Solving the Classical Unequal Area Static Facility Layout Problem Using A Modified Grey Wolf Optimization Algorithm

A Special Problem by

Sean Francis N. Ballais
2015-04562
BS Computer Science

Presented to the Faculty of the
Division of Natural Sciences and Mathematics

In Partial Fulfillment of the Requirements
For the Degree of
Bachelor of Science in Computer Science

University of the Philippines Visayas
TACLOBAN COLLEGE
Tacloban City

Month Year

This special problem, entitled "**SOLVING THE CLASSICAL UNEQUAL AREA STATIC FACILITY LAYOUT PROBLEM USING A MODIFIED GREY WOLF OPTIMIZATION ALGORITHM**", prepared and submitted by **SEAN FRANCIS N. BALLAIS**, in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** is hereby accepted.

PROF. VICTOR M. ROMERO II
Special Problem Adviser

Accepted as partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**.

DR. EULITO V. CASAS JR.
Chair, DNSM

Acknowledgements

First of all I would like to thank the Lord for his guidance during the course of my research and for making this thesis possible. I would like to thank my family who served as my inspiration, and never failed to support me all throughout my studies. Thanks to ...

Abstract

Although the abstract is the first thing that appears in the thesis, it is best written last after you have written your conclusion. It should contain spell out your thesis problem and describe your solution clearly.

Make sure your abstract fits in one page !

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Positioning assets, such as facilities and equipment, within a pre-defined region, such as a plot of land or a building, in a fashion that is tailoured towards a criteria of optimality for a specific problem is one endeavour that has multiple applications in different fields, primarily due to the benefits it provides. Finding the best possible asset positioning can result in improved operations efficiency, better productivity [10], and even decreases in expenses [34]. As matter of fact, due to the benefits of asset positioning, $300 billion dollars have been spent each year on just determining sub-optimal locations of buildings and facilities in the United States alone [3]. This is further proof of the importance of asset positioning. One entertaining example of said application is showcased by Barriga et al. (2014). In their paper, the authors developed a genetic algorithm that optimized placement of buildings in a StarCraft match. The algorithm produced building placements that allowed the defending player's base to better survive base assaults from the opposing player [5]. Developing an open-plan office layout is another application of asset positioning. Chen et al. (2020) also developed a genetic algorithm that generates an open-office layout where the space utilization is maximized as possible [6]. This task of arranging assets within a given

space according to some criteria has a formal term, which is the "facility layout problem", often abbreviated as "FLP". We will be discussing facility layout problems in more detail in this chapter.

FLP is a field that has been researched as early as 1957 (with Koopsman T.C., and Beckman, M. being the first to model the problem) [20], and there is still active research around it to this day. This research paper is one of the testaments to that. In this research, we will be solving the classical facility layout problem using a recent optimization algorithm called the Grey Wolf Optimization (GWO) algorithm. The specific type of FLP that we will solve is called the unequal area static FLP. The categorization will be discussed later in this paper. The proposed algorithm will then be compared to a genetic algorithm using experimental data used in other related papers.

1.1 Facility Layout Problem

The problem of arranging a set of facilities and/or machines in a pre-determined area, or a set of possible locations (such as in the work of Farmakis, P., and Chassiakos, A. [11]) is called the facility layout problem (FLP). The facilities and/or machines are arranged in such a way that the resulting layout is in line with some criteria or objectives and under certain constraints. These constraints, which must not be violated, include shape, size, orientation, pick-up/drop-off points [18], and usable area [15]. Facilities and/or machines must also not overlap. Solutions that satisfy the aforementioned conditions are called feasible solutions [23].

Generally, the facility layout problem is considered to be an **NP-Hard** problem [8]. Hosseini-Nasab, H., Fereidouni, S., and Fatemi, S. have noted in their systematic

review of FLP that most researches dealing with the facility layout problem model their problems either as a quadratic assignment problem (QAP) or a mixed integer programming problem [18]. According to Drira, A., Pierreval, H., and Hajri-Gabouj, S., the former is sometimes used in discrete FLP formulations, while the latter is often used in continuous formulations [8]. Discrete and continuous FLP formulations will be discussed later. **Quadratic assignment problems** deal with placing n facilities in n locations in such a way that minimizes the assignment cost. The assignment cost is the sum of all facility pairs's flow rate between each other multiplied by their flow rate [1]. This assignment cost is commonly seen in many FLP researches, as we will discuss later. QAP is also known to be an NP-Hard problem [14]. It should be noted though that *some* instances of QAP are easy to solve [12]. The other modeling framework, **mixed integer programming**, can solve problems with both discrete decisions and continuous variables. An example of such problem is the assignment problem [30], which the FLP can be classified under. In this formulation, a set of integer and real-valued integers are being optimized based on an objective function that is being minimized or maximized, while satisfying constraints which are linear equations or inequalities [33]. Mixed integer programming, when in the context of optimization, is also known to be NP-Hard [30]. These two formulations being known to be generally NP-Hard proves that FLP is indeed generally NP-Hard.

The fact that FLP is an NP-Hard problem has resulted in many research works that utilize heuristics (such as simulated annealing and genetic algorithms). Note that there are also works that utilize exact methods, which seek to find the *optimal* solution for a problem. However, the NP-Hard nature of the FLP prevents them from finding the solution in large problems within reasonable time [4].

The Basic Mathematical Model

Each problem instances of the facility layout problem naturally will have their own mathematical models tailor-fit for their problem instance. Nevertheless, based on our observations and from readings, most of those models are derivatives of or use (such as in [13], [21], and [26]) what will be calling a basic minimization function, which is defined as:

$$\min F = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} f_{ij} d_{ij}$$

where N is the number of facilities, c_{ij} is the cost of handling materials between locations i and j, f_{ij} is the flow rate between i and j, and d_{ij} is the distance between the centroids of i and j. The distance function may differ from work to work. For example, Liu, J., et. al. uses the Manhattan distance in their work [22], while in the work of Ripon, K. S. N., et. al., Euclidean distance was used [31]. In works that derive from this formula, such as in [11], [32], and [27], it was observed that d_{ij}, or a similar variable or expression, is commonly present in the work's objective function while c_{ij} and f_{ij} *may* be present and/or the work uses more or fewer variables.

Drira, A., Pierreval, H., and Hajri-Gabouj, S. note the same observation but showcase a slightly differing formula in their 2007 survey of facility layout problems. Unlike the basic minimization formula above, their formula has f_{ij} and c_{ij} combined. They also note that the function above is typically used in continuous formulations of the FLP. The discrete formulation uses a similar function, but ensures that a facility is only in one location, a location only contains one facility, and makes sure that only pairs of locations that contain facilities contribute to the fitness value of a solution. (The descriptions and the differences of the discrete and continuous formulations are

discussed in the next section.) Additionally, they mention that the function is also subject to the following constraints: (1) facilities must obviously not overlap with one another, and (2) the total area used by the facilities must be equal to or less than the allotted area [8]. These constraints have been observed to be generally in many FLP works.

Discrete vs Continuous Formulations

Solving instances of the facility layout problem requires determining the form of the solution. The form is highly dependent on the problem being solved. Some problems may require a solution that assigns assets to pre-existing locations, while others may require more flexibility. Facility layout problems may be categorized based on the characteristics of these solutions, or formally known as formulations: discrete, and continuous.

In a discrete formulation, the region where the facilities will be laid out are divided into equal rectangular blocks of the same shape and size, or have pre-determined possible facility locations [8]. Each facility will be given a number of blocks, or be assigned to one facility location, respectively. This formulation, however, does not suit well when the facilities require exact positions and it cannot model facility attributes such as orientation. In problems that have such requirements, a continuous formulation is more appropriate [18]. Facilities in a continuous formulation are usually located by either their centroid coordinates, half length, and half width, or by their bottom-left coordinates, length, and width [8]. This allows for the formulation's flexibility compared to its discrete counterpart. However, this does provide challenges towards ensuring that no two facilities overlap with one another. Discrete formulations do not need to consider this problem due to their inherent characteristics.

Static vs Dynamic Facility Layout Problems

Another categorization for facility layout problems is based on whether the layouts will change over time. There are situations where a regular change of layout over some periods of time is necessitated. The layout of facilities in a construction is one example. As the construction of a building moves to from phase to another, the layout of facilities within the construction site change to better fit the needs of the current phase of construction [11]. A similar need is the motivation behind changing layouts in manufactories. Product demand variations, and even a change in product design can incline a factory's management to reorganize facilities in the building to be more efficient in response to the changes [29]. There are two categories for the aformentioned criteria. These are: (1) static, and (2) dynamic. We will refer to these categories as **"period-based layout categories"** in this paper.

The survey of Hoisseini-Nasab et al. (2018) showed that the most common period-based categorization in literature is the static facility layout problem [18]. This is likely due to the fact that static facility layout problems are easier to solve than dynamic facility layout problems. Though, it is also possible that many problems just happen to not require consideration of variable changes over time. The **static facility layout problem**, abbreviated as SFLP, is a type of FLP where variables to be considered such as material handling costs do not change for a considerable amount of time [28]. For this type of problems, only a single layout is generated since no changes are made in the considered variables over time.

However, some industries will find SFLPs inadequate for their needs. There are companies that require adaptability to changes to, for example, product demands. For cases like this, the other category, dynamic facility layout problems are more

appropriate [7]. In the dynamic facility layout problem, abbreviated to DFLP, the variables to be considered change over time, unlike in SFLP. The cost of rearranging facilities are also considered in the problem [17]. The solutions for DFLPs are also divided into time periods, where each period has a different layout. This period may equate to years, seasons, months, or weeks [7]. DFLPs can also be viewed as extensions of SFLP, since each layout in a period can be viewed as a solution to an SFLP with that period's variables into consideration but with rearrangement costs considered. While most research today is focused on SFLPs, Hosseini-Nasab et al. (2018) recommends that research should deal with DFLPs more these days due to rapid scientific developments, and product changes [18].

Other FLP Classifications

Facility layout problems can also be categorized based on different characteristics. FLPs can be divided by the area of their facilities. The facilities may have the same areas, referred to as equal areas, or have different areas, referred this time to as unequal areas [7]. They can also be divided based on the possible arrangements of facilities. Some problems may have facilities located only in a single pre-defined row (single-row), or they may be placed anywhere in the region (open field) [8]. There are multiple classifications for FLP and discussing them in this chapter would take long and dislocate the focus of this paper. Due to that, we would like to refer the reader to the papers of Drira et al. (2007) [**?**] and Hosseini-Nasab et al. (2018) [18] for more information on FLP classifications.

1.2 The Grey Wolf Optimization Algorithm

In this paper, we will be using the Grey Wolf Optimization algorithm to solve the unequal-area static facility layout problem. As such, we will be introducing the algorithm here for us to gain a better understanding of the algorithm.

The Grey Wolf Optimization algorithm, abbreviated as GWO, was first conceived by Mirjalili et al. (2014) in 2014. The optimization algorithm is inspired from the hunting and social behaviour of grey wolves. There is a hierarchy in packs of wolves. Each category in the hierarchy have specific responsibilities. There are four categories: alpha (α), beta (β), delta (δ), and omega (ω). Alpha wolves are responsible for making major decisions for the pack. Every wolf must follow the alpha. However, sometimes the alpha follows other wolves. The second in line is the beta, which ensures the discipline of the pack and advises the alpha. They also command other wolves and reinforces the alpha's commands. The lowest in the hierarchy are the omegas. They must follow the orders of the other wolves, and are the last to eat. Despite their low status, they are still crucial in the pack as their absence causes the pack to face internal fighting and problems. If a wolf is not an alpha, beta, nor omega, they are considered to a delta, the third category in the hierarchy. Deltas may act as scouts, sentinels, elders, hunters, or caretakers. They are also at a category higher than the omegas [25] [16]. As an interesting side note, the inspiration for Grey Wolf Optimization initially came from The Grey, a movie where survivors of a plane crash must survive, but a pack of grey wolves surround them [2].

Mathematical Model

The mathematical model assumes the existence of a "pack of wolves". The number of wolves in this pack can be determined by the researcher. Each wolf of this a solution to the problem. We will be delving into the model more in this section, discussing about the model of leadership hierarchy, encircling, and hunting behaviour of grey wolves. The prey being hunted in this scenario is the best solution for a given problem [25].

Leadership Hierarchy

Solutions are assigned to a certain hierarchy in the mathematical model of GWO. The fittest solution is considered the alpha (α), while the second and third fittest are considered to be the beta (β) and delta (δ) solutions. The rest of the solutions are referred to as the omega solutions. The leading wolves guide the omegas towards the prey throughout the search process [16].

Encircling the Prey

Prey encirclement, which is one of the first steps when grey wolves hunt for their prey, can be modeled with the following:

$$\vec{X}(t+1) = \vec{X_p}(t) - \vec{A} \cdot \vec{D}$$
$$\vec{D} = \left| \vec{C} \cdot \vec{X_p}(t) - \vec{X}(t) \right|$$
$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r_1} - \vec{a}$$
$$\vec{C} = 2 \cdot \vec{r_2}$$

where $\vec{X}(t)$ and $\vec{X}(t+1)$ are the positions of the wolf at the iteration t and

$t+1$ respectively, $\vec{X_p}(t)$ represents the location of the prey at the iteration t, $\vec{D}$ is the difference vector, $\vec{A}$ and $\vec{C}$ are coefficient vectors, and $\vec{r_1}$ and $\vec{r_2}$ are uniformly random vectors with the range $[0, 1]$. $\vec{a}$ is vector that linearly decreases from 2 to 0 over the course of iterations [**?**]. The original paper on GWO does not specify but Gupta, S. and Deep, K. provided the following equation to specify the decrease of $\vec{a}$ from 2 to 0 [16]:

$$\vec{a} = 2 - 2 \cdot \left(\frac{t}{\text{maximum number of iterations}} \right)$$

Note that the multiplication of vectors in the equations above is a component-wise multiplication, and not a dot product [24].

Hunting

We typically do not know the position of the prey in an abstract search space. As such, it is presumed that the α, β, and δ solutions have the best idea so far of the position of the prey [**?**]. Each wolf update their positions based on the following equations.

$$\vec{X_1'} = \vec{X_\alpha}(t) - \vec{A_\alpha} \cdot \vec{D_\alpha} \tag{1.2.1}$$

$$\vec{X_2'} = \vec{X_\beta}(t) - \vec{A_\beta} \cdot \vec{D_\beta} \tag{1.2.2}$$

$$\vec{X_3'} = \vec{X_\delta}(t) - \vec{A_\delta} \cdot \vec{D_\delta} \tag{1.2.3}$$

$$\vec{X}(t+1) = \frac{\vec{X_1'} + \vec{X_2'} + \vec{X_3'}}{3} \tag{1.2.4}$$

where $\vec{X_\alpha}$, $\vec{X_\beta}$, and $\vec{X_\delta}$ represent the α, β, and δ solutions [16].

Exploration and Exploitation

The exploration phase of metaheuristics is modeled by the search phase, while exploitation is modeled by the attack phase. When $\left|\vec{A}\right| < 1$, or $\vec{C} < 1$, GWO is undergoing exploitation of the search space. Exploitation can be viewed as the wolves approaching towards the prey. On the other hand, when $\left|\vec{A}\right| > 1$, or $\vec{C} > 1$, the algorithm is in the search phase, where the wolves can be viewed as searching for the prey. In the search process, as the number of iterations t reach the maximun possible number, the algorithm tends to focus more on exploitation than exploration. $\vec{A}$ and $\vec{a}$ eventually approach 0, leaving $\vec{C}$ the sole vector to eventually influence the search exploration. At this point, the algorithm will intensify towards exploitation.

Chapter 2

Review of Related Literature

The polygonal bounding area building placement problem (PBABPP) deals with the arrangement of buildings within a polygonally-shaped area. As far as the authors know, no previous research has been conducted for the problem. More so with the fact that this research also takes into account areas prone to natural hazards, namely flooding and landslides. Nevertheless, PBABPP is still an extension of the facility layout problems (FLP), which delves with determining the placement of various assets in a facility. Many techniques utilized in solving FLP instances can be adapted in PBABPP. As such, this literature review will mostly consist of prior works that attempt to solve facility layout problems. Majority of the FLP works included here utilize approximation methods since this work uses one. FLP researches are also easy to find thanks to the fact that it is NP-Hard. Being NP-Hard resulted in numerous researches being done for the field [8].

All research works in the literature produce layouts with varying degrees of fitness and performance. The process of layout generation differ from proposal to proposal based on the specific FLP instance they are working on. Various techniques are used to solve the FLP. Exact methods have been used, but stochastic-based methods like

local search and population-based evolutionary algorithms are popular.

The instance of the facility layout problem that is most related to this work is the unequal area static facility layout problem (UA-SFLP). The works dealing with the UA-SFLP (and even the unequal-area dynamic facility layout problem) use a rectangle to mark the bounds of the area where assets can be placed. This is unlike the problem we are solving here where a polygonal area is used instead.

Chapter 3

Statement of the Problem

Raster images are composed of a pixel matrix. This makes their data representation simple. However, this reduces the amount of detail and quality they have. This limitation is clearer when scaling raster images. This motivates the use of an alternative form of representing images. One form is vector images, which uses mathematical equations to represent an image. The process of converting a raster image to a vector image is called *vectorization*. Semi-structured imagery, such as those used in graphic designs, is one of the classes of images that would benefit from vectorization. This process will allow such images to be easily scaled without sacrificing quality nor detail.

There have been numerous works that tackles image vectorization for semi-structured images. Many of these works primarily use a curve optimization algorithm such as variants of NEWUOA and conjugate gradient. A machine learning approach has been used in one of the works, but as a preprocessing step only [**?**]. Deep learning have been used to solve problems in multiple domains such as natural language processing (NLP), object detection, and playing board games. No other known work has applied deep learning as a core step in image vectorization. This study will deal with such application.

Chapter 4

Objectives

This study is primarily aims to apply deep learning via artificial neural networks to the problem of vectorizing semi-structured imagery. However, there are still some key objectives that this study seeks to accomplish:

1. To develop an approach that considers Gestalt psychology.
2. To evaluate the effectiveness of using deep learning for image vectorization.
3. To evaluate the accuracy of results obtained from the proposed approach to the target raster inputs.
4. To evaluate and compare the results of the proposed approach to that of previous semi-structured image vectorization methods.
5. To evaluate and compare the speed of the proposed approach compared to previous semi-structured image vectorization methods.

Chapter 5

Methodology

The methodology used in this research is based on the frameworks proposed by Asl et al. (2015) [4], Asl, A. and Wong, K. (2015) [3], and Jiang, T. and Zhang, C. (2018) [19]. It is a mixture of the works of the aforementioned authors In this chapter, we will first discuss about the mathematical model of the problem being solved. Later, we will be delving into the inner workings of the solution representation, the algorithm, and then the technologies that were used in implementing the approach.

5.1 Mathematical Model

The goal of any metaheuristic, like what is being proposed in this paper, is to optimize a certain objective function. As mentioned in the first chapter, in facility layout problems, we minimize the following function:

$$\min F = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} f_{ij} d_{ij}$$

For the problem we are solving in this paper, we are optimizing the following equation that is not only a slight modifications of the basic mathematical model for FLPs, but also adds penalties to solutions that are infeasible, no matter the degree

of infeasiblity.

$$\begin{aligned}\min F = &\sum_{i=1}^{|B|}\sum_{j=i+1}^{|B|} c_{ij}d_{ij} \\ &+ \sum_{i=1}^{|B|}\sum_{j=i+1}^{|B|}\left(P_B\frac{A_0(i,j)}{\min(w_ih_i, w_jh_j)} + P_B\right)\cdot\alpha_0(i,j) \\ &+ \sum_{i=1}^{|B|}\left(P_R\frac{w_ih_i - A_1(i)}{w_ih_i} + P_R\right)\cdot\alpha_1(i)\end{aligned}$$

where:

x_i	top-left x coordinate of building i
y_i	top-left y coordinate of building i
w_i	width of building i
h_i	height of building i
R_x	top-left x coordinate of the bounding region
R_y	top-left y coordinate of the bounding region
R_w	width of the bounding region
R_h	height of the bounding region
c_{ij}	flow rate from building i to building j
d_{ij}	distance from the center of building i to the center of building j
P_B	penalty value for building intersection
P_T	penalty value for any building going out of bounds, even with a portion of a building

We elected to remove the flow rate from the basic formulation of the model that was discussed earlier in Equation 5.1.1.

$$\sum_{i=1}^{|B|}\sum_{j=i+1}^{|B|} c_{ij}d_{ij} \tag{5.1.1}$$

This is because we can consider flow rate as simply part of cost. In the original formulation, we were considering it from a material handling cost perspective, which

requires having both a cost and flow rate variable. However, in a general problem, we can consider cost to also include the frequency of movement from one facility to another, which is essentially the flow rate. As such, we can merge cost and flow rate into one variable.

The mathematical model allows for infeasible solutions to allow for better solutions in the long run. To follow this specification, the model includes expressions that penalizes solutions that meet any of the following conditions: (1) at least one building is intersecting with another building, and (2) a building, either in whole or in part, is outside the bounding area.

$$\sum_{i=1}^{|B|} \sum_{j=i+1}^{|B|} \left(P_B \frac{A_0(i,j)}{\min(w_i h_i, w_j h_j)} + P_B \right) \cdot \alpha_0(i,j) \tag{5.1.2}$$

Equation 5.1.2 is the expression that applies a penalty to solutions that meet the first condition. Notice that it has the functions $A_0(i,j)$ and $\alpha_0(i,j)$. They are defined by the following:

$$A_0(i,j) = I_L(x_i, x_j, w_i, w_j) \cdot I_L(y_i, y_j, h_i, h_j) \tag{5.1.3}$$

$$I_L(x_1, x_2, l_1, l_2) = \max(0, \min(x_1 + l_1, x_2 + l_2) - \max(x_1, x_2)) \tag{5.1.4}$$

$$\alpha_0(i,j) = \begin{cases} 1 & \text{if } A_0(i,j) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1.5}$$

$A_0(i,j)$ simply gets the area of intersection of buildings i and j. This is achieved by the use of $I_L(x_1, x_2, l_1, l_2)$, which computes the length or width of an intersection of buildings.

In the equation, for every pair of buildings that intersect, we apply a penalty that is the percentage of the area of the smallest building by area that is intersecting with

the other building multiplied by the penalty value for building intersection. This will allow for rewarding the algorithm for moving the buildings towards non-intersection. The same penalty value is also added to ensure that the algorithm prioritizes removing intersections over reducing the distance between the centers of the buildings. $\alpha_0(i, j)$ ensures that the penalty is only applied to pairs of buildings that intersect with one another.

$$\sum_{i=1}^{|B|} \left(P_R \frac{w_i h_i - A_1(i)}{w_i h_i} + P_R \right) \cdot \alpha_1(i) \tag{5.1.6}$$

The other part of the mathematical model, Equation 5.1.6, works in a similar principle as Equation 5.1.2. This equation applies a penalty value when the second condition of infeasibility is met. Like in 5.1.2, it has specific functions to help compute the penalty. They are defined as:

$$A_1(i) = I_L(x_i, R_x, w_i, R_w) \cdot I_L(y_i, R_y, h_i, R_h) \tag{5.1.7}$$

$$\alpha_1(i) = \begin{cases} 0 & \text{if} \quad \begin{array}{ll} R_x \leq x_i & \leq R_x + R_w \\ R_x \leq x_i + w_i & \leq R_x + R_w \\ R_y \leq y_i & \leq R_y + R_h \\ R_y \leq y_i + h_i & \leq R_y + R_h \end{array} \\ 1 & \text{otherwise} \end{cases} \tag{5.1.8}$$

$A_1(i)$ simply computes the area of intersection of the building and the bounding area. Now, since this only computes the intersection, we must subtract the intersection with the area of the building to get the area of the building that is outside of the bounding area. This is expressed by the numerator of the fractional expression in Equation 5.1.6. Similar to Equation 5.1.2, the equation applies a penalty value that is the percentage of the area of the total building area that is outside the bounding

x_0	y_0	$\angle_0$	$\ddots$	x_{n-1}	y_{n-1}	$\angle_{n-1}$

Figure 5.1: Visualization of the solution representation.

region multiplied and then added by the penalty value. The addition is also to ensure that the algorithm gives more priority to removing out-of-bounds buildings. $\alpha_1(i)$ ensures that the penalty is only applied to buildings that are, in part or in whole, out of bounds.

5.2 Solution Representation

The solution is represented using a one-dimensional array of floating numbers. In the array, every group of three consecutive elements are considered to be the x and y positions, and angle, respectively, of one building. While the x and y positions are allowed to be of any value, the angle value is restricted to only 0° and 90°. A visualization of the solution representation is shown by Figure 5.1.

5.3 The Algorithm

The proposed algorithm contains multiple phases to solve the unequal area static facility layout problem. The basic framework of the algorithm is inspired from the works of Asl et al. (2015) [4] and Asl, A. and Wong, K. (2015) [4]. The Grey Wolf Optimization aspect of the algorithm is inspired from Jiang, T. and Zhang, C. (2018) [19]. Figure 5.2 shows a flowchart of the algorithm. We will be further discussing the algorithm in detail in this section.

Population Generation

In generating the initial population, the order in which facilities are placed in the bounding region is shuffled. Once it is shuffled, each building i is then given a random x position between the inclusive range $\left[R_x + \frac{w_i}{2}, (R_x + R_w) - \frac{w_i}{2}\right]$, a random y position between the inclusive range $\left[R_y + \frac{h_i}{2}, (R_y + R_h) - \frac{h_i}{2}\right]$, and angle that is either $0°$ or $90°$ (chosen in a uniformly random fashion). This will generate a solution where the buildings are spread out throughout the bounding region. There will be building intersections (though this is dependent on the number of buildings and size of the bounding region), but no building will be out-of-bounds. This process is repeated until we generate a specified number of solutions. This number is determined by the population size.

Swapping Method

The swapping method is used to find a possible configuration for a solution that is better than the current configuration. This method is applied to all solutions in the population, but only in the first 100 iterations. Pseudocode for the swapping method is provided in Algorithm 1.

Crossover, Mutation, and Elitism

Experimentations with directly applying the formulas of Grey Wolf Optimization in solving our instance of the facility layout problem have not generated satisfying results. However, results generated by an algorithm inspired from the paper of Jiang, T., and Zhang, C. (2018) [19] showed promise. As such the GWO portion of our proposed approach has been taken from their work. Their work takes certain elements from genetic algorithms, particularly the crossover and mutation operators, but kept

Algorithm 1 Pseudocode for the swapping method.

Let S be the collection of generated solutions.
Set S_{curr} be the current solution.
Add S_{curr} to S.
Set N_B be the maximum number of buildings.
for i = 0 until $N_B - 2$ **do**
    for j = i + 1 until $N_B - 1$ **do**
        Building i's orientation in S_{curr} is changed to the other orientation, ↪ and the resulting new solution is added to S
        Building j's orientation in S_{curr} is changed to the other orientation, ↪ and the resulting new solution is added to S
        Building i's and j's orientations in S_{curr} are changed to the other ↪ orientation, and the resulting new solution is added to S
        Building i's and j's positions are exchanged in S_{curr}, and the resulting new ↪ solutiA movement and a orientation changeon is added to S.
        Building i's and j's positions are exchanged in and the orientation of ↪ building i is changed in S_{curr}, and the resulting new solution ↪ is added to S.
        Building i's and j's positions are exchanged in and the orientation of ↪ building j is changed in S_{curr}, and the resulting new solution ↪ is added to S.
        Building i's and j's positions are exchanged in and the orientations of ↪ buildings i and j are changed in S_{curr}, and the resulting new solution ↪ is added to S.
    end for
end for
return the best solution in S.

the general principle of Grey Wolf optimization intact. We will be discussing each operator in detail in this section. Elitism is also implemented in our proposed approach to ensure that the best solutions found so far do not get lost throughout iterations. It will also be discussed in this section.

Crossover

Grey wolf optimization algorithms breed with the other solutions with the top three solutions in the population. That is modelled by the mathematical definition of the optimization algorithm. However, directly the definition does not yield good results. Due to that, we are using a crossover operator from the work of Jiang, T., and Zhang, C. (2018) [19].

The crossover operator used here can be mathematical defined as:

$$X(t+1) = \begin{cases} f(X(t), X_{alpha}(t)) & \text{if rnd} \leq \frac{1}{3} \\ f(X(t), X_{beta}(t)) & \text{if } \frac{1}{3} < \text{rnd} \leq \frac{2}{3} \\ f(X(t), X_{delta}(t)) & \text{if rnd} \geq \frac{2}{3} \end{cases}$$

where f is the crossover function, and rnd $\sim U(0, 1)$. This means that a solution will be crossed over by one of the best three solutions, selected randomly. In our approach, we have used the uniform crossover. Note that this crossover is applied to every solution in a population in accordance with the principle of GWO. In our approach, only one offspring is generated from the crossover.

Mutation

Over time, as more and more iterations are performed, the diversity of the population will eventually be lost. This is due to the fact that the wolves are only updated based on the best three solutions. This will also result in premature convergence [19]. To

combat this, a mutation operator is performed to reintroduce diversity. The mutation operator used is an operator that we dub the "Buddy-Buddy Mutation".

The **Buddy-Buddy Mutation** is a mutation operator that simply selects two pairs of buildings D and S, and move one of them to the side of the other building. Building D is referred to as the dynamic buddy, while building S is referred to as the static buddy. Building D will be the building that will be moved towards the other building, which is building S in our case. When moving building D, a side E of building S will first be randomly chosen. Afterwards, an orientation for building D will be randomly chosen, whether it will be parallel or perpendicular to E. Once a side and orientation has been selected, building D will be moved adjacent towards building S at side E with the chosen orientation. The implementation of this mutation operator in our proposed approach gives buildings that intersect with another building more chances of being selected as the dynamic buddy. Pseudocode and a visualization of the operator is provided by Algorithm 2 and Figure **??**, respectively.

Algorithm 2 Pseudocode for the Buddy-Buddy Mutation.

1: Randomly select two buildings D and S, with more weight given to buildings that are intersecting with another.
2: Set E to be a randomly selected side of building S.
3: Set O to either be a parallel or perpendicular orientation, randomly selected.
4: Move building D adjacent to side E of building S with the orientation O.

The rate at which a solution is mutated is highly dependent on the fitness of the solution. The worse the fitness of a solution is, the more likely it is to be mutated. This encourages the proposed algorithm to improve solutions that are generally bad. This rate scheme makes this an adaptive mutation operator [19]. The mutation rate is mathematically modelled as:

$$m(X,t) = 1 - \frac{fit_{max}(t) - fit(X(t))}{fit_{max}(t) - fit_{min}(t)} \tag{5.3.1}$$

where m_k refers to the mutation rate of a solution X at iteration t, fit is a function that gets the fitness of a solution, and fit_{min} and fit_{max} gets the minimum and maximum fitnesses of the population, respectively.

Elitism

One variant of genetic algorithms includes elitism. This elitism allows a genetic algorithm to keep a number of best solutions in the next generation, ensuring that the best solutions do not get discarded over time. Note that this elitism strategy is not only limited to genetic algorithms. Other evolutionary algorithms may also utilize this strategy [9]. We are also taking this principle into our proposed approach. In our proposed approach, we are keeping the best three solutions in the previous iteration to the next iteration. We chose three due to the fact that the best three solutions in a population have significance in GWO.

Local Searches

Remember that our implementation is based on aforementioned previous works that used local search algorithms in conjunction to genetic algorithms. They combined GAs with local search algorithms because GAs find it hard to explore within the convergence area. Hybridizing it with a local search algorithm improves performance [31]. In our proposed approach, we are keeping this aspect of the previous works. This will also ensure that we are able to search within the convergence area more intensely and find better solutions. In the previous works and in ours, there are two local search algorithms, dubbed ”Local Search 1” and ”Local Search 2”. They vary

in terms of searching intensity, but both attempts to obtain better solutions. We will be discussing details of both in this section.

Local Search 1

The first local search algorithm, "Local Search 1", performs a local search by creating a number of solutions by moving each building in different directions by a certain random amount and changing its orientations after movement and obtaining the best solution from these activities. In our approach, the certain amount of movement is a random number between 1 and 5. This search algorithm is only applied to the best solution of the current iteration, and the best solution found in this search becomes the new best solution and replaces the previously best solution. The movements of each building is defined by a set of "activities". This set of activities is shown by Table 5.1. Additionally, pseudocode of the search algorithm is shown in Algorithm 3.

Algorithm 3 Pseudocode for Local Search 1.

1: Set S to be a collection of solutions.
2: Set S_{curr} to be the solution being optimized.
3: Add S_{curr} to S.
4: Set N_B be the maximum number of buildings.
5: Set N_A be the maximum number of activities.
6: **for** i = 0 until $N_B - 1$ **do**
7: **for** a = 0 until $N_B - 1$ **do**
8: Perform activity a with building i in S_{curr} and save the new solution in S.
9: Perform activity a with building i, and change the orientation of the building to the other orientation in S_{curr} and save the new solution in S.
10: **end for**
11: **end for**
12: **return** the best solution in S.

Activity Number	Description
0	A building is moved to the right along the x-axis by a random number between 1 and 5.
1	A building is moved to the left along the x-axis by a random number between 1 and 5.
2	A building is moved to the upwards along the y-axis by a random number between 1 and 5.
3	A building is moved to the downwards along the y-axis by a random number between 1 and 5.
4	Generate two random numbers from 1 and 5 and a building is moved to the right and then upward, respectively, by those numbers.
5	Generate two random numbers from 1 and 5 and a building is moved to the right and then downward, respectively, by those numbers.
6	Generate two random numbers from 1 and 5 and a building is moved to the left and then upward, respectively, by those numbers.
7	Generate two random numbers from 1 and 5 and a building is moved to the left and then downward, respectively, by those numbers.

Table 5.1: Activities for moving a building in Local Search 1

Local Search 2

Local Search 2 is a more intense version of Local Search 1, in order to find the best solution so far. Unlike the latter that only moves one building at a time, Local Search 2 moves two buildings instead. The two buildings will also have their orientations changed after each activity. This local search is only applied to the best solution found in the last 50 iterations. The set of activities for this local search is shown by Table 5.2, and a pseudocode of the search algorithm is shown in Algorithm 4.

Activity Number	Description
0	Building i and $i+1$ are moved to the right along the x-axis by a random number between 1 and 5.

1	Building i and $i+1$ are moved to the left along the x-axis by a random number between 1 and 5.
2	Building i and i+1 are moved upwards along the x-axis by a random number between 1 and 5.
3	Building i and $i+1$ are moved downwards along the x-axis by a random number between 1 and 5.
4	Generate two random numbers from 1 and 5 and buildings i and i+1 are moved to the right and then upward, respectively, by those numbers.
5	Generate two random numbers from 1 and 5 and buildings i and $i+1$ are moved to the right and then downward, respectively, by those numbers.
6	Generate two random numbers from 1 and 5 and buildings i and $i+1$ are moved to the left and then upward, respectively, by those numbers.
7	Generate two random numbers from 1 and 5 and buildings i and $i+1$ are moved to the left and then downward, respectively, by those numbers.
8	Generate two random numbers a and b that are from 1 to 5, and building i is moved upward by a and building $i+1$ is moved to the right by b.
9	Generate two random numbers a and b that are from 1 to 5, and building i is moved upward by a and building $i+1$ is moved downward by b.
10	Generate two random numbers a and b that are from 1 to 5, and building i is moved upward by a and building $i+1$ is moved to the left by b.

11	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the right by a and building $i+1$ is moved downward by b.
12	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the right by a and building $i+1$ is moved upward by b.
13	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the right by a and building $i+1$ is moved to the left by b.
14	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the left by a and building $i+1$ is moved to downward by b.
15	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the left by a and building $i+1$ is moved to the right by b.
16	Generate two random numbers a and b that are from 1 to 5, and building i is moved to the left by a and building $i+1$ is moved upward by b.
17	Generate two random numbers a and b that are from 1 to 5, and building i is moved downward by a and building $i+1$ is moved to the right by b.
18	Generate two random numbers a and b that are from 1 to 5, and building i is moved downward by a and building $i+1$ is moved to the left by b.
19	Generate two random numbers a and b that are from 1 to 5, and building i is moved downward by a and building $i+1$ is moved upward by b.

Table 5.2: Activities for moving a building in Local Search 2

Algorithm 4 Pseudocode for Local Search 2.

1: Set S to be a collection of solutions.
2: Set S_{curr} to be the solution being optimized.
3: Add S_{curr} to S.
4: Set N_B be the maximum number of buildings.
5: Set N_A be the maximum number of activities.
6: **for** i = 0 until $N_B - 2$ **do**
7: **for** a = 0 until $N_B - 1$ **do**
8: Perform activity a with building i in S_{curr} and save the new solution in S.
9: Perform activity a with building i, and change the orientation of ↪ building i to the other orientation in S_{curr} and save the new ↪ solution in S.
10: Perform activity a with building i, and change the orientation of ↪ building $i + 1$ to the other orientation in S_{curr} and save the new ↪ solution in S.
11: Perform activity a with building i, and change the orientations of ↪ buildings i and $i+1$ to the other orientations in S_{curr} and save the new ↪ solution in S.
12: **end for**
13: **end for**
14: **return** the best solution in S.

5.4 Implementation Technologies

Our proposed approach was developed using C++17 compiled using the Clang 11 compiler in an elementaryOS Hera environment. Building was handled by CMake, and package management was handled by Conan. Our implementation is built on top of CoreX, which is a custom 2D game engine. Using a game engine allowed us to visualize the results and configure experiments in a graphical manner. The libraries EASTL, ImGUI, SDL 2, SDL 2 TTF. sdl-gpu, nlohmann JSON, and EnTT were used in developing the engine, with EnTT and ImGUI directly used by our implementation itself.

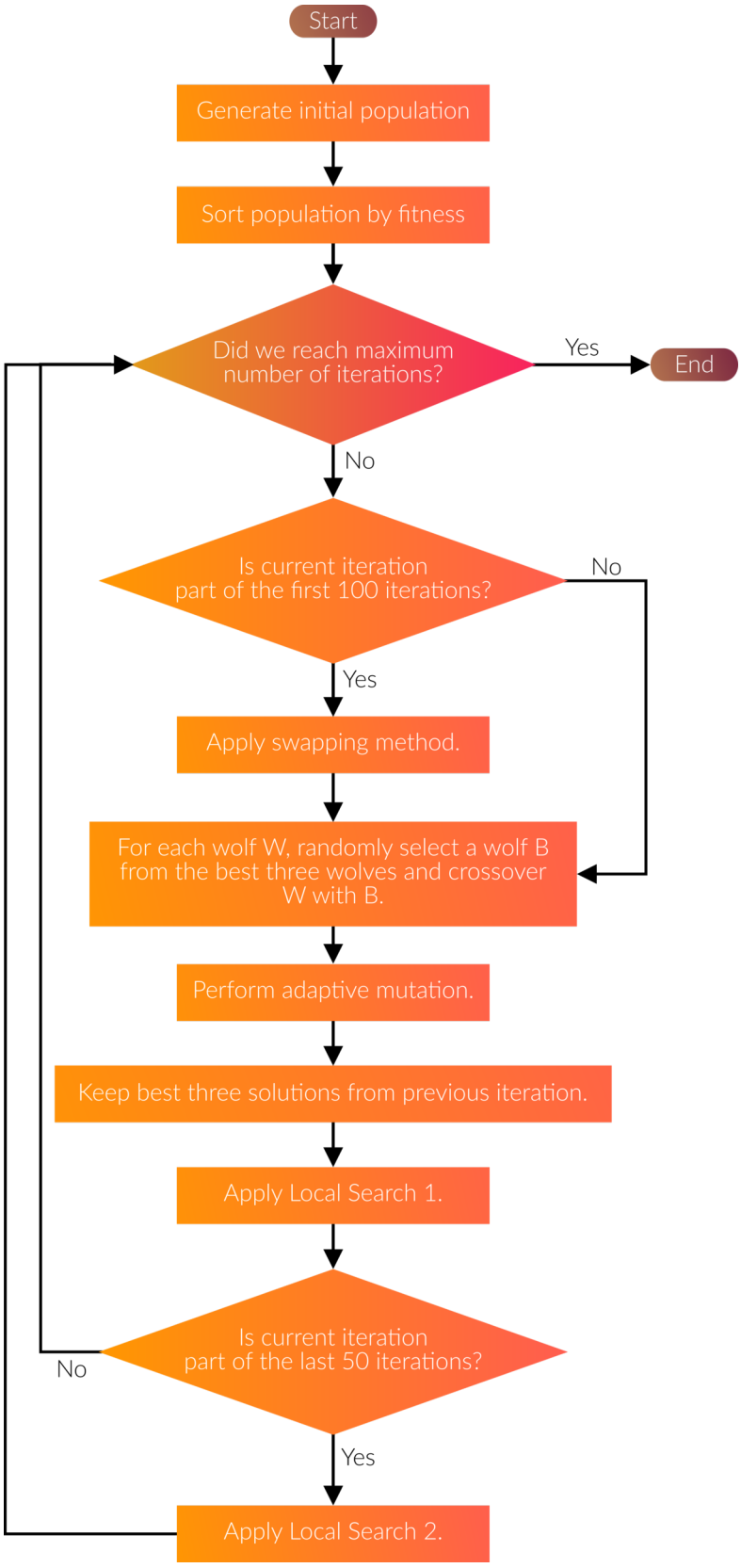

Figure 5.2: Flowchart detailing the algorithm.

Chapter 6

Describing How You Validated Your Approach.

Chapter 7

Stating Your Results and Drawing Insights From Them.

Chapter 8

Summarizing Your Thesis and Drawing Your Conclusions.

Appendix A

What should be in the Appendix

What goes in the appendices? Any material which impedes the smooth development of your presentation, but which is important to justify the results of a thesis. Generally it is material that is of too nitty-gritty a level of detail for inclusion in the main body of the thesis, but which should be available for perusal by the examiners to convince them sufficiently. Examples include program listings, immense tables of data, lengthy mathematical proofs or derivations, etc.

Bibliography

[1] Quadratic Assignment Problem — NEOS.

[2] The Grey (2011) - Plot Summary - IMDb.

[3] Ali Derakhshan Asl and Kuan Yew Wong. Solving unequal area static facility layout problems by using a modified genetic algorithm. *Proceedings of the 2015 10th IEEE Conference on Industrial Electronics and Applications, ICIEA 2015*, pages 302–305, 2015.

[4] Ali Derakhshan Asl, Kuan Yew Wong, and Manoj Kumar Tiwari. Unequal-area stochastic facility layout problems: solutions using improved covariance matrix adaptation evolution strategy, particle swarm optimisation, and genetic algorithm. *International Journal of Production Research*, (August 2015):0–25, 2015.

[5] Nicolas A. Barriga, Marius Stanescu, and Michael Buro. Building placement optimization in Real-Time Strategy games. *AAAI Workshop - Technical Report*, WS-14-15(October):2–7, 2014.

[6] Chen Chen, Ricardo Jose Chacón Vega, and Tiong Lee Kong. Using genetic algorithm to automate the generation of an open-plan office layout. *International Journal of Architectural Computing*, 2020.

[7] Ali DerakhshanăAsl and Kuan Yew Wong. Solving unequal-area static and dynamic facility layout problems using modified particle swarm optimization. *Journal of Intelligent Manufacturing*, 28(6):1317–1336, 2017.

[8] Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007.

[9] Haiming Du, Zaichao Wang, Wei Zhan, and Jinyi Guo. Elitism and distance strategy for selection of evolutionary algorithms. *IEEE Access*, 6:44531–44541, 2018.

[10] M. Adel El-Baz. A genetic algorithm for facility layout problems of different manufacturing environments. *Computers and Industrial Engineering*, 47(2-3):233–246, 2004.

[11] Panagiotis M. Farmakis and Athanasios P. Chassiakos. Genetic algorithm optimization for dynamic construction site layout planning. *Organization, Technology and Management in Construction: an International Journal*, 10(1):1655–1664, 2018.

[12] Mohammad Javad Feizollahi and Hadi Feyzollahi. Robust quadratic assignment problem with budgeted uncertain flows. *Operations Research Perspectives*, 2:114–123, 2015.

[13] Laura Garcia-Hernandez, Antonio Arauzo-Azofra, Lorenzo Salas-Morera, Henri Pierreval, and Emilio Corchado. Facility layout design using a multi-objective interactive genetic algorithm to support the DM. *Expert systems (Print)*, 32(1):94–107, 2013.

[14] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, 1979.

[15] José Fernando Gonçalves and Mauricio G. C. Resende. A biased random-key genetic algorithm for the unequal area facility layout problem. 246:86–107, 2015.

[16] Shubham Gupta and Kusum Deep. Cauchy grey wolf optimiser for continuous optimisation problems. *Journal of Experimental and Theoretical Artificial Intelligence*, 30(6):1051–1075, 2018.

[17] Seyed Shamsodin Hosseini and Mehdi Seifbarghy. A novel meta-heuristic algorithm for multi-objective dynamic facility layout problem. *RAIRO - Operations Research*, 50(4-5):869–890, 2016.

[18] Hasan Hosseini-Nasab, Sepideh Fereidouni, Seyyed Mohammad Taghi Fatemi Ghomi, and Mohammad Bagher Fakhrzad. Classification of facility layout problems: a review study. *International Journal of Advanced Manufacturing Technology*, 94(1-4):957–977, 2018.

[19] Tianhua Jiang and Chao Zhang. Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases. *IEEE Access*, 6:26231–26240, 2018.

[20] Andrew Kusiak and Sunderesh S. Heragu. The facility layout problem. *European Journal of Operational Research*, 29(3):229–251, 1987.

[21] Zhang Lin and Zhang Yingjie. Solving the Facility Layout Problem with Genetic Algorithm. *2019 IEEE 6th International Conference on Industrial Engineering and Applications, ICIEA 2019*, pages 164–168, 2019.

[22] Jingfa Liu, Huiyun Zhang, Kun He, and Shengyi Jiang. Multi-objective particle swarm optimization algorithm based on objective space division for the unequal-area facility layout problem. *Expert Systems with Applications*, 102:179–192, 2018.

[23] R. D. Meller and Y. A. Bozer. A new simulated annealing algorithm for the facility layout problem. *International Journal of Production Research*, 34(6):1675–1692, 1996.

[24] Seyeda Mirjalili. Mathematical models for the Grey Wolf Optimizer - YouTube, 2020.

[25] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey Wolf Optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[26] Maricar M. Navarro and Bryan B. Navarro. Evaluations of crossover and mutation probability of genetic algorithm in an optimal facility layout problem. *Proceedings of the International Conference on Industrial Engineering and Operations Management*, 8-10 March:3312–3317, 2016.

[27] Yunfang Peng, Tian Zeng, Lingzhi Fan, Yajuan Han, and Beixin Xia. An Improved Genetic Algorithm Based Robust Approach for Stochastic Dynamic Facility Layout Problem. *Discrete Dynamics in Nature and Society*, 2018, 2018.

[28] Pablo Pérez-Gosende, Josefa Mula, and Manuel Díaz-Madroñero. Overview of dynamic facility layout planning as a sustainability strategy. *Sustainability (Switzerland)*, 12(19):13–15, 2020.

[29] Mohammad Reza Pourhassan and Sadigh Raissi. An integrated simulation-based optimization technique for multi-objective dynamic facility layout problem. *Journal of Industrial Information Integration*, 8:49–58, 2017.

[30] Arthur Richards and Jonathan How. Mixed-integer programming for control. *Proceedings of the American Control Conference*, 4:2676–2683, 2005.

[31] Kazi Shah Nawaz Ripon, Kyrre Glette, Kashif Nizam Khan, Mats Hovin, and Jim Torresen. Adaptive variable neighborhood search for solving multi-objective

facility layout problems with unequal area facilities. *Swarm and Evolutionary Computation*, 8:1–12, 2013.

[32] Maghsud Solimanpur and Amir Jafari. Optimal solution for the two-dimensional facility layout problem using a branch-and-bound algorithm. *Computers and Industrial Engineering*, 55(3):606–619, 2008.

[33] Laurence A. Wolsey. Mixed Integer Programming. In *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., Hoboken, NJ, USA, sep 2008.

[34] Ramazan ahin. A simulated annealing algorithm for solving the bi-objective facility layout problem. *Expert Systems with Applications*, 38(4):4460–4465, 2011.