry calculations to get the intersections between the bus routes and the critical squares' corresponding squircles. These intersections are the critical points that make up the intervals. The order of the critical points in the interval can later be trivially derived from the coordinates of the beginning of the bus route and the bus route's poly-points. Note that a single critical square may have multiple non-continuous intervals on one same bus route.

Procedure 1 demonstrates how intervals are obtained. The idea is to calculate the intersections of the squircle and the bus route's polylines. After this process, intervals might have to be merged, reducing the number of critical squares and thus provide better search space for heuristic algorithms.

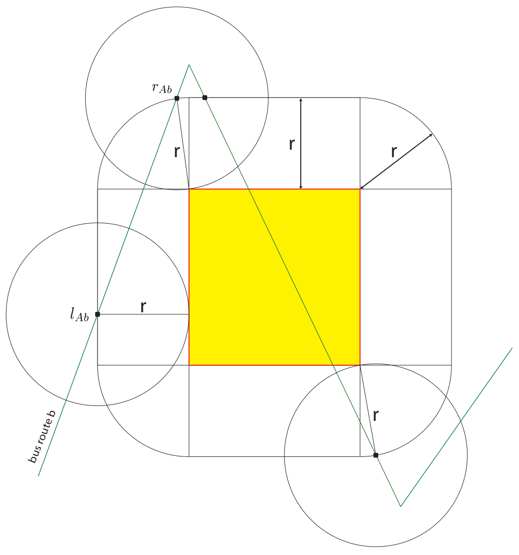

Figure 2.2: Critical point interval

Let d be an arbitrary turn-on point in bus route b, from d we will only be able to observe a set of critical squares $C = \{A \mid t \in [\ell_{Ab}, r_{Ab}]\}$. Thus, in order to minimize the search space of the problem, only the critical points will be considered to be the turn-on positions of the sensors.

2.3 Representation and evaluation

As mentioned, the SOBP can be viewed as a hybrid problem of optimizing both the trajectory and the schedule of the sensors. Thus, a two-layer representation was used to represent a candidate solution. To represent the bus routes chosen to install sensors on, we use a binary n-tuple to encode the genotype $\bar{b} = (b_1, ..., b_n)$ such that gene b_k represents whether a sensor should be deployed on the corresponding k^{th} bus route in the bus map:

$$b_k = \begin{cases} 1 & \text{if } k^{th} \text{ bus route is selected to have a sensor installed on} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The same approach was used for the scheduling of the sensors, for each bus route, a binary tuple of length d where d is the number of critical points in that bus route was used to encode the genotype $\bar{u_i} = (u_{i1}, ..., u_{id})$, such that gene u_k is used to represent whether the sensor would be turned on at the exact position of the k^{th} critical point on the i^{th} bus route. Thus, an array of length n of binary tuples is used to encode the scheduling of the sensors $\bar{U} = \{\bar{u_1}, ..., \bar{u_n}\}$, including bus routes that are not selected. The decision to include genes for non-selected bus routes will be explained and examined further later on in the paper.

$$u_k = \begin{cases} 1 & \text{if the } k^{th} \text{ critical point is selected} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Prior to the evaluation of the candidate, the critical points and their *observable* critical squares must first be calculated. In order to determine the set of critical squares that an arbitrary critical point can observe, we must first recall that let u be an arbitrary turn-on point in bus route b, from u we will only be able to observe a set of critical squares $C = \{A \mid d \in [\ell_{Ab}, r_{Ab}]\}$. The idea is to keep track of the current interval while looping orderly through the list of critical points. If an interval $[\ell_{Ab}, r_{Ab}]$ has not been escaped from, then every point that was being examined was within the interval and thus be able to observe critical square A corresponding to that interval. Assume that the critical points list is already sorted, procedure 2 describes the process of evaluating $2d$ critical points in a bus route.

The evaluation process of a candidate solution is as follows. Let R be a set of critical squares. For each binary encoding in the genotype of chosen bus routes, if $n_k = 1$ then we check the genotype of the chosen critical points of that bus route, if $u_k = 1$ then we add all of the *observable* critical squares from the corresponding critical point to R. After having looped through all the genotypes, R will have all the *observable* critical squares of the candidate solution.

The total search space when using the proposed candidate solution modeling is as follows:

$$O(\text{bus route combinations} \times \text{critical points combinations}^n) = O\left(\binom{n}{m} \times \binom{c}{k}^n\right).$$

Indeed storing non-selected bus routes' critical points genes increases the search space by a significant amount. However, this does not hinder the performance of the algorithms to any significant extent. In fact, keeping track of unused critical point genes of bus routes might improve the result of heuristic methods. As memorizing past genes of unused bus routes could

Procedure 1 Calculate intervals

```
1: procedure calculateIntervals(busMap)
2:     $criticalPoints \leftarrow newarray$
3:     for $i \leftarrow 1$ to $n$ do
4:         for $j \leftarrow 1$ to $c$ do
5:             for all continuous pairs of polypoints $p$ in bus route $i$ do
6:                 $l \leftarrow lineSegment(p_1, p_2)$
7:                 $x_1, x_2 \leftarrow null$
8:                 if $l$ intersects critical square $j$'s squircle at two points then
9:                     $(x_1, x_2) \leftarrow intersects(l, j)$
10:                if $p_1, p_2$ are both in critical square $j$'s squircle then
11:                    $(x_1, x_2) \leftarrow (p_1, p_2)$
12:                if $l$ intersects bus route $j$'s squircle at one point then
13:                    $x_1 \leftarrow intersects(l, j)$
14:                    if $p_1$ is in critical square $j$'s squircle then $x_2 \leftarrow p_1$
15:                    if $p_2$ is in critical square $j$'s squircle then $x_2 \leftarrow p_2$
16:                    else$x_2 \leftarrow x_1$
17:                if $d(x_1, p_1) < d(x_{2p1})$ then
18:                    $x_1.isBeginning = true$
19:                    $x_2.isBeginning = false$
20:                    $criticalPoints.add(x_1, x_2)$
21:                else
22:                    $x_1.isBeginning = false$
23:                    $x_2.isBeginning = true$
24:                    $ccriticalPoints.add(x_2, x_1)$
25:    Sort $2d$ critical points in increasing order
26:    $mergeIntervals(busMap)$    ▷ Critical points at poly-points that was produced by the
   same critical square are merged or deleted
```

Procedure 2 Critical points' observables critical squares calculation

```
1: procedure calculateCriticalPointsObservables()
2:     $currentObservablesCriticalSquares \leftarrow \emptyset$
3:     for $i \leftarrow 1$ to $2d$ do
4:         if i.isBeginning then
5:             $currentObservables.add(i.correspondingCriticalSquare)$
6:             $i.observables \leftarrow currentObservables$
7:         else
8:             $i.observables \leftarrow currentObservables$
9:             $currentObservables.remove(i.correspondingCriticalSquare)$
```

help in the evolution process as bus routes could switch their state between used and unused. Additionally, having knowledge of this beforehand, we could resolve to different strategies in our heuristic methods.

Procedure 3 Candidate solution evaluation

```
1: procedure evaluate(candidateSolution)
2:     $S \leftarrow \emptyset$
3:     $\forall$ square $A$, calculate the interval $[\ell_{Ab}, r_{Ab}]$
4:     Sort $2d$ critical points in increasing order
5:     for $i \leftarrow 1$ to $n$ do
6:         if Bus route $j$ is chosen then
7:             for $j \leftarrow 1$ to $2d$ do
8:                 if critical point j is chosen then
9:                     $S \leftarrow S \cup observables(j)$
10:    return $S$
```

The fitness of a candidate solution with bus route gene $\bar{b}$ and for the heuristic algorithm can be calculated as follows:

$$fitness(\bar{b}, \bar{U}) = size\ of\ \bigcup_{i=1}^{n} \bigcup_{j=1}^{d} observables(p_{ij})\ \ if\ u_{ij} = 1$$

with p_{ij} being the j^{th} critical point on the i^{th} bus route.

Thus, having a higher fitness value means the candidate solution can provide better performance and better coverage to the problem.

By altering the fitness function, we can use the same proposed heuristics algorithm to solve variations of the SOBP. For instance, by assigning a weight to each of the grid's squares, we can solve problem instances where some squares are more important to be monitored than others. Or by having a penalty function representing the cost to deploy sensors and removing the constraint of having a maximum of m sensors which might be another real-world problem.

3 Problem solving strategies

3.1 Greedy algorithm

Nguyen et al. has proposed a $\frac{1}{2}$ greedy approximation algorithm for the SOBP [18]. Their idea involves a dynamic programming function called *largestSet*, which outputs the largest possible set of current *observable* critical squares of an input bus route, given the information of the bus map. The *largestSet* function would be re-calculated for every bus route for every available sensor, deploying them in the process. After each iteration, the size of the set of $non-observable$ critical squares in the bus map shrinks due to the subtraction of the previously mentioned sensor deployment. The exact pseudo-code for the greedy algorithm presented in Nguyen et al.'s paper [18] is as follows:

Procedure 4 Greedy approximation algorithm

```
1: procedure solveSOBP($busMap$, $r$, $m$, $k$)
2:     $S \leftarrow \emptyset$
3:     for $i \leftarrow 1$ to $m$ do
4:         $X \leftarrow \emptyset$, $b \leftarrow 0$                    ▷ $b$: bus route to install sensor
5:         for $j \leftarrow 1$ to $n$ do
6:             if Bus route $j$ has not been chosen yet then
7:                 $Y \leftarrow largestSet(busMap, r, k, j)$
8:             else
9:                 $Y \leftarrow \emptyset$
10:            if $|Y| \geq |X|$ then
11:                $X \leftarrow Y$, $b \leftarrow j$
12:        $S \leftarrow S \cup X$
13:        Mark bus route $b$ as chosen
14:        Modify squares in $X$ to non-critical on $busMap$
15:    return $S$
```

The algorithm has been proven to have an efficiency of at least $\frac{1}{2}$ by its authors. It has also been tested with the Hanoi's real-world bus routes. The results of which yielded an astounding 72.68% guaranteed efficiency [18].

3.2 Genetic algorithm

A genetic algorithm is a probabilistic search heuristic that mimics the idea of natural selection and evolution in organisms inspired by Darwin's evolutionary theory [5]. In GAs, a population of candidate solutions is first defined, this population would go through repetitive operations to explore the problem's search space (recombination) and to examine the local candidates (mutation). The main driving force for the GA is the selection operation at the end of each repetition, which applies a selective pressure that decides which individuals will survive and participate in the next iteration (generation).

The process of a typical GA or Evolutionary Algorithm [23] is as follows:

1. Population initialization: Generate population with either random individuals or predefined values granted that a decent start point is discovered in advance. This population is usually evaluated before moving into the generation loop.

2. Generation loop: For a defined termination condition, for instance, the algorithm has acquired a quality solution, or the algorithm has reached a certain amount of evaluations, perform the following operations:

   - $selectParents(population)$: strategically select a set of parents from the population (usually the "fittest" are chosen, hence the phrase "survival of the fittest" in evolution theory). Here, the first selective pressure is applied upon the population.
   - $recombineParents(parents)$: selected parents go through predefined operations to recombine them to create offspring that have both (or some) characteristics and properties of their parents.
   - $mutateOffSpring(offspring)$: offspring that are created after the recombination process is then mutated, to not only explore the local search space, but to also introduce new gene sequences into the population. The mutation operation helps to ensure the genetic diversity of the population. In many evolutionary algorithms, this is the only way to introduce new genes to the population.
   - $evaluate(population)$: After the generation of candidate solutions, they need to be evaluated in order for the algorithm to be able to determine which are the superior individuals.
   - $selectSurvivors(population)$: another selection operation that eliminates those which are not selected in the next iteration. This applies the final selective pressure to the population.

Procedure 5 General genetic algorithm

```
1: $population \leftarrow initializePopulation()$
2: $evaluate(population)$
3: while termination condition is not satisfied do
4:     $parents \leftarrow selectParents(population)$
5:     $offspring \leftarrow crossoverParents(parents)$
6:     $offspring \leftarrow mutate(offspring)$
7:     $evaluate(offspring)$
8:     $population \leftarrow population + offspring$
9:     $selectSurvivors(population)$
```

Procedure 3 demonstrates the scheme of a typical genetic algorithm. Depending on the specific problem, operations in the GA might be altered accordingly to match the problem's constraint or to improve the performance of the algorithm. Our algorithm follows a general $(\mu+\lambda)$ EA model, that is, the characteristics of the survivors are chosen from both the parents (μ) and the offspring (λ) produced in the generation.

3.2.1 Recombination operation

The recombination process is greatly affected by the encoding and the data structure used to represent the candidate solutions. In consideration of the previously mentioned structure of genetic information, the recombination operation has to carry out on not only the bus route genotype but also the list of critical points genotypes in correspondence with it.

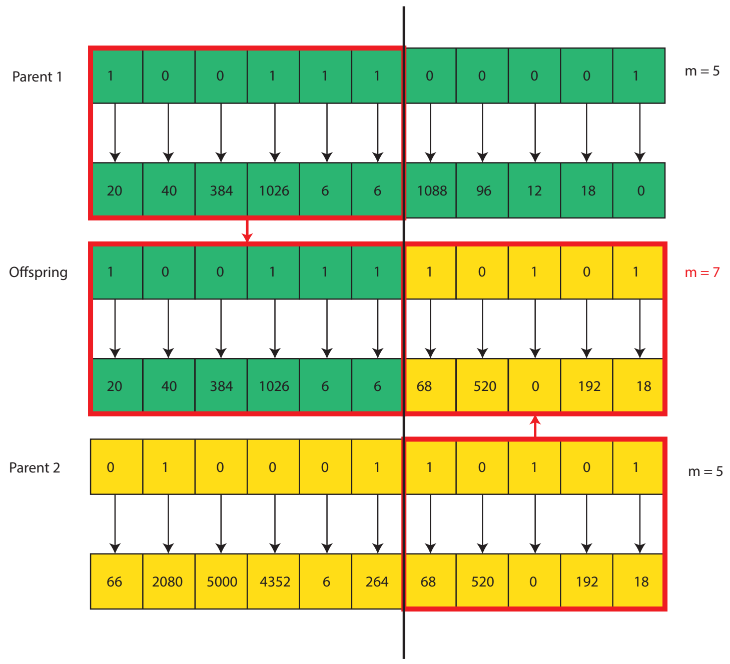

Figure 3.1: SOBP Crossover operation for $n = 11$, $m = 5$, $k = 2$.
Genes for the selection of critical points are encoded as integer values. These values can be converted to their binary counterpart, which, in our case the binary strings representing the selection of critical points. They are also attached to their corresponding bus route selection genes in a one to one relationship during the recombination process. In the sense that during the crossover operation, the bus route gene and the critical points gene for that bus route are moved together as one entity. The crossover operation can result in a candidate solution that violates the m amount of sensors constraint. Hence, a repair operation is essential after the recombine operation.

In our experiment, we proceeded with a simple single-point crossover with the crossover point being the middle of the bus route genotype binary string. However, this method could violate the m constraint of the problem (i.e. number of available sensors constraint). Illegal candidate solutions that allow too many sensors could be produced after the recombination process. To remedy this, in the case of a constraint violation, random genes (bits) in the bus route encoded binary string will be flipped to 0 until the offspring is a valid solution ($\sum b_i \leq m$). Figure 3.2 demonstrates the reparation process of a binary bit string.

Parents in the recombination process are selected via a tournament selection operator with bracket size 4.

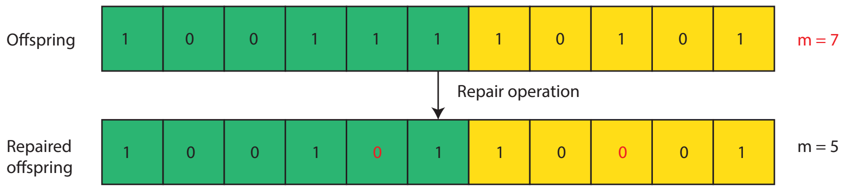

Figure 3.2: SOBP repair operation for $n = 11$, $m = 5$. Random 1 bits in the unqualified bus route string are flipped to 0 until the string is once again not violating any constraint.

3.2.2 Mutation operation

The nature of the SOBP and the data structures used to represent a candidate solution raised three problems in the designing of an appropriate strategy for the mutation process. Firstly, the SOBP has two constraints that directly affect the overall genotype (i.e. $\sum b_i = m$ and $\sum u_i = k$). Secondly, there would be $n+1$ encoded binary strings of genotypes (or 2 layers of genotypes) for both the selection of bus routes and the selection of critical points. Thirdly, there would be critical points genotypes that do not have any impact on the evaluation of a candidate.

For the bus route layer, we adopted a simple swap mutation for the mutation operator. This involves a mutation probability P_b that two bits swap their value. In our case, these two bits have to be 0 and 1 to ensure the constraint integrity of the candidate solution. The critical point genotypes for bus routes will not be moved or changed in this operation, as they correspond not to the value of the bus route genes but to the positions of the genes themselves.

For the critical points layer, we have to acknowledge that since there are n discrete genotypes the total search space for the second layer is quite large, especially when the number of critical squares c were to be increased. However, not all of the genes in this layer play a role in the calculation of the candidate solution's fitness value. We proposed a variant of the bit swap mutation operator, which involves a mutation probability P_d that a pair of bits are swapped if the bits are exactly two opposite bits and the critical points they are representing both belong to the same chosen bus route. To counter the problem of having a large search space, on the repeated success of the probability P_d, the operation for the second layer is again performed on the genotypes.

3.2.3 Selection operations

Taken into account the potentially huge search space of the problem, we implemented the tournament selection operator for the selection process. In which, the individual with the best performance is selected from a uniformly chosen subset of a pre-determined size from the population.

3.2.4 Population premature convergence

Premature convergence is a common but serious problem in many EAs, which can drastically hinder the performance of the algorithms [12]. Premature convergence occurs when the genes of some high fitness individuals become too dominant in the population, overcrowding the population with its neighbour or variants of themself, trapping the EAs in a local optimum. Theoretically, EAs can counter this by increasing the mutation probability. However, increasing the mutation probability might impair the inheritance properties through generations of EAs, turning it into a random search, rendering the algorithm ineffective. There exist multiple mechanisms to prevent premature convergence in EAs in current literature. There are methods that try to counter this problem by preventing the population from converging, namely fitness-sharing [22], parallel EAs [16] or cellular-structured EAs [7]. All of which requires extra effort to set up and maintain specific structures which in our case is not necessary or not applicable. On the other hand, there are techniques that deal with an already converged population by diversifying it through making the mutation chance dynamic in correspondence to some convergence indicator [9], or population injection (a technique that injects into the population a specific amount of random individuals to increase the genetic diversity) [15]. These techniques might introduce too much randomness into our population if the parameters are not set up correctly, thus weakening our algorithm.

In addition, the SOBP faces a very unique problem due to its multi-layer property which none of the previously mentioned techniques can address. That is, the candidate solution population could easily fall into premature convergence at the bus route layer. This is due to the fact that over multiple generations, the critical points genotypes of a few chosen bus routes could get significantly improved when compared to other bus routes' critical points genes, especially if the bus routes were not previously chosen in any generation. Thus, over the length of the algorithm, a few bus route genes could get overly superior because more work was put into developing them.

Hence, we employed several methods to reduce the probability of premature convergence occurrence. Firstly, a probability P'_d for the mutation of critical points in non-chosen bus routes which use the previously mentioned mutation operator is included in the algorithm. As these bus routes would not play any role in the evaluation of the candidate solution, mutating the critical points genes layer of them would not alter the fitness value of the result. However, without this operation, the candidate solution population could easily fall into premature convergence as described above. Secondly, after the survivor selection phase, a small number of candidate solutions from the population of removed candidate solutions in this generation would be uniformly selected and inserted back to the population. Despite the fact that the deployed tournament selection operator already had a role in countering premature convergence, it is still prone to premature convergence, especially when the population was starting to have signs of convergence.

3.2.5 Initial population

In GAs, an initial population has to be filled with randomly generated individuals or, when a decent start point is known, some pre-defined candidate solutions. For our experiment, we decided that we should evaluate the algorithm over 3 types of initial populations:

Table 3.1: Genetic algorithm's parameters

PARAMETERS		VALUE
GENETIC ALGORITHM		
Initial population size		300
Number of survivors	(μ)	150
Number of offspring	(λ)	150
Number of evaluations		60000
Crossover operation		Single-point crossover
Selection operation		Tournament selection
Tournament size (survivor selection)		5
Survivor selection strategy		$(\mu + \lambda)$
Mutation strategies		Repeating bit swap mutation
	(P_d)	0.11
Mutation probabilities	(P'_d)	0.04
	(P_b)	0.06

1. Random initial population: All individuals are generated randomly to satisfy the constraints of the SOBP.

2. Random initial population with best bus routes individual: a best bus routes individual is an individual that has its chosen bus routes being the bus routes that can separately observe the highest amount of critical squares in the list of bus routes without considering critical squares that have been covered by other routes. The set of total observable critical squares of the solution candidate might have a smaller size than the sum of observable critical squares of each bus route. This is because observable critical squares in these bus routes might duplicate each other. This individual can be obtained similarly to the greedy algorithm proposed by Nguyen et al. [18] by not taking into account chosen bus routes. It should be noted that our best bus routes individual always yields a worse amount of observable critical squares than the result of the greedy algorithm for obvious reasons. Despite the individual might have a lower quality than the greedy algorithm's result, the instance might provide the GA with some valuable insight about other possible high-performance bus routes without bias as in the case of the greedy algorithm. For instance, if there were two bus routes that share the same trajectory for most of their distance, only to diverge at the very end. This would result in two bus routes with very similar observables set, yet with the greedy algorithm, only one of the bus routes would be considered. Whereas in the best bus routes individual, both will be taken into account given they were qualified to be chosen.

3. Random initial population with Nguyen et al.'s [18] greedy algorithm's result individual: a randomly generated population together with the greedy algorithm's result.

3.3 Mixed Integer Programming model

In order to calculate the optimal solution for the problem, a Mixed Integer Programming model was developed. The objective of the model is to maximize the amount of covered critical squares by the sensors system. Let C_i be the *observable* status (1 means the square is covered, 0 means the square is not covered) of the i^{th} critical square, then the goal of this model is to:

$$maximize : \sum_{i=1}^{c} C_i$$

Let $\{u_{i1}, u_{i2}, ..., u_{io}\}$ be the list of binary sensor-activation-status of the critical points that can observe the critical square C_i. Then, if $C_i = 1$, there must be a least an element in the previously mentioned list which have its value equals to 1. Hence, we have the following constraint:

$$C_i \leq \sum_{j=1}^{o} u_{ij}.$$

The value of C_i can still take a value of 0 when a critical point that can cover it was chosen. However, since the objective of the model is to maximize the amount of *observable* critical squares, the solver will always prefer the value 1 of C_i than to its 0 counterpart.

There are two constraints to the problem. Firstly, there is the limitation of the number of deployable sensors. Let $\{b_1, b_2, ..., b_n\}$ be the set of sensor-installation-status of the bus routes (1 means the bus route has a sensor installed, 0 means the bus route has no sensors installed), then the constraint could be expressed as:

$$\sum_{i=1}^{n} b_i = m$$

Lastly, for each sensor, there is a constraint on the number of times that each sensor could be turned on. For each bus route, for the sake of the efficiency of the problem, we only take into account the critical points as the activate points for the sensors. Let b_i be the sensor-installation-status of the i^{th} bus route, and $\{u_{i1}, u_{i2}, ..., u_{id}\}$ be the binary sensor-activation-status of the critical points in that bus route. Then the constraint could be formulated as:

$$\sum_{j=1}^{d} u_{ij} = b_i \times k,\ if\ d > k$$
or
$$\sum_{j=1}^{d} u_{ij} = b_i \times d,\ if\ d \leq k$$

In these two equations, d and k act as coefficients of b_i, thus preserving the linear property of our model. They enforce two constraints. Firstly, the critical points of the bus route can only have its sensor-activation-status be 1 if and only if a sensor is installed on that bus route. Secondly, if such a sensor is installed on the bus route, then the sum of the sensor-activation-status of every critical point in the bus route must be equal or less than (in the case of $d < k$) k.

The MIP model is as follows:

$$\begin{cases} maximize : \sum_{i=1}^{c} C_i \\ \sum_{i=1}^{n} b_i = m \\ C_i \leq \sum_{j=1}^{o} u_{ij} \\ \sum_{j=1}^{d} u_{ij} = b_i \times k,\ if\ d > k\ for\ i \leftarrow 0\ to\ n \\ \sum_{j=1}^{d} u_{ij} = b_i \times d,\ if\ d \leq k\ for\ i \leftarrow 0\ to\ n \end{cases} \tag{3.1}$$

The MPSolver provided by Google's open-source OR-Tools was used for obtaining the optimal solution through our previously proposed Mixed Integer Programming model.

4 Experimental results

4.1 Experiment settings

4.1.1 Experiment goals

As SOBP is an $\mathcal{NP}$-hard problem, heuristic approaches are necessary to find a good solution for the problem. We conducted this experiment to answer the following research questions:

RQ1. Can the proposed genetic algorithm reliably get results that are at least equal or better than the approximation algorithm proposed by Nguyen et al. [18]?

RQ2. Can including the best bus routes individual in the starting population improve the performance of the genetic algorithm so that it can reliably get results that are at least equal or better than the approximation algorithm proposed by Nguyen et al. [18]?

RQ3. Can making the greedy algorithm [18]'s result a start point improve the performance of the genetic algorithm so that it can reliably get results that are at least equal or better than the approximation algorithm proposed by Nguyen et al. [18]?

From these research questions, the following hypotheses can be derived:

$H1_1$ The mean of the number of observable critical squares of the best solutions found by the genetic algorithm is greater or equal to the mean of the greedy algorithm's results.

$H2_1$ The mean of the number of observable critical squares of the best solutions found by the genetic with the best bus routes individual as a start point is greater or equal to the mean of the greedy algorithm's results.

$H3_1$ The mean of the number of observable critical squares of the best solutions found by the genetic with the greedy algorithm's result individual as a start point is greater or equal to the mean of the greedy algorithm's results.

$H4_1$ The GA with the greedy algorithm's result in the starting population can improve the solution found by the GA with the best bus routes individual in its starting population.

These hypotheses will later be statistically tested against each other via the Wilcoxon Signed Rank Test [6][24].

4.1.2 Test data generation process

The bus routes and critical squares map instances used in this experiment are the same data sets used in Nguyen et al. [18] paper to evaluate SOBP through the mentioned greedy algorithm. These data sets describe the real bus routes in Hanoi, Vietnam. Additionally, we also include maps of actual bus routes in Ho Chi Minh City, Vietnam, which consists of 40 bus routes. The topology of the bus routes in test cases is kept constant and scaled according to the size of the map ($p \times q$). The data set for Ho Chi Minh City was created using the same generation method as its Hanoi counterpart. This generation process can be described as follows.

To generate the test cases, areas where we examine bus routes must first be chosen. This area is later on divided into a grid of $p \times q$ squares. In Hanoi test cases, an area of roughly $20km \times 30km$ was used. And in Ho Chi Minh City test cases, an area of size approximately $17km \times 22km$ was used. Then a set of c squares were chosen randomly to be the set of critical squares. The radius r is scaled accordingly to the edge size of a square of that particular map. Hence, each test case is defined by 4 parameters p, q, c and r.

4.1.3 Performance evaluation methods

In every test case, we execute 2 algorithms (Nguyen et. al.'s greedy approach [18], GA) to try and tackle the SOBP. In the case of probabilistic algorithms, we also include executions of the algorithms with the starting point being the bus map's greedy algorithm result. In test cases with a low amount of critical squares ($c \leq 50$), we also compute the instances' optimal solution through the MIP model. Due to the computational effort required to complete our proposed heuristic methods, we only consider a small number of (m, k) pairs, such that $m \leq 20$ and $k \leq 20$.

Let S be the best set of critical squares of the algorithm that is being examined could find and O be the optimal solution of the problem instance. The efficiency of the algorithms should be calculated as follows:

$$\text{Efficiency} = \frac{|S|}{|O|}$$

However, since SOBP is an $\mathcal{NP}$-hard problem, we cannot find its optimal result in polynomial time. Using the proposed MIP model, we can find the optimal solutions for small test cases (i.e. $c \leq 50$), otherwise, it would take too much time and resources to compute the optimal solutions for other test cases. Hence, to evaluate heuristic algorithms results in larger test cases, we will be comparing the result to the Greedy approximation algorithm's results G proposed by Nguyen et al. [18]. The improvement is calculated as follows:

$$\text{Improvement} = \frac{|S| - |G|}{|G|}$$

Therefore, in large test cases, the result for the efficiency will be blank. However, the improvements made by heuristic algorithms are always measured and shown.

In addition, we also record our program's running time in every problem instance. The experiment was implemented in Java and executed on a server with Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz CPU (16 cores, 32 threads), and 32GB RAM. Graphs, evaluations and statistics of the experiment were done in Python3 with the well-known SciPy [14] and Matplotlib [10].

4.1.4 Impacts of parameters on the efficiency and improvement over the greedy algorithm

It was noted by Nguyen et al. [18] that for the SOBP that the efficiency of the greedy algorithm follows a specific pattern. That is, with a certain lower bound for m and k, the efficiency of the greedy algorithm is usually perfect. The exact value for the bound, however, depends greatly on the test cases' parameters and specifications. This pattern is also noticed in our experiment, through the observation of where the heuristic methods could make an improvement over the greedy algorithm usually matches this pattern. The result for other test cases follows a rather similar pattern, with slight differences in the possible value of the bounds of m and k. Following is the heatmap of the test case with $p = 42, q = 50, r = 1$ and $c = 50$:

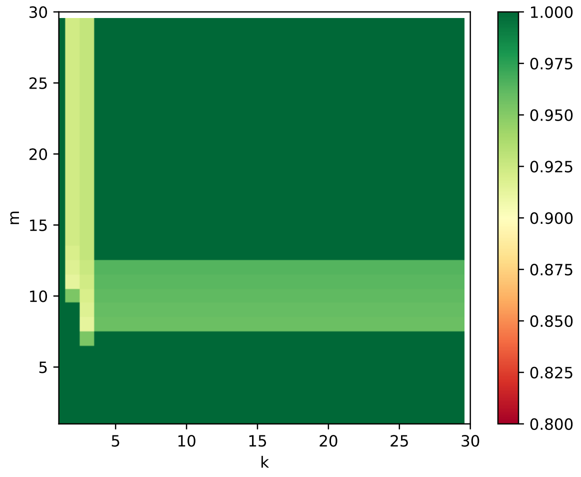

Figure 4.1: Greedy algorithm efficiency heatmap when $(p, q, c, r) = (42, 50, 50, 1)$. Efficiency was calculated by taking the result of the greedy algorithm and divided it by the optimal solution found by the MIP solver

Through Figure 4.1, we can observe that when m is kept constant and k changes, the greedy algorithm's efficiency is to be altered with a minimal amount of deviation for all values of k. This is due to the fact that there are a lot more critical points than the amount of necessary turn-on points for the sensor to cover every critical square that the bus route could cover. Such that in the greedy algorithm, the best bus routes in some specific bus route choosing iterations with the $largestSet$ procedure would eventually come to a halt in the growth of its observable squares set when k were to be increased. However, none of the other bus routes could match these bus routes' performance. Consequently, the same pattern for choosing bus routes is, to some degree, repeated even when k was increased.

Alternatively, when k is fixed, there exists an interval $[m_{k1}, m_{k2}]$ such that if $m \notin [m_{k1}, m_{k2}]$ then the efficiency of the greedy algorithm would be 100%. This has also been observed and explained by Nguyen et. al. [18], this is mainly because a candidate solution can only cover at most $c - e$ squares with e being the amount of critical squares that are not observable by any bus route. When $c - e$ is obtained by a set of (m, k), increasing m or k will not increase

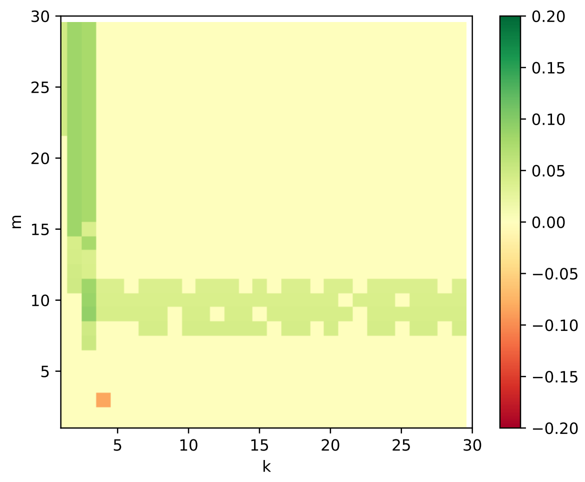

Figure 4.2: Improvement of the proposed GA with random start points over the Greedy Algorithm heatmap when $(p, q, c, r) = (42, 50, 50, 1)$.

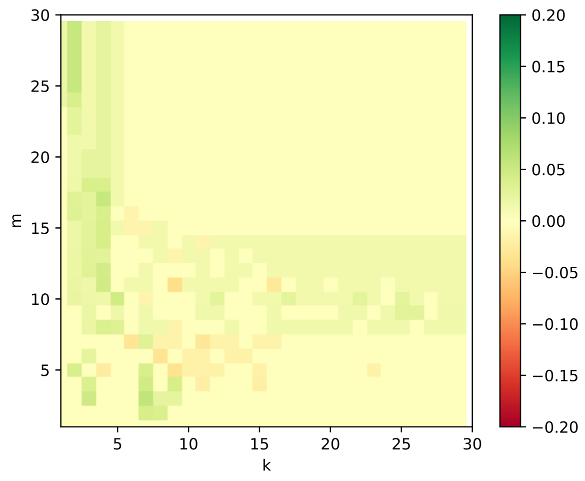

Figure 4.3: Improvement of the proposed GA with random start points over the Greedy Algorithm heatmap when $(p, q, c, r) = (25, 30, 100, 1)$.

the amount of observable critical squares, hence the interval $[m_{k1}, m_{k2}]$. The interval's bounds can vary depending on its problem instance.

A known weakness for heuristic approaches is that they do not know whether a candidate solution is optimal for the problem instance or not. As a consequence, the iteration process of these algorithms would keep on running even if the optimal solution has been found. In the SOBP, we can achieve a lower bound for the optimal solution [18] for each pair of (m, k) before proceeding with the execution of any heuristic algorithm. Which might help the algorithms terminating themselves much sooner since they have found the optimal solution for the instance. However, in our experiment, this check is turned off so we could get a better view of the parameters' effects on the runtime of the algorithms.

The running times of heuristic algorithms are greatly affected by the parameters m, k, r and the density of critical squares in the particular instance (i.e. $\frac{c}{p \times q}$). Increasing m or k would raise the effort needed in the evaluation operation within these heuristic approaches, thus increasing the overall runtime. Meanwhile, a higher critical squares density or a higher radius r would likely mean that the critical points genotypes would have more genes, hence, increasing the necessary computational resources to proceed with the operations on the candidate solutions. For example, in Figure 4.4 and Figure 4.5, both suggest that as m and k increase, the runtime would be higher for the instance of execution. And since the density of critical points of $(p, q, c, r) = (42, 50, 200, 1)$ is twice as high as the density of $(p, q, c, r) = (42, 50, 100, 1)$, the former took significantly longer to compute.

The MIP model solver was set to run on every bus map instance, however a 2000 seconds termination condition for the solver was set. If the condition gets triggered, the solver would be set to stop for that particular bus map. The termination condition was necessary due to the fact that the solver could take up to days to solve the instance with $(p, q, c, r) = (42, 50, 100, 1)$.

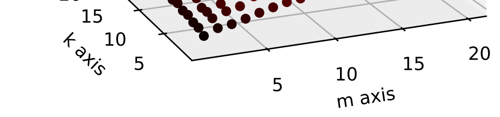

Figure 4.4: GA's runtime scatter plot on instance $(p, q, c, r) = (42, 50, 100, 1)$.

4.2 Results

In this section, the results of the experiment are presented. They are divided into multiple tables for every algorithm, each of which contains the runtime information and the assessment of the solution quality. For each pair of the bus map and the algorithm, the following columns are displayed:

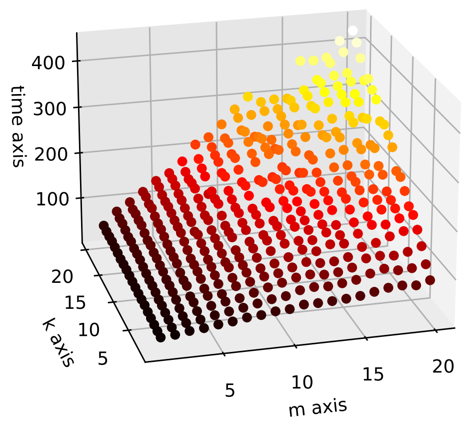

Figure 4.5: GA's runtime scatter plot on instance $(p, q, c, r) = (42, 50, 200, 1)$.

1. Avg. Eff.: The average efficiency of the algorithm over every pair of (m, k) that was used for this instance of bus map.
2. Avg. Imp.: The average improvement of the algorithm over the greedy algorithm for every pair of (m, k). This value can be negative if the algorithm yields a worse result than the greedy algorithm.
3. Avg. Imp.*: The average improvement of the algorithm over the greedy algorithm for every pair of (m, k) such that in the particular execution the greedy algorithm's result is less than the found optimal result. This value can be negative if the algorithm yields a worse result than the greedy algorithm.
4. %R: Short for percentage of regression. The ratio of pairs of (m, k) that when applied to the algorithm and the test instance, yields a lesser result than the greedy algorithm over the total number of (m, k) pairs.
5. %I: Short for percentage of improvement. The ratio of pairs of (m, k) that when applied to the algorithm and the test instance, yields a better result than the greedy algorithm over the total number of (m, k) pairs.
6. %I*: Short for percentage of improvement. The ratio of pairs of (m, k) that when applied to the algorithm and the test instance, yields a better result than the greedy algorithm over the number of (m, k) pairs when applied to the algorithm and the test instance, the greedy algorithm yields a worse result than the found optimal solution.
7. Avg. RT: The average runtime of the algorithm over all pairs of (m, k)

8. Best Sol. Found Highest RT: for each execution, the best solution found by the algorithm is not usually found at the last generation, but rather somewhere in between the execution time and the start time. This property shows the longest time it took in every execution to find the best solution.

9. Best Sol. Found Avg. RT: The average runtime for every execution to find the best solution.

It was pointed out by Nguyen et al. [18] that an edge of any given square in the 42×50 grid of Hanoi is already significantly smaller than the radius r of a real sensor. In our case, the same can be said toward the Ho Chi Minh City's 30×40 configuration, since the Ho Chi Minh bus map is smaller in size when compared to Hanoi bus map. Hence, the configuration of $(p, q) = (42, 50)$ for Hanoi and $(p, q) = (30, 40)$ for Ho Chi Minh City can be considered as sufficiently large enough for them.

4.2.1 Result discussion

In order to test our hypotheses in Section 4.1.1, we utilize the Wilcoxon Signed Rank Test [6] [24] to obtain the appropriate statistics and p-value. For each test, the best solution found by each algorithm is gathered and paired together by their corresponding execution instances (i.e. the same bus maps and the same pair of (m, k)). Table 4.4 displays the results after the tests had been conducted, where the p-value is shown for each comparison made.

Within the confidence interval of 95%, Table 4.4 depicts that only in 16 out of 34 cases, the GA can generate statistically significant better results than Nguyen [18]'s greedy algorithm for SOBP. However, it can be observed that the GA's performance is significantly worse against instances with higher critical square density. The results in table 4.1 confirm this, as in the Ho Chi Minh bus map, with $(p, q, c, r) = (30, 40, 200, 2)$, the GA failed to reach the greedy algorithm's performance 85.75% of the time and on average decrease the overall quality of the solutions by 4.5%. Hence, we can not reject hypothesis $H1_0$, therefore hypothesis $H1_1$ is not accepted.

Table 4.4 also shows that out of a total of 34 cases, 26 of which suggest that with over 95% confidence, the GA with the best bus routes individual as a start point is superior to or at least as good as the greedy algorithm in terms of finding good solutions for the SOBP. If examined carefully, most of the cases where the test could not gather statistically significant results are either the optimal solution has been found by both algorithms (Efficiency = 100%) or the GA had found all the optimal solutions but the number of instances with improvement over the greedy algorithm was not large enough. However, when we raise the amount of critical squares in each bus map, the performance of the algorithm quickly deteriorates. A prime example of this is the cases where $(p, q, r) = (42, 50, 500)$, the amount of lower performance results (when compared to the greedy algorithm) spikes tremendously, reaching 56.75% of instances where the GA performed worse than the greedy algorithm. Despite having achieved excellent results in smaller test cases, hypothesis $H2_1$ can not be accepted due to the algorithm and the initial population's low performance in larger test cases. This can also be confirmed via the average results from table 4.2, wherein every bus map, the algorithm the average improvement over the greedy algorithm results are always positive except for only three instances where there were high critical squares density combined with high sensor radius. Suggesting that the algorithm

Table 4.1: Experimental results of the GA with randomized initial populations

$p \times q$	c	r	Avg. Eff.	Avg. Imp.	Avg. Imp.*	%R	%I	%I*	Avg. RT	Highest Best Sol. Found RT	Avg Best Sol. Found RT
HN 10×12	10	1.0	100%	0%	0%	0%	0%	0%	52.8s	0.65s	0.002s
		2.0	100%	0.1%	18.7%	0%	1%	100%	76.7s	2.38s	0.02s
	20	1.0	99.9%	0.08%	5.8%	0%	1%	66.6%	126.1s	38.0s	0.16s
		2.0	99.9%	0.05%	3.3%	0.5%	1.75%	87.5%	187.6s	23.8s	0.11s
	25	1.0	99.9%	-0.03%	1.8%%	1%	0.5%	28.5%	138.5s	29.9s	0.30s
		2.0	99.7%	-0.05%	5.7%	3.75%	1.5%	66.6%	246.2s	82.0s	0.61s
HN 30×36	25	1.0	99.8%	2.4%	2.6%	0.25%	32%	96.9%	41.2s	35.6s	0.38s
		2.0	99.6%	0.17%	1.0%	2%	5.25%	56.7%	51.8s	37.8s	0.41s
	50	1.0	99.7%	0.7%	5.3%	1.25%	12.75%	83.6%	57.6s	55.47s	1.26s
		2.0	99.2%	1.5%	3.2%	3.25%	37.75%	79.0%	87.3s	80.3s	5.11s
	100	1.0	n/a	-0.03%	n/a	24.25%	9.5%	n/a	112.1s	143.4s	16.3s
		2.0	n/a	0.07%	n/a	11.75%	54.75%	n/a	165.4s	180.7s	24.5s
HN 42×50	25	1.0	99.6%	-0.1%	4.6%	3.5%	1.75%	70%	37.9s	36.0s	0.22s
		2.0	99.8%	1.6%	3.8%	0.25%	26.75%	94.6%	43.51s	35.5s	0.77s
	50	1.0	99.3%	0.73%	2.7%	0.25%	15.25%	55.4%	51.8s	49.4s	0.85s
		2.0	n/a	0.6%	n/a	4.75%	27%	n/a	69.3s	58.0s	3.97s
	100	1.0	n/a	0.3%	n/a	5.25%	12.5%	n/a	79.7s	92.03s	5.89s
		2.0	n/a	1.0%	n/a	16.75%	47.25%	n/a	98.7s	142.9s	16.5s
	200	1.0	n/a	-0.1%	n/a	28.5%	38.25%	n/a	156.7s	221.4s	43.79s
		2.0	n/a	-1.2%	n/a	37.75%	45%	n/a	221.3s	404.0s	43.0s
	500	1.0	n/a	-1.0%	n/a	94.5%	0%	n/a	254.7s	504.6s	88.35s
		2.0	n/a	-1.3%	n/a	95%	0%	n/a	365.1s	1324.1s	100.3s
SG 30×40	10	1.0	100%	0%	0%	0%	0%	0%	28.9s	0.006s	$\ll$ 0.001s
		2.0	100%	0%	0%	0%	0%	0%	34.8s	15.5s	0.04s
	25	1.0	99.9%	2.0%	6.6%	0%	30.5%	99.18%	51.0s	41.9s	0.72s
		2.0	99.9%	1.5%	7.4%	0.25%	20.25%	97.5%	66.8s	38.4s	0.36s
	50	1.0	n/a	0.7%	n/a	1.5%	11.75%	n/a	76.6s	45.7s	1.35s
		2.0	n/a	1.4%	n/a	3%	42.75%	n/a	120.44s	88.3s	3.64s
	100	1.0	n/a	1.1%	n/a	8.5%	58.5%	n/a	145.5s	190.4s	11.3s
		2.0	n/a	1.9%	n/a	5.75%	65.75%	n/a	204.1s	181.8s	9.7s
	200	1.0	n/a	-0.2%	n/a	40%	28.25%	n/a	211.0s	474.2s	34.7s
		2.0	n/a	-4.5%	n/a	85.75%	1.5%	n/a	339.4s	513.0s	34.1s
	300	1.0	n/a	-4.0%	n/a	77.5%	7.75%	n/a	256.5s	404.3s	43.5s
		2.0	n/a	-7.7%	n/a	94.0%	0%	n/a	446.8s	909.5s	41.5s

Table 4.2: Experimental results of the GA with best bus route individual in the initial population

$p \times q$	c	r	Avg. Eff.	Avg. Imp.	Avg. Imp.*	%R	%I	%I*	Avg. RT	Highest Best Sol. Found RT	Avg Best Sol. Found RT
HN 10×12	10	1.0	100%	0%	0%	0%	0%	0%	22.8s	$\ll$ 0.001s	$\ll$ 0.001s
		2.0	100%	0.1%	18.7%	0%	1%	100%	76.9s	0.01s	$\ll$ 0.001s
	20	1.0	99.9%	0.04%	3.2%	0.75%	0.75%	50%	126.2s	7.95s	0.02s
		2.0	100%	0.1%	5.7%	0%	2%	100%	188.5s	55.2s	0.16s
	25	1.0	99.9%	0.08%	5.4%	0.25%	1.75%	100%	138.9s	57.4s	0.47s
		2.0	99.8%	0.04%	4.7%	1.5%	1.25%	55.5%	247.4s	10.2s	0.07s
HN 30×36	25	1.0	99.7%	2.3%	7.2%	0%	30.25%	91.6%	40.2s	32.2s	0.45s
		2.0	99.7%	0.2%	3.1%	0.25%	5%	54.0%	51.0s	33.0s	0.24s
	50	1.0	99.9%	0.9%	6.0%	0%	14.5%	95.0%	56.4s	50.7s	0.6s
		2.0	99.1%	1.4%	3.0%	2.25%	36.5%	76.4%	83.3s	88.8s	4.7s
	100	1.0	n/a	0.4%	n/a	3.25%	19.75%	n/a	109.9s	145.2s	5.7s
		2.0	n/a	1.1%	n/a	0.5%	50%	n/a	165.0s	300.1s	21.6s
HN 42×50	25	1.0	99.9%	0.1%	6%	0%	2.25%	90.0%	37.4s	0.34s	0.003s
		2.0	99.8%	1.6%	5.8%	0.25%	26.5%	93.8%	43.51s	42.0s	0.6s
	50	1.0	99.5%	0.9%	3.4%	0.25%	19.5%	70.9%	50.8s	48.39s	0.94s
		2.0	n/a	1.0%	n/a	2%	32%	n/a	67.5s	44.6s	1.4s
	100	1.0	n/a	0.6%	n/a	0.75%	16.25%	n/a	77.9s	86.8s	2.88s
		2.0	n/a	1.8%	n/a	4.25%	51.75%	n/a	98.3s	120.4s	8.4s
	200	1.0	n/a	0.5%	n/a	12%	45%	n/a	158.6s	329.2s	29.6s
		2.0	n/a	1.2%	n/a	8%	63%	n/a	223.9s	504.3s	43.6s
	500	1.0	n/a	-0.7%	n/a	49.25%	18.75%	n/a	265.8s	497.7s	30.1s
		2.0	n/a	-1.3%	n/a	56.75%	11.5%	n/a	387.1s	772.9s	33.5s
SG 30×40	10	1.0	100%	0%	0%	0%	0%	0%	28.8s	0.01s	$\ll$ 0.001s
		2.0	100%	0%	0%	0%	0%	0%	34.8s	15.5s	0.04s
	25	1.0	99.9%	2.0%	6.6%	0%	30.75%	100%	49.1s	34.8s	0.36s
		2.0	99.9%	1.5%	7.2%	0%	19.5%	93.9%	65.5s	17.8s	0.14s
	50	1.0	n/a	0.9%	n/a	0	12.5%	n/a	73.9s	60.7s	0.61s
		2.0	n/a	1.6%	n/a	1%	44.75%	n/a	119.6s	76.6s	2.34s
	100	1.0	n/a	1.8%	n/a	1.25%	64%	n/a	142.9s	107.1s	6.9s
		2.0	n/a	2.5%	n/a	0.75%	73%	n/a	204.1s	126.4s	6.7s
	200	1.0	n/a	1.4%	n/a	18.75%	42%	n/a	212.3s	458.9s	18.8s
		2.0	n/a	0.8%	n/a	8%	40%	n/a	351.7s	402.3s	13.4s
	300	1.0	n/a	2.2%	n/a	18.5%	63.75%	n/a	262.9s	488.3s	29.9s
		2.0	n/a	-0.2%	n/a	46.25%	36.0%	n/a	445.2s	814.8s	42.3s

Table 4.3: Experimental results of the GA with greedy algorithm's result in the initial population

$p \times q$	c	r	Avg. Eff.	Avg. Imp.	Avg. Imp.*	%R	%I	%I*	Avg. RT	Highest Best Sol. Found RT	Avg Best Sol. Found RT
HN 10×12	10	1.0	100%	0%	0%	0%	0%	0%	48.5s	$\ll 0.001s$	$\ll 0.001s$
		2.0	100%	0.1%	18.75%	0%	1%	100%	74.0s	0.73s	0.001s
	20	1.0	99.9%	0.07%	4.7%	0%	0.75%	50%	94.4s	19.1s	0.05s
		2.0	99.9%	0.08%	4.2%	0%	1.5%	75%	150.4s	6.86s	0.03s
	25	1.0	99.9%	0.08%	4.7%	0%	1.5%	85.7%	112.4s	16.2s	0.06s
		2.0	99.9%	0.1%	4.7%	0%	1.25%	55.5%	211s	32.19s	0.08s
HN 30×36	25	1.0	99.8%	2.4%	7.2%	0%	30.75%	93.1%	38.6s	36.6s	0.67s
		2.0	99.9%	0.4%	5.1%	0%	8%	86.4%	48.6s	24.4s	0.18s
	50	1.0	99.9%	0.9%	5.9%	0%	14.75%	96.7%	55.4s	33.6s	0.47s
		2.0	99.1%	1.4%	2.9%	0%	32.5%	68.0%	82.61s	81.91s	5.27s
	100	1.0	n/a	0.4%	n/a	0%	20.5%	n/a	108.2s	99.9s	2.0s
		2.0	n/a	1.3%	n/a	0%	62.75%	n/a	158.6s	171.1s	10.2s
HN 42×50	25	1.0	100%	0.1%	6.8%	0%	2.5%	100%	36.36s	29.2s	0.13s
		2.0	99.4%	1.2%	4.4%	0%	19.75%	69.9%	40.6s	23.1s	0.15s
	50	1.0	99.4%	0.7%	n/a	0%	14.25%	n/a	48.0s	48.0s	1.16s
		2.0	n/a	0.9%	n/a	0%	30%	n/a	68.1s	66.1s	0.99s
	100	1.0	n/a	0.6%	n/a	0%	15%	n/a	79.6s	63.3s	1.73s
		2.0	n/a	1.7%	n/a	0%	51.75%	n/a	96.74s	103.73s	8.64s
	200	1.0	n/a	0.9%	n/a	0%	46%	n/a	156.0s	297.85s	17.63s
		2.0	n/a	1.4%	n/a	0%	60.5%	n/a	221.7s	457.7s	42.17s
	500	1.0	n/a	0.2%	n/a	0%	28%	n/a	262.5s	503.1s	21.5s
		2.0	n/a	0.1%	n/a	0%	22.5%	n/a	383.2s	330.9s	7.4s
SG 30×40	10	1.0	100%	0%	0%	0%	0%	0%	28.2s	$\ll 0.001s$	$\ll 0.001s$
		2.0	100%	0%	0%	0%	0%	0%	32.7s	$\ll 0.001s$	$\ll 0.001s$
	25	1.0	99.9%	2.0%	6.6%	0%	30.75%	100%	49.1s	34.8s	0.36s
		2.0	99.9%	1.5%	7.4%	0%	20.25%	97.5%	64.59s	32.24s	0.39s
	50	1.0	n/a	0.8%	n/a	0%	12.5%	n/a	72.3s	20.82s	0.26s
		2.0	n/a	1.7%	n/a	0%	46.75%	n/a	114.4s	82.9s	2.97s
	100	1.0	n/a	1.7%	n/a	0%	57%	n/a	139.4s	162.82s	11.1s
		2.0	n/a	2.6%	n/a	0%	73.75%	n/a	210.5s	237.4s	9.25s
	200	1.0	n/a	2.0%	n/a	0%	44.75%	n/a	208.7s	205.2s	10.9s
		2.0	n/a	1.0%	n/a	0%	36.25%	n/a	344.0s	396.4s	9.62s
	300	1.0	n/a	2.2%	n/a	0%	70.5%	n/a	265.7s	370.9s	20.3s
		2.0	n/a	0.6%	n/a	0%	43.5%	n/a	455.6s	324.4s	7.9s

Table 4.4: Statistics and p-value of Paired Wilcoxon Signed Rank Test for testing hypotheses $H1_1$, $H2_1$ and $H3_1$

$p \times q$	c	r	GA vs Greedy algorithm	GA (Best routes) vs Greedy algorithm	GA (Greedy result) vs Greedy algorithm	GA (Greedy result) vs GA (Best routes)
			p-value	p-value	p-value	p-value
HN 10 × 12	10	1.0	n/a	n/a	n/a	n/a
		2.0	0.089	0.089	n/a	n/a
	20	1.0	0.029	0.041	0.034	0.32
		2.0	0.124	0.002	0.007	0.92
	25	1.0	0.84	0.016	0.007	0.5
		2.0	0.76	0.28	0.02	0.28
HN 30 × 36	25	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.29
		2.0	0.007	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
	50	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.79
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.72
	100	1.0	0.99	$\ll 0.01$	$\ll 0.01$	0.07
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
HN 42 × 50	25	1.0	0.93	0.001	$\ll 0.01$	0.15
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.99
	50	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.99
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.68
	100	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.67
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.87
	200	1.0	0.02	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
		2.0	0.77	$\ll 0.01$	$\ll 0.01$	0.22
	500	1.0	1	1	$\ll 0.01$	$\ll 0.01$
		2.0	1	1	$\ll 0.01$	$\ll 0.01$
SG 30 × 40	10	1.0	n/a	n/a	n/a	n/a
		2.0	n/a	n/a	n/a	n/a
	25	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.98
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.04
	50	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.71
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
	100	1.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	0.99
		2.0	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
	200	1.0	0.99	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
		2.0	1	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$
	300	1.0	1	0.96	$\ll 0.01$	$\ll 0.01$
		2.0	1	1	$\ll 0.01$	$\ll 0.01$

in conjunction with the best bus routes initial population performs well in smaller test cases, whereas in larger test cases, the performance deteriorates significantly.

Unlike the other 2 discussed initial populations, the population that includes the greedy algorithm's result when used for our proposed GA will undoubtedly always ensure that the best candidate solution found is either greater or equal to the greedy algorithm's. This can be explained by the tournament selection operator was employed in our GA, which will always guarantee that the best solution will survive for the next generation. However, despite having this advantage, starting with an overly superior individual might potentially make the GA susceptible to premature convergence (see Section 3.2.4). Even though the greedy algorithm's results might have a better fitness value than the best bus routes individual, including this in the starting population may not improve the performance of the GA. Table 4.4 suggests that there are only 10 out of 34 cases where the greedy algorithm's result population was able to outperform the best routes individual population in the GA as initial populations. The results in table 4.3 and table 4.2 confirm this, despite the GA with the best bus routes individual had instances where it performed worse than the greedy algorithm, it was still able to yield better improvement to the greedy algorithm than the GA with the greedy solution in the initial population in some smaller test cases. Hence, we can therefore not reject hypothesis $H4_0$. However, when we consider test cases where there was higher critical squares density, the GA with the greedy solution tends to perform significantly better than other proposed initial populations. This perfectly rectifies the previous initial population strategies' weakness. Not only so, the algorithm has also demonstrated its capability to improve the greedy algorithm's result in almost every bus maps, averagely providing positive improvements over the greedy algorithm where it had not yet obtained the optimal results. In every case where the greedy algorithm had not found all the optimal solutions, the GA with the greedy algorithm's results in its initial population was able to improve it with a confidence level of more than 95%. We can therefore accept hypothesis $H3_1$.

In all instances of GAs, it can be observed that the higher the critical squares density, the lower the performance of the algorithm. This can be recognized most clearly through the increasing percentage of worse results when compared to the greedy algorithm. A possible explanation for this is that as the density of critical squares increases, so does the length of genotypes at the critical points layer. However, the length of the genotypes at the bus route layer is kept constant. Thus, the imbalance between the probability to mutate the bus route genes and the probability to mutate the critical points becomes more serious as the critical squares grow, ultimately hindering the GA. This could be rectified through the use of different sets of parameters. However, finding the optimal set of parameters is in itself a very difficult problem of finding a good combination of discrete variables. Another possible cause for this might be the multi-dimensional property of the SOBP. As the critical squares density increases, the search space is also increased exponentially, making the optimal/good solutions become more and more elusive to search algorithms. This is known as the curse of dimensionality, first described by Bellman [3]. Evolutionary algorithms (in our case, GAs) unlike other optimization algorithms, have been found to work in high dimensional problems [13]. However, fine-parameters-tuning is required in order to achieve such success, which brings us back to our initial proposed problem.

In smaller test cases where the optimal solutions can be found in a timely manner, our GA has a tendency to perform quite well. With proposed initial start points, the GA tends to achieve very high efficiency and is highly effective in improving the greedy algorithm's solution. Despite the better solutions that the GA may provide, the process for which to be found is extremely time and resource consuming. While the greedy algorithm may only take seconds to compute such a solution, our proposed GA may take hundreds if not thousands of seconds to reach its termination condition. However, in our case, the termination condition is just merely a high-enough arbitrary number (number of evaluations) to ensure the performance of the GA. Thus, the number of evaluations termination condition could be altered to match the specific test case. As tables 4.1 and 4.2 suggest, the average time for an instance to find its best candidate solution is far lower than the average time it took to run the whole algorithm. Depending on each test case's specific parameters, more effective termination conditions should be considered. This is one of the GA's and many other heuristic methods' inherent weaknesses, they are not aware if their current solutions are optimal or not [20]. Thus, deciding when to stop for these algorithms is again a problem on its own. However, in the SOBP, Nguyen et al. [18] proposed a lower bound for the optimal solution, which in our case could be utilized as one of the stop conditions for the GA.

Regarding the runtime of the algorithms, the greedy algorithm outperforms the other proposed heuristic methods by an enormous margin. Consequently, it is recommended that the greedy algorithm be used for execution instances with some specific m and k values (these values might vary from bus maps to bus maps) where the greedy algorithm can guarantee a very high efficiency (See section 4.1.4 and Nguyen's method to calculate the lower bound for the efficiency [18]). Otherwise, when the greedy algorithm cannot guarantee a solution with high efficiency, GA with the greedy algorithm's result in the initial population should be considered for the SOBP, or when the bus map instances' critical squares density is relatively low, the GA with the best bus route individual should be considered for the problem.

5 Threats to validity

In this section the possible threats to the validity of this paper are presented and discussed. The possible threats are as follows:

Construct validity the SOBP was to some extend abstracted in our experiment. Real conditions might greatly differ from what was expected in the problem. For example, the air quality within an area (square) might be different from point to point, assuming that the sensors can detect the whole area when its radius just merely touched the edge of the critical square might not be accurate to the real-world. This could have a significant impact on the optimization effort. Hence, applying the proposed approaches to real-world conditions may require additional thoughts and considerations.

Conclusion validity Since the GA are probabilistic algorithms and to a certain extend include some degrees of fortuitousness, results may vary depending on the specific execution. With the aim to counter this problem, a good amount of repetitions over various instances of (m, k) was instituted on each test instance. Despite this, it can still not change the stochastic nature of the proposed methods, further executions might differ from what we have achieved and observed.

Internal validity The performance of the GA depend greatly on the parameters and operators chosen for the experiments. Finding a good set of parameters and operators alone involves a lot of resources and effort. However, most of our operators and parameters were chosen according to the recommendations of relevant literature on the subject.

External validity Due to constraints on the time and the computational resources, we could only perform most of the experiment on instances with relatively low critical square density ($\frac{c}{p \times q}$). A higher amount of critical squares may greatly hinder the performance of the algorithms, in both the quality of the solutions and the time needed to compute these solutions, as shown and discussed in a few test instances.

6 Conclusion and future work

In this paper, we investigated some heuristic approaches for the SOBP optimization problem. These approaches were tested on real-world data such as the bus maps of Hanoi and Ho Chi Minh City, Vietnam. Results show that with the cost of high running time, the GA can statistically find better solutions than its greedy approach counterpart given a good initial population was selected. However, it should be noted that in the cases where (m, k) surpasses a particular threshold, the greedy algorithm can almost always guarantee solutions with very high efficiency. Such that to save time and resources, in these cases, only the greedy algorithm should be used to find effective solutions.

Regarding future work, it is crucial to take into consideration the different sets of values and operators chosen for the heuristic algorithms. As described in the paper, using a fixed set of parameters might be hampering the algorithms. Additionally, given sufficient resources, different problem instances (e.g. different bus maps of other cities, higher critical squares density) should be tested and added to the result. Also, for the justification of the results, attributes and properties of the bus maps should be further analyzed and taken into account. Lastly, the GA should be run with regards to some alternative termination condition, for instance, the optimal solution lower bound suggested by Nguyen et al. [18].

Bibliography

[1] https://pamair.org/, September 2021.

[2] D Arivudainambi, S Balaji, and TS Poorani. Sensor deployment for target coverage in underwater wireless sensor network. In *2017 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6. IEEE, 2017.

[3] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[4] Ritamshirsa Choudhuri and Rajib K Das. Coverage of targets in mobile sensor networks with restricted mobility. *IEEE Access*, 6:10803–10813, 2018.

[5] Charles Darwin. On the origin of species, 1859, 2016.

[6] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[7] Stephanie Forrest. Genetic algorithms. *ACM Computing Surveys (CSUR)*, 28(1):77–80, 1996.

[8] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, January 1985.

[9] Tzung-Pei Hong and Hong-Shung Wang. A dynamic mutation genetic algorithm. In *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, volume 3, pages 2000–2005. IEEE, 1996.

[10] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

[11] Sami Kaivonen and Edith C.-H. Ngai. Real-time air pollution monitoring with sensors on city bus. *Digital Communications and Networks*, 6(1):23 – 30, 2020.

[12] Yee Leung, Yong Gao, and Zong-Ben Xu. Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Transactions on Neural Networks*, 8(5):1165–1176, 1997.

[13] Qifeng Lin, Wei Liu, Hongxin Peng, and Yuxing Chen. Efficient genetic algorithm for high-dimensional function optimization. In *2013 Ninth International Conference on Computational Intelligence and Security*, pages 255–259. IEEE, 2013.

[14] K Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011.

[15] Robin Mueller-Bady, Martin Kappes, Inmaculada Medina-Bulo, and Francisco Palomo-Lozano. Maintaining genetic diversity in multimodal evolutionary algorithms using population injection. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 95–96, 2016.

[16] Heinz Mühlenbein, M Schomisch, and Joachim Born. The parallel genetic algorithm as function optimizer. *Parallel computing*, 17(6-7):619–632, 1991.

[17] Tu N Nguyen, Bing-Hong Liu, and Shih-Yuan Wang. On new approaches of maximum weighted target coverage and sensor connectivity: Hardness and approximation. *IEEE Transactions on Network Science and Engineering*, 7(3):1736–1751, 2019.

[18] Viet-Dung Nguyen, Phi Le Nguyen, Trung Hieu Nguyen, and Phan Thuan Do. A $\frac{1}{2}$ -approximation algorithm for target coverage problem in mobile air quality monitoring systems. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[19] Santhosh Pandey, Shaoqiang Dong, Prathima Agrawal, and Krishna M Sivalingam. On performance of node placement approaches for hierarchical heterogeneous sensor networks. *Mobile Networks and Applications*, 14(4):401–414, 2009.

[20] Kumara Sastry, David Goldberg, and Graham Kendall. Genetic algorithms. In *Search methodologies*, pages 97–125. Springer, 2005.

[21] Fatih Senel and Mohamed Younis. Relay node placement in structurally damaged wireless sensor networks via triangular steiner tree approximation. *Computer Communications*, 34(16):1932–1941, 2011.

[22] William M Spears et al. Simple subpopulation schemes. In *Proceedings of the Evolutionary Programming Conference*, volume 3, pages 296–307. World Scientific, 1994.

[23] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[24] RF Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.