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 [19], and even decreases in expenses [68]. As a 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 [6]. 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 [8]. 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 [11]. 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) [40], 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. [21]) 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 [33], and usable area [28]. Facilities and/or machines must also not overlap. Solutions that satisfy the aforementioned conditions are called feasible solutions [48].

Generally, the facility layout problem is considered to be an **NP-Hard** problem [14]. 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 [33]. 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 [14]. 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 [2]. 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 [26]. It should be noted though that *some* instances of QAP are easy to solve [22]. 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 [59], 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 [67]. Mixed integer programming, when in the context of optimization, is also known to be NP-Hard [59]. 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 [7].

1.1.1 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 [25], [43], and [54]) 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 [44], while in the work of Ripon, K. S. N., et. al., Euclidean distance was used [60]. In works that derive from this formula, such as in [21], [63], and [55], 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 [14]. These constraints have been observed to be generally in many FLP works.

1.1.2 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 [14]. 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 [33]. 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 [14]. 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.

1.1.3 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 [21]. 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 [57]. 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 [33]. 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 [56]. 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 [13]. 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 [32]. 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 [13]. 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 [33].

1.1.4 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 [13]. 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) [14]. 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) [33] 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.

Figure 1.1: The Grey Wolf Optimization algorithm was inspired from the behaviour of grey wolves. Pictured are white wolves, different from grey wolves, but why pass up the opportunity to add a meme in a research paper? Profeshonal.

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 [50] [29]. 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 [3].

1.2.1 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 [50].

1.2.1.1 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 [29].

1.2.1.2 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:

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

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 [29]:

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

Note that the multiplication of vectors in the equations above is a component-wise multiplication, and not a dot product [49].

1.2.1.3 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 [29].

1.2.1.4 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

Facility layout problems is a well-known problem with many decades of research behind it. The benefits it provides in many situations, such as in factories and office spaces, while being a rather challenging problem to solve motivates the ongoing research for it. As mentioned in the previous chapter, facility layout problems are known to be NP-Hard. This makes the use of metaheuristics popular when it comes to solving FLPs. Based on our review, genetic algorithms are the most popular form of metaheuristics that have been used to solve various forms of facility layout problems [33]. Newer forms of metaheuristics, such as variable neighbourhood search, are being applied to FLPs more and more. Despite the popularity of the use of metaheuristics, exact methods have also been adapted to facility layout problems but not as widely used as metaheuristics. In this chapter, we will be reviewing many of the previous works available within the literature of facility layout problems. This will provide us with a good enough understanding about the current state of research around facility layout problems and allow us to find where this work may fit in the ocean of previous works. Due to the vastness of the field, we will only be focusing particularly on unequal area facility layout problems in this chapter. We will also be mentioning works that have been used in other types of facility layout problems. Additionally, most of the works mentioned here will be from the past 10 years.

2.1 Exact Methods

The survey of Hosseini-Nasab et al. (2017) showed that metaheuristics are popular approaches in solving facility layout problems. However, exact methods have also been used to solve FLPs [33]. Exact methods are algorithms that are able to find the optimal solution for an optimization problem [17]. However, they are not well-suited for large NP-Hard problems, such as the facility layout problem, due to the amount of time they will require in solving them. This does not mean that they are never used to solve NP-Hard problems. Small instances of those problems and multi-objective combinatorial optimization problems may still be solved by exact methods [36][18]. Numerous techniques may be used to improve the speed of these methods [66].

In 2006, Amaral, A. (2006) proposed a new mixed-integer linear programming model for the single-row facility layout problem (SRFLP). The author's model provided fewer continuous variables compared to the model he was comparing the new model against. Both models were solved with CPLEX 8.0 using a branch-and-bound method. It was found that the new model performed better than the previous one [5]. The branch-and-bound method solves optimization problems by exploring the entire search space [12]. It produces a search tree of subproblems and solves a subproblem on every iteration. This is repeated until no subproblems remain [52]. Solimanpur, M., and Jafari, A. (2008) developed a branch-and-bound algorithm to solve an instance of the facility layout problem. Their method managed to find good solutions for small and medium problem instances. However, in line with the expectations of the performance of exact methods, they found that it is inefficient for large-sized problem instances [63].

2.2 Works Using Metaheuristics

Exact methods have been used to solve many different problems, particularly those problems with known optimal solutions. Unfortunately, not all problems have known best solutions, and looking for them will take a reasonably long time to find [27]. Facility layout problems are under these types of problems. As such, metaheuristics are popular when it comes to solving FLPs [14]. This is further supported by the survey of Hosseini-Nasab et al. (2018) [33], where it is found that most papers they have surveyed used a metaheuristic to solve FLPs.

2.2.1 Genetic Algorithms

There are various forms of metaheuristics. Common of which is the genetic algorithm [33]. Genetic algorithm is a form of evolutionary-based metaheuristic. It works by breeding a generation of individuals from pairs of parents (through a crossover operation). The children produced from the breedings may undergo mutation to improve the diversity of the population and help find better solutions. This process is repeated until the algorithm reaches a certain number of generations, or a stopping condition has been met [45]. We allocated a section for discussing works that utilize genetic algorithms for facility layout problems due to its popularity in terms of use within the field [33].

2.2.1.1 Pure Genetic Algorithms

In literature, to our knowledge, there is no term called pure genetic algorithms. However, for the sake of ease of differentiation, we will be referring to the genetic algorithms in prior related works without any combination with other optimization algorithms as ”pure”. Genetic algorithms that have been combined with other algorithms will be called ”hybridized” genetic algorithms. These algorithms are discussed in the next subsection.

One work that uses pure genetic algorithms is that of Hasda et al. (2016). In their work, they attempted to solve the static unequal-area facility layout problem using a modification of the genetic algorithm. They have also used elitism in their modification. Their variation of the genetic algorithm still includes the traditional operators (despite being named differently in their paper), but with the inclusion of a rotation operator. The rotation operator is simply an operator that rotates a facility of a solution. It is similar to that of the mutation operator in that it only runs when a certain rotation probability is reached, and this probability is user-defined and is recommended to be of a small value. Their method has proven to be slightly better than the works they compared it to [31]. Another paper, proposed by Besbes et al. (2020) [9], also modifies the genetic algorithm for use with the facility layout problem. In most papers dealing with facility layout problems, the distance between the geometric centers of facilities considered in the objective function are computed using Euclidean or rectilinear distance. Besbes et al. changed this by using the A* algorithm to compute the distance more realistically and consider obstacles. This use of A* search has produced better solutions than when using the other two distance computation functions. Fernando, J., and Resende, M. (2015) modified the genetic algorithm to change the parent selection behaviour. Their method has the population partitioned into the elite individuals (those with the best fitness, and they are a small number) and non-elite individuals. During breeding, one parent will be from the elite partition and the other from the non-elite partition. The facilities are also arranged using maximal spaces and placing facilities in those spaces in such a way that it is as close to the rest of the facilities as possible. Their scheme created the better solutions for many of the datasets they applied it to compared to previous studies [28]. Placing facilities within a site layout, especially when considering multiple time periods, is another problem that may be considered to be under facility layout problems. Farmakis, P, and Chassiakos, A. (2018) developed a genetic algorithm to minimize the resource transportation costs between facilities or between facilities and work fields, and facility construction and relocation costs in a construction site considering changing requirements over time (an instance of the dynamic facility layout problem). According to the authors, their method produces ”rational solutions”, and the consideration for the changing demands over time produced a more effective layout than a static layout [21]. Similar to Farmakis, P, and Chassiakos, A. (2018), Peng et al. (2018) are also dealing with an instance of the dynamic facility layout problem. In their problem instance, they are also considering transport devices, such as conveyers and tow trains. A Monte Carlo simulation method has been used to generate scenarios, due to demand uncertainty. The crossover and mutation probability of an offspring in their genetic algorithm implementation is determined by its fitness relative to the fitness of the other individuals. The authors compared their genetic algorithm to particle swarm optimization and found that it produces the better results in all but two experiment data sets [55]. A genetic algorithm for facility layout problems can produce subjectively more desirable results when interactively given feedback from a decision maker. This idea is being utilized in the work of Garcia-Hernandez et al. (2013). In their work, they used two genetic algorithms to find an suboptimal layout. The first genetic algorithm is non-interactive and traditional, and only optimizes for material flow. The second genetic algorithm now takes into account the subjective evaluation by the decision maker, along with the material flow cost. This second algorithm is also partly based on NSGA-II, and only stops when the decision maker is satisfied with the results. The authors applied their genetic algorithm to two real-world cases, and found that their approach managed to capture the preferences of the decision maker and good solutions were generated in a reasonable number of iterations [25].

Genetic algorithms may also be applied to non-traditional configurations of FLPs. Barriga et al. (2014) used genetic algorithms to produce the best layout of buildings in a Protoss base in classic StarCraft. The fitness of a base's configuration is based on the health of its army, workers, and pylons [8].

2.2.1.2 Hybridized Genetic Algorithms

There are many other papers that modified genetic algorithms to solve facility layout problems. However, many of them did not only slightly modify the genetic algorithm. Rather, they also combined it with another algorithm, usually a local search algorithm. This resulted in **hybridized algorithms** that better exploited the search space of the solution produced by the genetic algorithm.

Asl et al. (2015) [7] and Asl, A. and Wong, K. (2015) [6] produced works that hybridized genetic algorithms with local search algorithms. The local search algorithms they used moved buildings in such a way that the solution generated is better than the original solution. The local search method used in the work of Asl, A. and Wong,

K. (2015) moves a building in different directions. This movement was performed for each building. The best new layout produced will replace the original solution if it is better than the original solution. The paper of Asl et al. (2015) also uses this local search method, and it is referred to as Local Search 1. The same paper also uses another local search method called Local Search 2. It works the same as Local Search 1. However, it moves two buildings at the same time. Both papers also utilize a swapping method in their genetic algorithms, which swaps facility positions to find a better arrangement.

Genetic algorithms has also been hybridized with variable neighbourhood search. Variable neighbourhood search (VNS) is a relatively recent local search algorithm introduced in 1997 by Mladenovic, N. and Hansen, P.. The algorithm utilizes and moves through a set of neighbourhood structures to find the local optimum [51], performed within the three phases of its main step [30]. In the paper of Uddin, M. (2015), genetic algorithm was used in conjunction with VNS. The author used the combined algorithm of GA-VNS to solve a problem instance of the dynamic facility layout problem. In each iteration, a percentage of the current population is subjected to breeding using a genetic algorithm, while the rest are optimized using VNS. This hybridized algorithm produced the same results with half of the datasets it was tested to compared to some of the previous works, while performing the best in two of the datasets, the worst in one, and the second best in the last [65].

Variable neighbourhood search is not the only local search algorithm that has been hybridized with genetic algorithms for solving facility layout problems. Simulated annealing has also been combined wth genetic algorithms. Simulated annealing (SA) is a local search algorithm inspired by annealing, which is a process that finds the low energy state of a metal by melting and then cooling it slowly [41]. The main idea behind SA is to slightly modify a solution to form a new solution, and that solution is only accepted when it is better than the older solution or with a certain probability when it is worse [16]. A hybrid of genetic algorithms and simulated annealing was used in the work of Pourvaziri, B., and Naderi, B. (2014) in order to solve another instance of the dynamic facility layout problem. Contrary to traditional genetic algorithms, their work utilizes multiple populations to find the solutions. Each population is involved independently of the other populations. These populations are then coalesced into a main population, which is now composed of the best individuals of the initial populations, after a pre-determined number of generations. The main population is then evolved, and the most fit solution from the population is further optimized using simulated annealing. This evolution and local search optimization is repeated until a stopping condition is met [58].

2.2.2 Non-Genetic Algorithms

Genetic algorithms are not the only metaheuristics that have been used to solve facility layout problems. Metaheuristics, such as particle swarm optimization, simulated annealing, and even relatively recent algorithms such as fireworks algorithms, have found application in facility layout problems.

Simulated annealing without hybridization with genetic algorithm have been used in FLPs. The work of Turgay, S. (2018) is one such example. Turgay, S. sought to solve an instance of the unequal-area facility layout problem with consideration for multiple objectives. Each objective is given a weight, determining its impact, in the mathematical model of his work. The values of the weights of each objective are obtained using Shannon's entropy rule. Based on experiments, the SA implementation is capable of producing usable layouts. However, its performance was not compared against other metaheuristics [64]. McKendall et al. (2006) also developed a simulated annealing implementation that they used for the dynamic facility layout problem. They modified the simulating annealing algorithm to integrate a look-ahead/look-back strategy into the algorithm from the work of McKendall, A. and Shang, J. (2006) [46]. They compared their modified SA with the traditional SA and a number of other algorithms, including a genetic algorithm implementation and a dynamic programming approach, through a set of experimental data. They discovered that their modified simulated annealing is effective in solving the dynamic facility layout problem, producing the best results in most of the problems in that large experimental dataset [47]. Another paper that used simulated annealing is that of Hosseini-Nasab, H., and Mobasheri, F. (2013). Their simulated annealing implementation utilized two mutation operators in generating neighbourhood solution. They added this modification to allow the algorithm to escape from local optimum, and allow for distinctions between solutions. They compared their work against GAMS, a modelling and optimization software [1]. Based on experimental results, their method can produce results significantly faster than GAMS, and can produce the best opttmum solution is mostly better or equal to the best optimum solution produced by GAMS [53]. It should be noted, however, that it may be better for them to have performed more runs for each method, compared to the five runs for their simulated annealing and one run for GAMS, to ensure that the results are statistically significant. Nevertheless, their work is still useful. Sahin, R. (2011) also developed a simulated annealing implementation for the facility layout problem. No modification to the simulated annealing algorithm was introduced. However, the mathematical model it is optimizing for considers the total material handling cost and the total closeness rating score. The author compared his work to two previous works, and found that the proposed SA approach produced same or better results than the previous works [68].

Genetic algorithms are a population-based optimization algorithm that have seen wide use in solving facility layout problems. But, it is not the only population-based optimization algorithm that has been used in facility layout problems. Particle swarm optimization (PSO) is an optimization algorithm that has seen use in FLPs as well. Particle swarm optimization is an optimization algorithms inspired by the social behaviour of birds in finding safe locations in which to land on. This optimization algorithm utilizes particles that perform search in a search space but keep note of the best global solution and personal best solution found so far, to which they will tend to move towards to, with parameter settings determining the movement behaviour [62]. Derakhshan Asl, A. and Wong, K. Y. (2017) are two researchers that have utilized particle swarm optimization in their work. In their work, they developed a modified particle swarm optimization algorithm that solves the static and dynamic versions of an instance of the unequal-area facility layout problem. They applied local search and swapping methods into PSO to improve the quality of solutions, and prevent local optima for both version of UA-FLP. They compared this algorithm to a number of previous works to which they have determined that it produces better results than the previous works [13]. Liu et al. (2018) developed a particle swarm optimization algorithm that optimizes a multi-objective function. Their algorithm also utilized objective space division method and a mutation operation and local search method to prevent facility overlaps. The algorithm was compared to previous works and was found to produce the best results in most of the experimental data set [44].

The metaheuristics simulated annealing, particle swarm optimization, and genetic algorithms first appeared decades ago. Simulated annealing was first proposed in 1983 [39]. while the genetic algorithm and particle swarm optimization were proposed in the 1990s [37][38]. Between the time the aforementioned algorithms were proposed and the time of writing of this paper, new optimization algorithms were proposed. Among these optimization algorithms is the coral reef optimization algorithm. Coral reef optimization (CRO) is based off of the formation and reproduction processes of coral reefs. In CRO, solutions are located in a grid initially partially populated by corals. A coral represents a solution, and the health of a coral represents its fitness. Corals in the grid sexually reproduce to produce larvae that are released into the water. Larvae settle in a grid depending on its health and the state of the grid cell they are attempting to settle in. Some corals are then made to asexually reproduce and occupy different parts of the grid with the same mechanism as larvae settling mentioned in the previous sentence. Some corals are also made to die to open up space for the next generation. These steps are performed until a stopping condition is met [61]. Garcia-Hernandez et al. (2019) utilized CRO in solving an instance of the facility layout problem and with the use flexible bay structures. No major modifications to CRO were used in their work. In their experimentations, they compared their CRO implementation with previous works, including those that do not use flexible bay structures as their layout representations. When comparing only against implementations with a flexible bay structure representation, their work produces the best results for most of the 17 cases. However, when considering a slicing tree structure layout as well, it only improves results for 7 of the cases [24]. The next year of the publication of their work, another paper combined coral reefs optimization with variable neighbourhood search. In this paper by Garcia-Hernandez (2020), the CRO algorithm remained as the original algorithm, but the larvae settling phase of the algorithm has been combined with VNS to further improve the larva/solution that is settling. Note that VNS is only ran when the larva is assured to occupy the grid cell it is settling towards. Their work also uses a relaxed flexible bay structure. The addition of VNS as well as the utilization of a relaxed flexible bay structure for layout representation has proven to be effective as it produced the better results than those generated in most of the previous related works [23].

Chapter 3

Statement of the Problem

In many industries and fields, arranging buildings, assets, or facilities of varying areas in positions that will remain the same for a long amount of time according a certain criteria may result in better productivity, reduced expenses, and improved operations efficiency. Unfortunately, the best arrangements are extremely difficult and take too long to obtain. As a matter of fact, problems like these are determined to be NP-Hard. Thus, it is important to develop an approach that lets us find arrangements that are good enough within a reasonable amount of time.

Chapter 4

Objectives

This study primarily aims to develop an approach that integrates a relatively new metaheuristic, Grey Wolf Optimization, to solve the unequal area static facility layout problem. Other objectives of this study are:

1. To evaluate the performance of the proposed approach in generating solutions to the unequal area static facility layout problem.

2. To evaluate the performance of the proposed approach with varying parameters for various aspects of the approach.

3. To compare the performance of the proposed approach to the performance of a genetic algorithm-based approach.

Chapter 5

Methodology

The methodology used in this research uses a modification of the classical Grey Wolf Optimization algorithm first introduced by Mirjalili, S., Mirjalili, S., and Lewis, A. in 2014 [50]. As we will be discussing in this chapter, we have determined that using classical GWO as is does not result in usable solutions for the instance of facility layout problem we are solving. Hence, the necessity for the modification.

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 (including the justification for the modification), 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 modification 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 the 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:

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

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

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

Figure 5.1: Visualization of the solution representation.

$A_1(i)$ 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 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

In this paper, we are adapting the Grey Wolf Optimization algorithm into solving our instance of the facility layout problem. There have been no publicly available research that have previously used the metaheuristic in solving FLP, basing from our survey. This increases the significance of this paper. As mentioned earlier, the proposed algorithm requires modifications in order to produce feasible solutions. We will first be discussing the reasons why we require them, before proceeding to detailing the algorithm we are using for this research.

5.3.1 The Problem with Classical GWO

In classical GWO, the following equations are used:

$$\vec{X}_1' = \vec{X}_\alpha(t) - \vec{A}_\alpha \cdot \vec{D}_\alpha \tag{5.3.1}$$

$$\vec{X}_2' = \vec{X}_\beta(t) - \vec{A}_\beta \cdot \vec{D}_\beta \tag{5.3.2}$$

$$\vec{X}_3' = \vec{X}_\delta(t) - \vec{A}_\delta \cdot \vec{D}_\delta \tag{5.3.3}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1' + \vec{X}_2' + \vec{X}_3'}{3} \tag{5.3.4}$$

where $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$ represent the α, β, and δ solutions [29]. $\vec{D}$ and $\vec{A}$ are defined as:

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

The aforementioned equations may be usable as is for other problems. However, as we have discovered through our experiments, using these equations results in solutions that are infeasible and where the buildings tend to move to the axes of an origin and the origin itself. One example solution with these characteristics is shown in Figure 5.2, where we have set the origin of the buildings to the center of the bounding region.

As one may infer, using the equations above will require setting an origin point for the buildings. Not considering the affinity of building towards the axes, having the origin point at the center or in a certain location in the bounding region restricts the possible locations where the buildings can cluster around. This restriction prevents us from exploring the solution subspace where solutions are feasible but where the cluster point is not the origin. This lead us to solutions that are less ideal. Aside from requiring setting the origin point, buildings moving towards the axes also presents another problem. Basing from our experiments, it prevents us from producing feasible solutions.

This behaviour can be attributed to the formulas, $\vec{D} = \left|\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)\right|$ and $\vec{A} \cdot \vec{D}$. Remember that the latter is from $\vec{X}_l' = \vec{X}_l(t) - \vec{A}_l \cdot \vec{D}_l$, where l represents the alpha, beta, or delta solution. To understand why the aforementioned formulas contribute to the behaviour we have discussed earlier, we should understand what any of the aforementioned formulas mean. It is helpful to simply consider that only

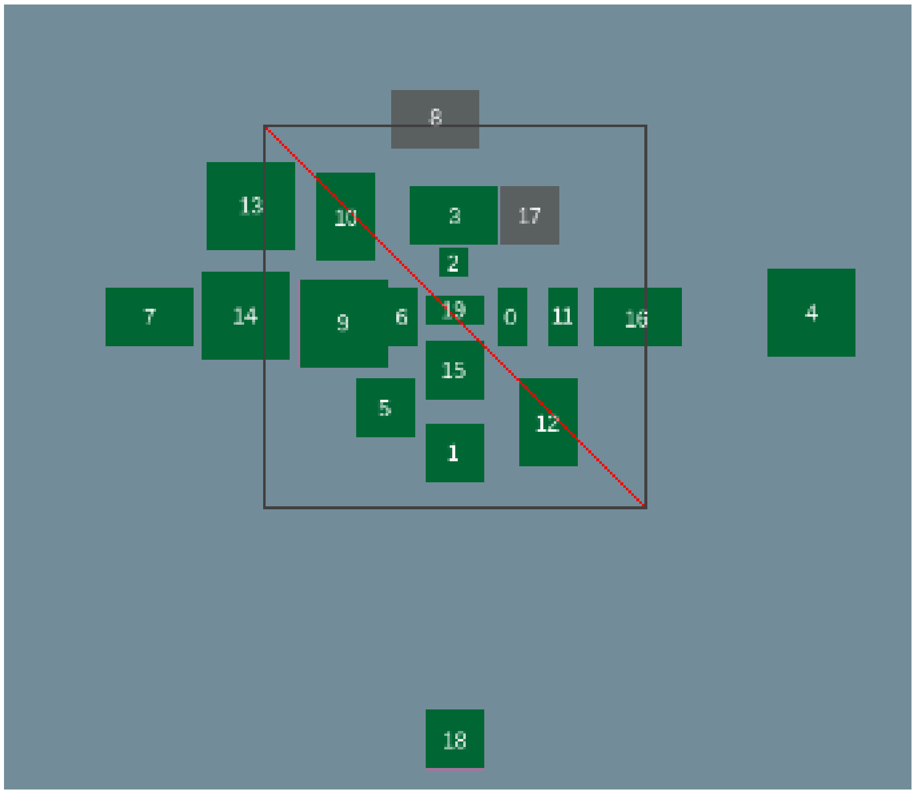

Figure 5.2: Flowchart detailing the algorithm.

building is being optimized in understanding the problems. Considering only the alpha solution may also provide better understanding as well.

Let us start with $\vec{C} \cdot \vec{X}_l(t)$ from $\vec{D} = \left|\vec{C} \cdot \vec{X}_l(t) - \vec{X}(t)\right|$. To simplify our explanation, let $K = \vec{C} \cdot \vec{X}_l(t)$. The range of each ith element in K will be $[0, 2 \cdot \vec{C}_{l,i}]$. Note that the operation is dot product, but pairwise multiplication. This means that K simply scales the x and y positions, and angle of the buildings. Figure 5.3 shows a visualization of this effect. Despite the figure only showing the effect with a building's position in the first quadrant, the same effect can be observed with other buildings located in other quadrants. Now, considering the entirety of $\vec{D}$, $\vec{D}$ would mean to be the distance between a building i moved to a different point in the region S (see

Figure 5.3) in $\vec{X}_l$ and a building i in $\vec{X}(t)$. A visualization for this is provided by Figure 5.4.

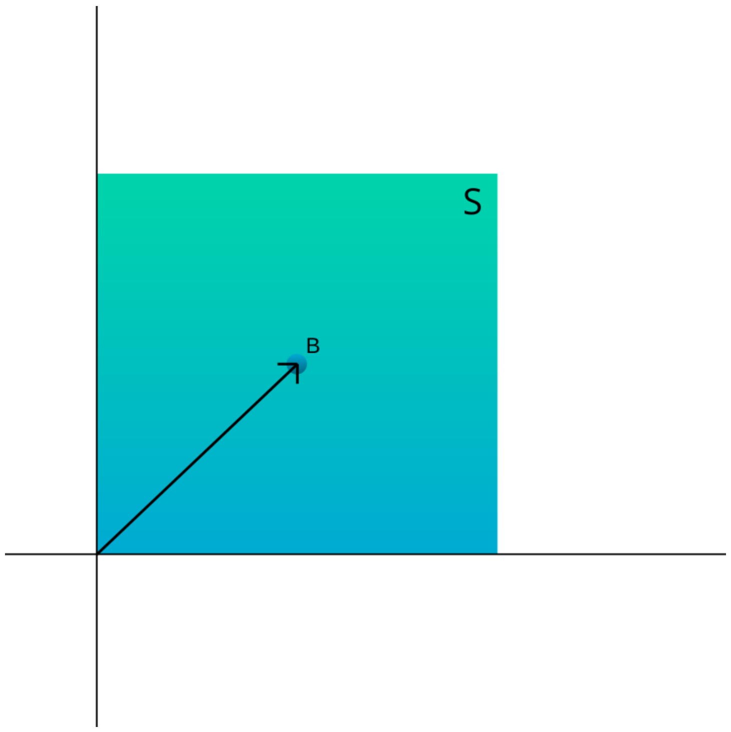

Figure 5.3: In K, C simply scales the x and y positions and angles of buildings. Assuming that the point B represents the x and y positions of a building, the region S is where B may be repositioned based on the values of C.

Since we now understand the meaning of $\vec{D}$, let us now move on to the other formula that causes the aforementioned problematic behaviour of classical GWO, $\vec{A} \cdot \vec{D}$. First, we should take note that $\vec{A} = 2 \cdot \vec{a} \cdot \vec{r_1} - \vec{a}$. a, as mentioned before, linearly decreases over time. Since a decreases linearly over time, $\vec{A}$ will also decrease over time. This behaviour of $\vec{A}$ would mean that in $\vec{A} \cdot \vec{D}$, $\vec{D}$ will eventually decrease as well. This behaviour in itself is not a problem. This is a necessity as the buildings will not cluster together (and eventually produce a solution with a lower fitness value) if not for this behaviour. The major issue lies with how this behaviour interacts with

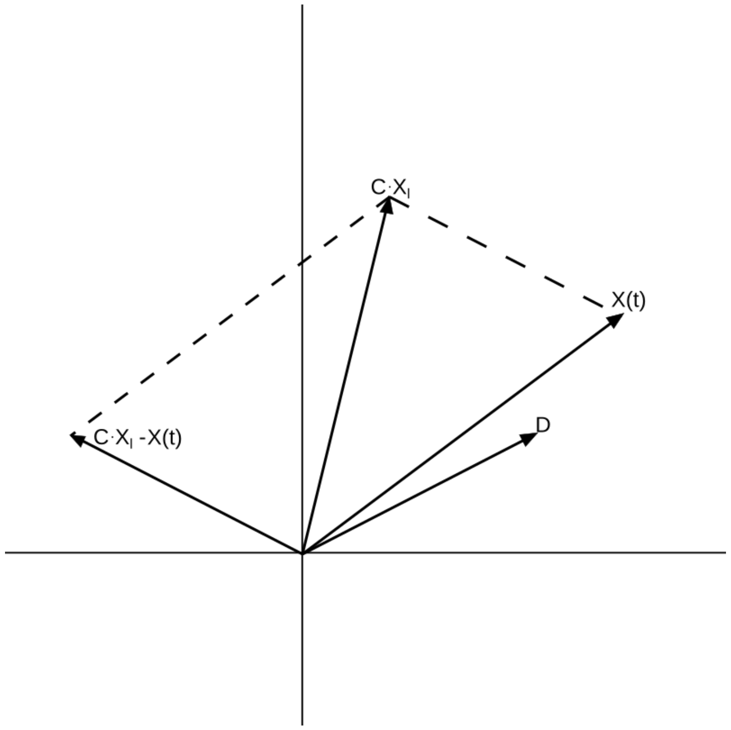

Figure 5.4: A visualization of how D is computed and its inherent meaning.

$\vec{D}$. Over time through the course of iterations, $\vec{A}$ will eventually scale down the $\vec{D}$. This scaling down will also eventually reduce the size of each of the region S of all buildings. This reduction of size will eventually cause the buildings to move towards the origin or the axes. Note that the penalty value for intersection prevents them from overlapping with one another. Buildings that are already on a certain axis will find it practically impossible to move in the direction of the perpendicular axis. Buildings that are on the origin itself will practically cease to move at all. Buildings will still be able to change their orientations, however. The reason for this behaviour of being stuck on an axis is due to the nature of axes themselves, where the value in one or both axes is zero. Since $K = \vec{C} \cdot \vec{X}_l(t)$ and when a building is near or already on an axis, the x, y, or both x and y positions of a building will barely, if at all, move away from the axes it is currently stuck to, when multiplying with $\vec{C}$. Hence, the behaviour we are noticing.

The two aforementioned formulas make classical GWO inadequate for our problem instance. We are unable to produce feasible nor satisfying results. In order for the grey wolf optimization algorithm to be successfully adapted into solving the facility layout problems, we must introduce a few changes into the algorithm. These changes will be discussed in the next subsection.

5.3.2 Modified GWO

Mirjalili, S., Mirjalili, S., and Lewis, A. [50] included a figure similar to Figure 5.5. It visualizes how a wolf ω will update its position based on the information provided by the leading wolves.

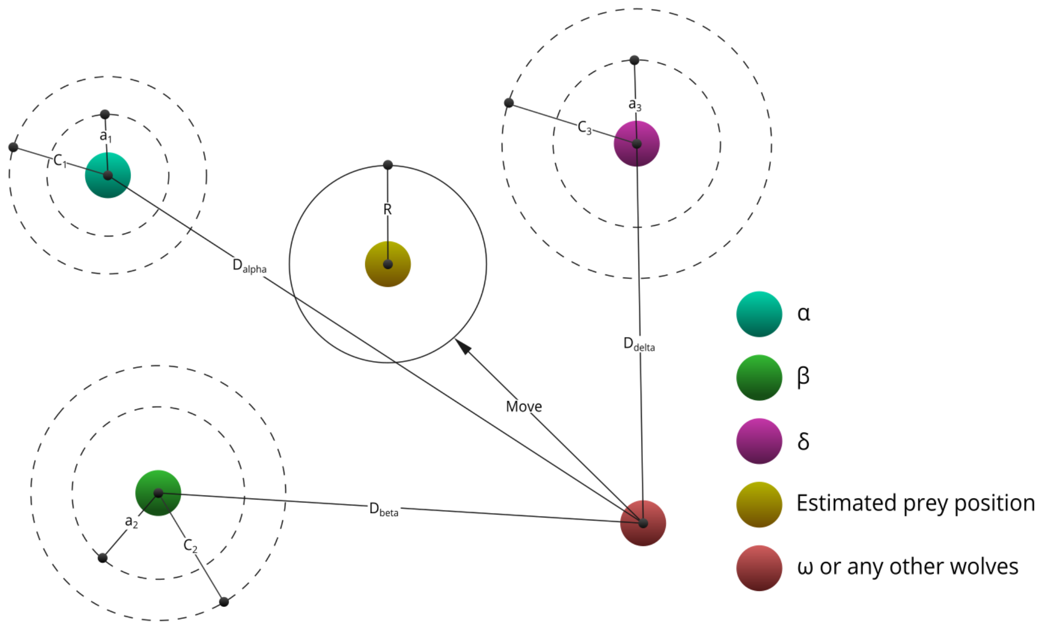

Figure 5.5: Visualization of how wolves in GWO update their positions. An ω wolf will move towards a random point inside the circle of the estimated prey position.

Basing from the visualization, notice that the $\vec{C}$ of the leading wolves specify the radius of the circle in which a $\vec{C} \cdot \vec{X}_l$ will be located it. The circle does **not** include an origin point. We have discussed before that performing a pairwise multiplication between $\vec{C}$ and $\vec{X}_l$ simply scales the elements i in the vector $\vec{X}_l$. This is different from the visualization. To achieve the same effect as the visualization, instead of performing pairwise multiplication, we must utilize vector addition between $\vec{C}$ and $\vec{X}_l$. See Figure 5.6 for a visualization of vector addition. This is the first modification we are introducing to classical GWO.

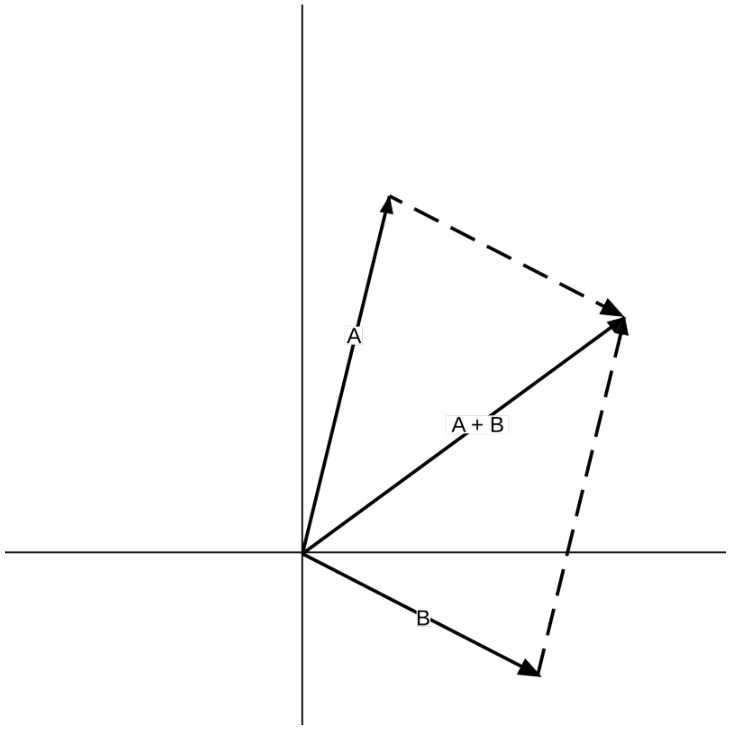

Figure 5.6: Vector addition pushes the point represented by $\vec{A}$ towards the direction of $\vec{B}$ by the magnitude of the same vector.

In our modified GWO, D is now defined as:

$$D = \left|(\vec{C} + \vec{X_l}) - \vec{X}(t)\right| \tag{5.3.5}$$

However, this alone is not enough to comply with the aforementioned visualization. Using this will only move the buildings to the right and/or top. In order for us to move the buildings, we must also modify $\vec{C}$ as shown below:

$$C = c \cdot \vec{r_3} \tag{5.3.6}$$

In this equation, c is a real-valued variable, and r_3 is a random vector in $[-1, 1]$. This modification will now allow us to move a building from any direction and at any magnitude. The magnitude in which the building will be moved is controlled by c.

These modifications remove the necessity to specify an origin point in the bounding region, and the behaviour of buildings to move towards the origin or axes. Unfortunately, this alone is not enough to produce feasible results. We have to add two more modifications before we are able to produce good results.

The first additional modification is the building clamping. Each building is restricted to the boundary. If a building is moved towards outside the boundary, it will be pulled back to within the boundary. Building clamping can be mathematically defined as the following. Given a building B in a solution $X(t)$ at iteration t after being updated by Equations 5.3.1 to 5.3.4, 5.3.5, and 5.3.6, clamping can be mathematically modeled as:

$$B_x = \max\left(R_x + \frac{B_w}{2}, \min\left(B_x, R_x + \left(R_w - \frac{B_w}{2}\right)\right)\right) \tag{5.3.7}$$

$$B_y = \max\left(R_y + \frac{B_h}{2}, \min\left(B_y, R_y + \left(R_h - \frac{B_h}{2}\right)\right)\right) \tag{5.3.8}$$

where B_x and B_y are the x and y positions of the centroid of a building B, B_w and B_h are the width and height from the top left corner of a building B, R_x and R_y are the x and y positions of the top-left corner of the bounding region R, and R_w and R_h are the width and height of the bounding region R. Based on our prior experiments, without this clamping, buildings will freely move to points outside the boundary, and, at the end of the run, will produce a bad solution.

This clamping should *almost* solve the positioning of the buildings and allow us to produce results that are feasible. Since GWO is a continuous metaheuristic, building attributes that must only be one of two values will eventually be a value that is between the two aforementioned values. In our problem, this attribute that is affected is the building orientation. The building orientation may only be 0° or 90°. It must never be a value between two. To solve this problem, we simply use the orientation of a building B from the α, β, or δ solutions, which are randomly selected. This idea is based off from the nature of GWO, where the best three solutions lead the search for the local optima. The building orientation of a building B is, therefore, obtained using:

$$B_o = \begin{cases} \alpha_{B_o} & \text{if } 0 \leq r < \frac{1}{3} \\ \beta_{B_o} & \text{if } \frac{1}{3} \leq r < \frac{2}{3} \\ \delta_{B_o} & \text{otherwise} \end{cases} \tag{5.3.9}$$

where B_o is the current orientation of a building B, α_{B_o}, β_{B_o}, and δ_{B_o} are the orientations of building B in the α, β, and δ solutions, respectively, and r is a random variable in $[0, 1]$. In our approach, assigning the building orientations is performed before clamping the buildings.

With all these modifications already discussed, we now need to briefly discuss how population initialization performed. The population is initialized by providing each building B a random x and y position values, and random orientation. The orientation is either 0 or 90. The x and y positions are clamped as well to ensure that the buildings are inside the bounding region. The positions are clamped using Equations 5.3.7 and 5.3.8, respectively. Algorithm 1 shows the pseudocode for the population initialization.

Algorithm 1 Pseudocode for the population initialization.

```
1: Set $\vec{X}$ to be the solution.
2: Set $R_x$ to be the x position of the top-left corner of the bounding region $R$.
3: Set $R_y$ to be the y position of the top-left corner of bounding region $R$.
4: Set $R_w$ to be the width of the bounding region $R$.
5: Set $R_h$ to be the height of the bounding region $R$.
6: Set $N$ to be the number of buildings in a population.
7: for i = 0 until $N$ do
8:     $\vec{X}_{(i*3)} = U(R_x, R_x + R_w)$
9:     $\vec{X}_{(i*3)+1} = U(R_y, R_y + R_h)$
10:    $\vec{X}_{(i*3)+2} = U(0, 90)$
11:    Apply Equation 5.3.7 to $\vec{X}_{(i*3)}$.
12:    Apply Equation 5.3.8 to $\vec{X}_{(i*3)+1}$.
13: end for
```

All these modifications for the classical GWO have allowed us to successfully adapt GWO to the facility layout problem. Equations 5.3.10 to 5.3.18, and Algorithm 2 summarises the entire modified GWO. Notice that these modifications are relatively simple, and do not significantly change the characteristics of classical GWO. The simplicity of GWO is still preserved. The next chapters will discuss how our modified version of GWO performs against configurations of unequal-area facility layout ptoblems.

$$\vec{A} = 2\vec{a} \cdot \vec{r_1} - \vec{a} \tag{5.3.10}$$

$$\vec{C} = c \cdot \vec{r_2} \tag{5.3.11}$$

$$\vec{D}_\alpha = \left| \left(\vec{C}_1 + \vec{X}_\alpha \right) - \vec{X} \right| \tag{5.3.12}$$

$$\vec{D}_\beta = \left| \left(\vec{C}_2 + \vec{X}_{beta} \right) - \vec{X} \right| \tag{5.3.13}$$

$$\vec{D}_\delta = \left| \left(\vec{C}_3 + \vec{X}_\delta \right) - \vec{X} \right| \tag{5.3.14}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha \tag{5.3.15}$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta \tag{5.3.16}$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \tag{5.3.17}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{5.3.18}$$

(**COMMENT**: Sir, can we give this modified GWO a name? mGWO is already taken so maybe The Ballais-Romero GWO Variant? Still haven't came up with an alternative name. HAHAHAHAHAHAHA. **REMOVE THIS IN THE FINAL DRAFT, SEAN.**)

5.4 Implementation Technologies

Our program was developed using C++17 compiled using the Clang 11 compiler in an elementaryOS Hera environment under release mode with the `-O3` optimized compilation flag turned on. Building was handled by CMake, and package management was

Algorithm 2 Pseudocode for the proposed modified GWO.

```
1: Set $T$ to be the maximum number of iterations.
2: Initialize the grey wolf population $\vec{X}_i (i = 1, 2, \ldots, n)$
3: Initialize $c$, and $a = 2$.
4: Calculate the fitness of each wolf.
5: Set $\vec{X}_\alpha$ to be the fittest wolf.
6: Set $\vec{X}_\beta$ to be the second fittest wolf.
7: Set $\vec{X}_\delta$ to be the third fittest wolf.
8: while t < T do
9:     for each wolf $\vec{X}_i$ do
10:        Initialize $\vec{A}_1$, $\vec{A}_2$, $\vec{A}_3$, $\vec{C}_1$, $\vec{C}_2$, and $\vec{C}_3$.
11:        Update the position of the current wolf $\vec{X}_i$ using Equations 5.3.12 to 5.3.18.
12:        Set the orientation of each building using Equation 5.3.9.
13:        Clamp the buildings in $\vec{X}_i$ using Equations 5.3.7 and 5.3.8.
14:    end for
15:    Calculate the fitness of each wolf.
16:    Update $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$.
17:    $a = 2 - \frac{2t}{T}$
18:    $t = t + 1$
19: end while
20: return $\vec{X}_\alpha$
```

handled by Conan. Our implementation is built on top of CoreX, a custom-developed 2D game engine. Using a game engine allowed us to visualize the results and configure experiments in a graphical manner. Using a custom engine over an over-the-shelf engine ensures that the implementation remains light and does not carry unnecessary features that are typically used in commercial game engines. 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.

Chapter 6

Validation of the Approach

Our proposed approach is validated by running it through three data sets based off of the data sets used by Hasda, R., Bhattacharjya, R., and Bennis, F. (2017) [31], and Lee, Y., and Lee, M. [42] that will test its performance and compare it to a genetic algorithm-based approach. Basing from previous studies, the fitness of the solution produced by an approach determines the performance of an algorithm. As such, we will be comparing the average fitness values of our approach and two competing approaches, a modified genetic algorithm approach, and a particle swarm optimization-based approach. 30 feasible solutions are obtained with each approach and data set, and the fitnesses obtained in each run are averaged. Note that, however, only the fitnesses of feasible solutions are included in the average. Due to the non-deterministic nature of metaheuristics, infeasible solutions are bound to be generated.

6.1 Data Sets Used

Let us call the data sets used as problem configuration. The first two problem configurations are based off of the configurations used by Hasda, R., Bhattacharjya, R., and Bennis, F. (2017) [31]. The first configuration used, which we will call SFLP-II, is shown in Table 6.1, contains 8 buildings, and the second configuration, which we will call mSFLP-III, is shown in Table 6.2, contains 20 buildings. The second configuration is called as such due to the fact that it is a modification of the third problem configuration used by Hasda, R., Bhattacharjya, R., and Bennis, F.. SFLP-II uses a 12x12 bounding region, while mSFLP-III uses a 260x260 bounding region. The third configuration, which we will call mLeeKra30a, is shown in Table 6.3, contains 30 buildings. The configuration consists of modified building dimension data from the 30-building data set by Lee, Y., and Lee, M. [42] and cost data from the Kra30a data set in QAPLIB [10]. A 250x250 bounding region is used for the data set.

	Cost of Material Flow Between Buildings								W	H
Building	1	2	3	4	5	6	7	8		
1	0	1	2	0	0	0	2	0	2	3
2	0	0	4	3	6	0	0	2	4	5
3	0	0	0	2	0	3	1	0	2	2
4	0	0	0	0	5	2	0	2	3	3
5	0	0	0	0	0	0	0	4	2	4
6	0	0	0	0	0	0	4	0	4	4
7	0	0	0	0	0	0	0	1	4	4
8	0	0	0	0	0	0	0	0	3	4

Table 6.1: Configuration of SFLP-II. W and H mean width and height, respectively.

6.2 Competing Approaches

In order for us to properly gauge the performance of our GWO approach for the unequal area static facility layout problem, we will be solving the problem configurations using two other approaches: (1) a modified genetic algorithm approach, and a particle swarm optimization-based approach. These two were chosen due to their popularity in solving facility layout problems [33].

	Cost of Material Flow Between Buildings																				W	H
Building	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	20	40
2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	40	40
3	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	20	20
4	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	40	60
5	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	60	60
6	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	40	40
7	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	40	20
8	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	40	60
9	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	60	40
10	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	60	60
11	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	40	60
12	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	20	40
13	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	60	40
14	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	60	60
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	60	60
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	40	40
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	60	40
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	40	40
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	40	40
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	40	20

Table 6.2: Configuration of mSFLP-III.

6.2.1 Modified Genetic Algorithm Approach

The competing GA approach, which we will compare our proposed GWO approach, 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) [7] and Asl, A. and Wong, K. (2015) [6]. We will be further discussing the algorithm in detail in this section.

6.2.1.1 Population Generation

In the competing GA approach, the population generation is the same the method for initialization the population as the one in our proposed modified GWO approach.

6.2.1.2 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 3.

6.2.1.3 Selection, Crossover, Mutation, and Elitism

Key parts of a genetic algorithm are the selection, crossover, and mutation operators. These drive the algorithm to produce good solutions. We will be discussing each operator used 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.

Algorithm 3 Pseudocode for the swapping method.

```
1: Let $S$ be the collection of generated solutions.
2: Set $S_{curr}$ be the current solution.
3: Add $S_{curr}$ to $S$.
4: Set $N_B$ be the maximum number of buildings.
5: for i = 0 until $N_B - 2$ do
6:     for j = i + 1 until $N_B - 1$ do
7:         Building $i$'s orientation in $S_{curr}$ is changed to the other orientation,
           $\hookrightarrow$ and the resulting new solution is added to $S$
8:         Building $j$'s orientation in $S_{curr}$ is changed to the other orientation,
           $\hookrightarrow$ and the resulting new solution is added to $S$
9:         Building $i$'s and $j$'s orientations in $S_{curr}$ are changed to the other
           $\hookrightarrow$ orientation, and the resulting new solution is added to $S$
10:        Building $i$'s and $j$'s positions are exchanged in $S_{curr}$, and the resulting new
           $\hookrightarrow$ solutiA movement and a orientation changeon is added to $S$.
11:        Building $i$'s and $j$'s positions are exchanged in and the orientation of
           $\hookrightarrow$ building $i$ is changed in $S_{curr}$, and the resulting new solution
           $\hookrightarrow$ is added to $S$.
12:        Building $i$'s and $j$'s positions are exchanged in and the orientation of
           $\hookrightarrow$ building $j$ is changed in $S_{curr}$, and the resulting new solution
           $\hookrightarrow$ is added to $S$.
13:        Building $i$'s and $j$'s positions are exchanged in and the orientations of
           $\hookrightarrow$ buildings $i$ and $j$ are changed in $S_{curr}$, and the resulting new solution
           $\hookrightarrow$ is added to $S$.
14:    end for
15: end for
16: return the best solution in $S$.
```

6.2.1.3.1 Selection

The selection operator used in the competing approach is tournament selection. Tournament selection works by selecting k (also known as tournament size) individuals from a population and using the best two selected individuals as parents for an offspring [4]. Algorithm 4 shows a pseudocode of tournament selection. Note that in the algorithm, P_p denotes the p-th solution in P, and $|P|$ denotes the population size.

Algorithm 4 Pseudocode for the tournament selection.

```
1: Set $X^{(1)}$ to be the first best parent.
2: Set $X^{(2)}$ to be the second best parent.
3: Set $P$ to be the population of solutions.
4: Set $k$ to be the tournament size.
5: Set $f(X)$ to be the fitness function.
6: $X^{(1)} = $ null
7: $X^{(2)} = $ null
8: for $i = 1, 2, \ldots, k$ do
9:     $p = U(1, |P|)$
10:    if $X^{(1)} ==$ null or $f(P_p) < f(X^{(1)}$ then
11:        $X_{(2)} = X^{(1)}$
12:        $X_{(1)} = P_p$
13:    else if $X^{(2)} ==$ null or $f(P_p) < f(X^{(2)}$ then
14:        $X_{(2)} = P_p$
15:    end if
16: end for
17: return $X^{(1)}$, and $X^{(2)}$.
```

6.2.1.3.2 Crossover

The crossover operator used in the competing approach is uniform crossover. Uniform crossover works by selecting a gene from a random parent and placing it in the offspring. This is applied for every gene in the offspring. Algorithm 5 shows a pseudocode of uniform crossover.

Algorithm 5 Pseudocode for the uniform crossover.

```
1: Set $X$ to be the offspring.
2: Set $N$ to be the number of genes a solution has.
3: Set $X^{(1)}$ to be the first parent.
4: Set $X^{(2)}$ to be the second parent.
5: for $i = 1, 2, \ldots, N$ do
6:     rand $= U(0, 1)$
7:     if rand $< 0.5$ then
8:         $X_i = X_i^{(1)}$
9:     else
10:        $X_i = X_i^{(2)}$
11:    end if
12: end for
13: return $X$.
```

6.2.1.3.3 Mutation

Over time, as more and more iterations are performed, the diversity of the population will eventually be lost. 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 6 and Figure 6.1, respectively.

Algorithm 6 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 $O$.
```

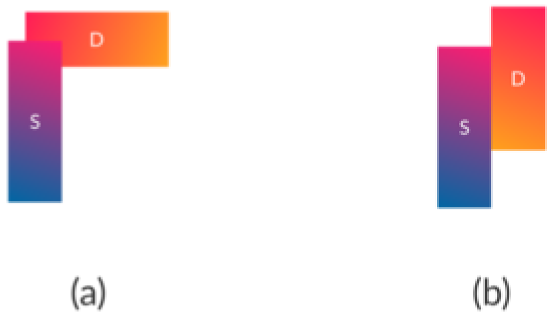

Figure 6.1: Visualization of how Buddy-Buddy Mutation works. On the left are two buildings that are overlapping one another. The right shows the same buildings but with the mutation applied, causing them to no longer overlap. Note that the right shows only one possible arrangement for both buildings.

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 [34]. The mutation rate is mathematically modelled as:

$$m(X,t) = 1 - \frac{fit_{max}(t) - fit(X(t))}{fit_{max}(t) - fit_{min}(t)} \tag{6.2.1}$$

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.

6.2.1.3.4 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 [15]. We are also taking this principle into our competing GA approach. In the competing approach, we are keeping the best E_N solutions in the previous iteration to the next iteration.

6.2.1.4 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 [60]. 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.

6.2.1.4.1 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 6.4. Additionally, pseudocode of the search algorithm is shown in Algorithm 7.

Algorithm 7 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 $S$.
```

6.2.1.4.2 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 6.5, and a pseudocode of the search algorithm is shown in Algorithm 8.

Algorithm 8 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 $S$.
```

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 6.5: Activities for moving a building in Local Search 2

6.2.1.5 GA Summarized

The entire competing GA algorithm is summarized in pseudocode with Algorithm 9. Note that our implementation of the competing GA algorithm produces two children during the crossover phase. If there is no space for the second child in the current geenration, the worst offspring will be replaced by the second child.

Algorithm 9 Pseudocode for the competing GA approach.

```
1: Initialize population $P$.
2: Calculate the fitness of each solution in $P$.
3: Set $T$ to be the number of generations.
4: Set $X^{(best)}$ to be the best solution found.
5: Set $E_N$ to be the number of elite solutions that will be kept in the next generation.
6: Sort $P$ from lowest to highest fitness value.
7: for $t = 1, 2, \dots T$ do
8:     if $t \leq 100$ then
9:         Apply swapping method.
10:    end if
11:    for $i = E_N + 1, \dots, |P|$ do
12:        Select parents $X^{(1)}$ and $X^{(2)}$ using tournament selection.
13:        Crossover parents and produce offspring $X^{(o)}$.
14:        Mutate $X^{(o)}$ if mutation probability allows.
15:        $P_i = X^{(o)}$.
16:    end for
17:    Sort $P$ from lowest to highest fitness value.
18:    Apply Local Search 1 to the best solution in $P$.
19:    if $t \geq T - 50$ then
20:        Apply Local Search 2 to the best solution in $P$.
21:    end if
22: end for
23: return best solution in $P$.
```

6.2.1.6 Particle Swarm Optimization

Particle swarm optimization (PSO) is a metaheuristic inspired by the social behaviour of animals. It was proposed by Kennedy, J. and Eberhart, R. [38]. A population of solutions is called a swarm, and each solution is called a particle P. Each particle has position and velocity vectors. The position vector X_i^t is the solution itself, while the velocity vector V_i^t which influences how each particle changes its position. During each iteration of a run, the particle's position is updated to a different position based on the swarm's best found position gbest so far, the particle's personal best found position pbest so far, and the current velocity vector V_{ij}^t. This behaviour is mathematically defined using the following equations.

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + c_1\vec{r}_1^t\left(\vec{pbest}_i - \vec{X}_i^t\right) + c_2\vec{r}_2^{\,t}\left(\vec{gbest}_i - \vec{X}_i^t\right) \tag{6.2.2}$$

$$\vec{X}_i^{t+1} = \vec{X}_i^t + \vec{V}_i^{t+1} \tag{6.2.3}$$

In Equation 6.2.2, w is the inertial weight constant, and is important for balancing between exploration and exploitation. It also determines how much the previous velocity will influence the current velocity. The second term in the equation is called the individual cognition term. This calculates the difference between the particle's own best position and current position. It is multiplied by the individual-cognition parameter, c_1, which influences how important the particle's previous experiences are. $\vec{r}_1$ is a random vector that has a range of $[0, 1]$. This helps the algorithm avoid premature convergences. The last term is the social learning term. This allows the swarm to share information to each other about the best global position found so far. Similar to the individual cognition term, this term computes the distance between the particle's current position, and the swarm's best known position. This term also attracts particles towards the gbest. c_2 is the social learning parameter, and it determines how influential the global learning of the swarm is. $\vec{r}_2$ does the same task as $\vec{r}_1$. Equation 6.2.3 finally updates the current position of a particle.

In our approach, little was changed from the classical particle swarm optimization algorithm. Population/Swarm generation is the same as in our approach. The initial velocity was for all particles is set to 0, as per recommendation by Engelbrecht, A. [20]. We also clamped the building to within the bounding region. Our prior experiments showed that without this clamping, many buildings would be positioned outside of the boundary. Clamping is done the same way as in our approach. Since PSO is a continuous metaheuristic, we would not be able to obtain a building orientation that is either 0 or 90 immediately after using equations 6.2.2 and 6.2.3. A building's orientation B_o may be any real value. As such, the current building orientation is also computed using the following equation, applied after using the PSO equations:

$$B_o = \begin{cases} 0 & \text{if } B_o \bmod 360 < 180 \\ 90 & \text{otherwise} \end{cases} \tag{6.2.4}$$

In Equation 6.2.4, the B_o is modulo-ed by 360 to ensure that the range of values are within the range of $[0, 360)$. This range was selected since all angles larger than $359.9\bar{9}9$ are symmetries with the values in the aforementioned range.

The entire PSO approach is summarized in pseudocode in Algorithm 10.

Algorithm 10 Pseudocode for the competing PSO approach.

```
1: Set $T$ to be the maximum number of iterations.
2: Initialize $w$, $c_1$, and $c_2$.
3: Initialize the swarm $\vec{X}_i (i = 1, 2, \ldots, n)$
4: Set the velocity $\vec{V}_i^t$ of each particle $i$ to 0.
5: Calculate the fitness of each particle.
6: Set the current solution of each particle $i$ as their personal best pbest$_i$.
7: Set the gbest to be the best solution in the swarm.
8: while t < T do
9:     for each particle $\vec{X}_i$ do
10:        Initialize $\vec{r}_1$ and $\vec{r}_1$.
11:        Update the position of the current particle $\vec{X}_i$ using Equations 6.2.2 and 6.2.3.
12:        Set the orientation of each building in $\vec{X}_i$ using Equation 6.2.4.
13:        Clamp the buildings in particle $i$ using Equations 5.3.7 and 5.3.8.
14:        Calculate the fitness of $\vec{X}_i$.
15:        if $f(\vec{X}_i) < f(\text{pbest}_i)$ then
16:            pbest$_i = \vec{X}_i$
17:        end if
18:        if $f(\vec{X}_i) < f(\text{gbest})$ then
19:            gbest $= \vec{X}_i$
20:        end if
21:    end for
22:    $t = t + 1$
23: end while
24: return gbest
```

	Cost of Material Flow Between Buildings																														W	H
Building	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30		
1	0	2	2	2	2	2	2	2	3	0	3	2	0	0	0	3	3	0	0	2	1	2	1	1	0	3	0	0	0	0	56	43
2	2	0	2	2	3	2	3	2	0	0	2	1	0	0	2	3	3	0	0	2	1	2	1	1	0	3	0	0	0	0	37	43
3	2	2	0	2	2	3	0	0	0	0	2	2	0	0	2	3	3	0	0	2	1	2	1	1	0	3	0	0	0	0	78	47
4	2	2	2	0	0	2	2	3	1	0	0	2	1	0	0	4	3	0	0	2	1	2	1	1	0	3	0	0	0	0	40	20
5	2	3	2	2	0	2	3	2	0	0	2	1	0	0	2	3	3	0	0	2	1	2	1	1	0	3	0	0	0	0	42	50
6	2	2	3	2	2	0	0	0	0	0	2	2	0	0	2	3	3	0	0	2	1	2	1	1	0	3	0	0	0	0	50	35
7	2	3	0	3	3	0	0	4	4	3	3	0	4	0	0	2	2	0	0	3	4	1	0	0	0	0	0	0	0	0	20	18
8	2	2	0	1	2	0	4	0	0	0	0	2	0	0	0	3	3	0	0	3	1	1	0	0	0	0	0	0	0	0	40	39
9	3	0	0	0	0	0	4	0	0	0	0	1	4	0	0	2	2	0	0	2	0	1	0	0	0	0	0	0	0	0	40	50
10	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	4	0	0	40	49
11	3	2	2	2	2	2	3	0	0	0	0	3	4	0	0	3	3	0	0	0	1	2	0	0	0	2	0	0	0	0	40	40
12	2	1	2	1	1	2	0	2	1	0	3	0	3	0	0	1	1	0	0	0	0	2	2	0	0	0	0	0	0	0	32	28
13	0	0	0	0	0	0	4	0	4	0	4	3	0	4	0	3	3	0	0	2	1	1	1	0	0	0	0	4	0	0	30	30
14	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	4	0	0	0	1	0	0	0	3	0	0	0	0	50	50
15	0	2	2	4	2	2	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	2	0	0	0	3	0	2	0	0	20	20
16	3	3	3	3	3	3	2	3	2	0	3	1	3	0	1	0	5	2	0	0	0	1	0	0	0	2	0	0	0	0	17	18
17	3	3	3	3	3	3	2	3	2	0	3	1	3	0	1	4	0	2	0	0	0	1	0	0	0	2	0	0	0	0	20	20
18	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	2	2	0	0	0	0	1	0	0	2	4	4	0	0	0	30	30
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	25	18
20	2	2	2	2	2	2	3	3	2	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	3	0	31	16
21	1	1	1	1	1	1	4	1	0	0	1	0	1	0	0	0	0	0	0	1	0	1	2	0	0	0	0	0	4	0	43	37
22	2	2	2	2	2	2	1	1	1	1	2	2	1	1	2	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	32	28
23	1	1	1	1	1	1	0	0	0	1	0	2	1	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	4	0	30	30
24	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	4	0	52	48
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	0	4	0	0	0	0	20	20
26	3	3	3	3	3	3	0	0	0	0	2	0	0	3	3	2	2	4	0	0	0	0	0	0	5	0	0	0	0	4	17	18
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	20	20
28	0	0	0	0	0	0	0	0	0	4	0	0	4	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	30
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	4	0	4	4	0	0	0	0	0	0	22	21
30	0	0	0	0	0	0	00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	18	28

Table 6.3: Configuration of mLeeKra30a. W and H mean width and height, respectively.

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 6.4: Activities for moving a building in Local Search 1

Chapter 7

Results and Discussion

The results of our experiments with our data set will be presented in this chapter. 30 runs of each approach that produced feasible solutions are included in the results. Note that some runs produce an infeasible solution. This due to the non-deterministic nature of metaheuristics, which will cause it to produce infeasible solutions sometimes.

7.1 Environment

All of the approaches were run in the following hardware and software configurations:

- Hardware
  - **CPU**: Intel Core i3-5010U @ 2.1GHz
  - **GPU**: NVIDIA GeForce 920M
  - **RAM**: 4GB
- Software
  - **OS**: elementaryOS 5.1.7 Hera
  - **Linux Kernel Version**: 5.4.0-80-generic

7.2 Experiments

Each approach has their own parameters, and the values we have set for those parameters are shown in Table 7.1. Both approaches use a population size of 50, and a maximum number of iterations of 400. The following parameter values for the PSO approach were taken from the work of Jolai, F., Tavakkoli-Moghaddam, R., and Taghipour, M. [35].

Approach	**Parameter**	**Value**
GWO	c	4
GA	Mutation Rate	0.05
	Tournament Size	4
	No. of Elites (EN)	5
PSO	w	0.05
	c1	2
	c2	2

Table 7.1: Parameter values of the GWO, GA, and PSO approaches.

The results obtained for each approach is shown in Tables 7.2, 7.3, and 7.4, respectively. As what the tables show, the competing genetic algorithm approach produces a solution that is better than our proposed GWO approach and the PSO approach, with a fitness averages of 273.537430766667, 50422.3174052667, and 88087.9226730667 for the SFLP-II, mSFLP-III, and mKra30a problem configurations respectively. This is compared to our proposed approach's fitness averages of 290.3857809, 52702.9314929, 101874.328208933. Fortunately for our approach, the PSO approach obtained the fitness averages of 324.356679733333, 64181.9165261333, and 118785.277772567, proving that our GWO is not the worst approach. For SFLP-II, the best and worst solutions have fitnesses of 221.042599 and 324.874681 for the GA, respectively, compared to our approach's 234.250481 and 339.099181. The PSO approach obtained

268.59568 and 380.688077s. In this data set, our approach was capable of producing the best solution. For mSFLP-III, the best and worst solutions have a fitness of 46802.666237 and 53474.353325 for the GA, respectively, compared to our approach's 48951.787331 and 53474.353325 and the PSO approach's 60037.826591 and 68194.108383. Lastly, for mKra30a, the best and worst solutions have fitnesses of 76651.01432 and 98512.468674, respectively, with our approach obtaining 90455.74585 and 118315.534424. PSO produces the poorest best and worst solutions with fitnesses of 106178.045845 and 130915.770554.

From the results, the competing GA approach is the most stable among the three, basing from the lower standard deviation in all data sets, with 25.3555205871573, 1630.42444301206, and 5077.72744984237 for SFLP-II, mSFLP-III, and mKra30a, respectively. This is compared to our approach's 32.4373567833344, 2224.64491886288, and 7190.18569614101, and the PSO approach's 38.4539160542525, 1956.45733138936, and 6138.39410532314.

The genetic algorithm approach is also faster when SFLP-II is being used with an average run time of 27s, compared to our approach's 89.5666666666667s and the PSO approach's 89.5666666666667s. However, as the number of buildings increase, the average runtime of the GA approach becomes worse compared to the two other approaches. With mSFLP-III, GA takes 145.766666666667s, while our approach and the PSO approach takes 197.733333333333s and 341.266666666667s, respectively. GA is still faster than GWO in this data set, but it is already slower than the PSO approach. Moving towards mKra30a, we can see that GA now takes 341.266666666667s. This is longer than our approach's 341.266666666667s, and the PSO approach's 114.366666666667s. Figure 7.1 shows this observation.

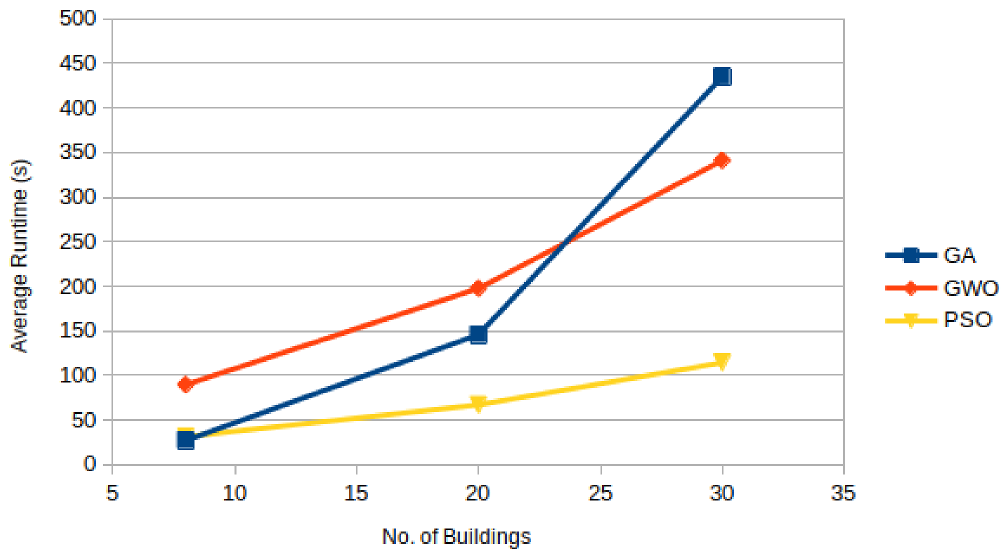

Figure 7.1: The average runtime (s) of each of the approaches as the number of buildings in a data set increase.

Problem	**Genetic Algorithm**				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	221.042599	324.874681	273.537430766667	25.3555205871573	27
mSFLP-III	46802.666237	53474.353325	50422.3174052667	1630.42444301206	145.766666666667
mKra30a	76651.01432	98512.468674	88087.9226730667	5077.72744984237	435.033333333333

Table 7.2: Results obtained from using the competing GA approach.

Figures 7.2 and 7.3 show the fitness graphs of the best solutions using the SFLP-II and mSFLP-III data sets. The non-linearity of the graph of our GWO approach that is obvious in Figure 7.2 is due to the nature of our approach. All solutions in a population are replaced. Hence, the best solutions may be replaced by poorer solutions. This characteristic is not as obvious in Figure 7.3. In Figure 7.2 with the SFLP-II data set, the GA approach converges was than the GWO approach. This is due to how our approach replaces all solutions in the population for the next iteration. The size of the bounding region in SFLP-II may also be reason as it does not have

Problem	GWO				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	234.250481	339.099181	290.3857809	32.4373567833344	89.5666666666667
mSFLP-III	48951.787331	57804.257366	52702.9314929	2224.64491886288	197.733333333333
mKra30a	90455.74585	118315.534424	101874.328208933	7190.18569614101	341.266666666667

Table 7.3: Results obtained from our proposed GWO approach.

Problem	PSO				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	268.59568	380.688077	324.356679733333	38.4539160542525	31.3666666666667
mSFLP-III	60037.826591	68194.108383	64181.9165261333	1956.45733138936	66.9666666666667
mKra30a	106178.045845	130915.770554	118785.277772567	6138.39410532314	114.366666666667

Table 7.4: Results obtained from our proposed PSO approach.

a huge space for our approach to move buildings in various directions and prevent intersections as much as possible. This is the different from the mSFLP-III where the bounding region is larger, allowing buildings to less likely intersect with one another. Figure 7.3 showcases the behaviour of our GWO approach with a larger bounding region.

Despite the wins showcased by the competing GA approach, our proposed approach is faster when mSFLP-III is used with 185.7s for our approach, while the GA approach takes 260.56666667s. We also argue that the competing approach is already a hybridized approach with phases for a local search to be performed. Our approach can be considered to be a relatively pure adaptation of classical GWO, rather than a hybrid approach. Additionally, the results are not far from the results generated by the competing approach. They show that the simplicity of our approach (which only uses three parameters needed, compared to our competing approach's six) does not prevent our proposed algorithm from almost reaching competitiveness. They also

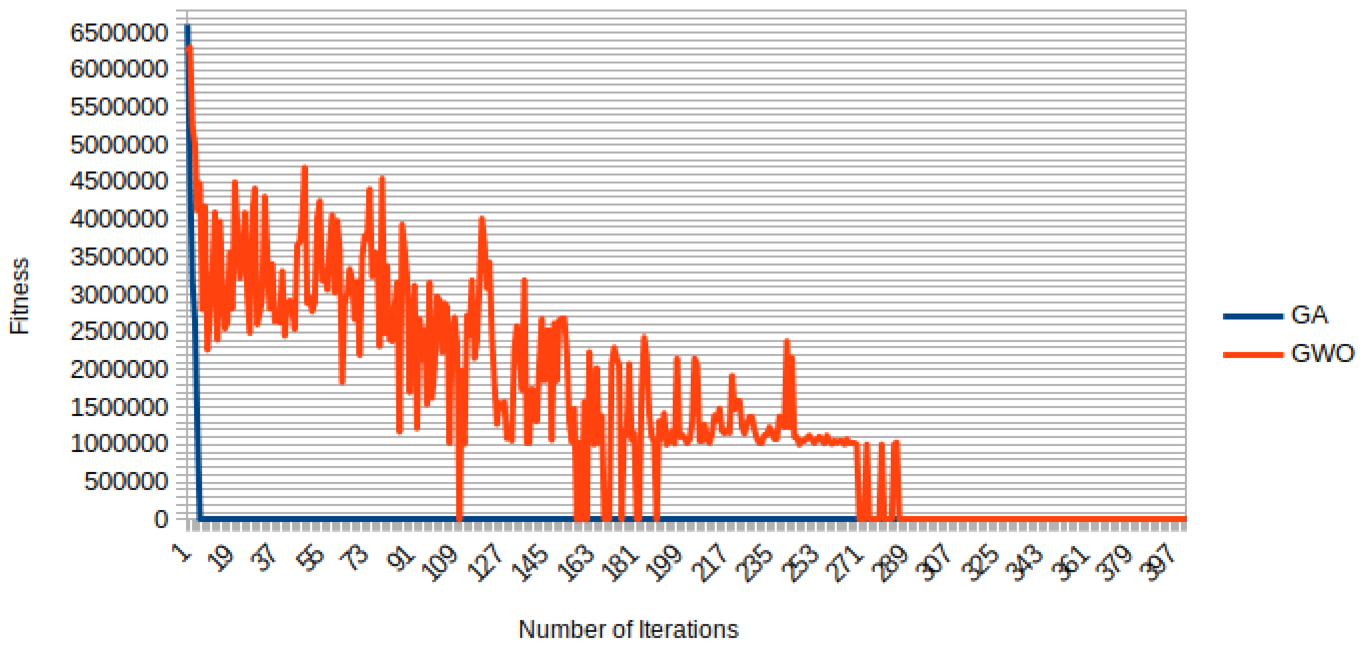

Figure 7.2: Fitness graph of the best solutions of the competing GA approach, and our GWO approach using the SFLP-II data set.

indicate that there is promise in further exploring the applicability of the grey wolf optimization algorithm in the facility layout problem.

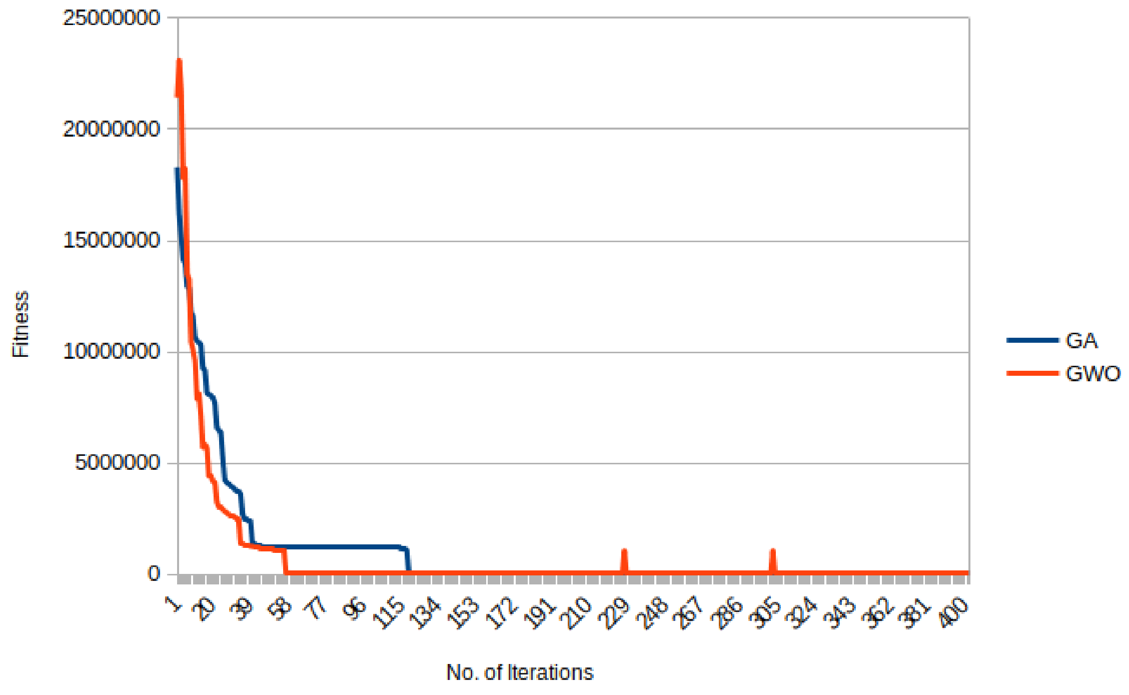

Figure 7.3: Fitness graph of the best solutions of the competing GA approach, and our GWO approach using the mSFLP-III data set.

Chapter 8

Conclusion and Summary

In this study, we proposed an alternative approach to the static unequal area facility layout problem, which was previously solved using genetic algorithm and particle swarm optimization. Our alternative approach utilizes the grey wolf optimization to solve the problem. We have introduced modifications to this metaheuristic in order for it to be able to produce feasible solutions. We have compared this modified GWO against a GA-based hybrid approach. Results from our experiment indicate that the GA-based approach is generally than our modified GWO. However, they showed that there is promise in GWO as an algorithm for solving FLPs. In one data set, our approach produced results faster than the GA-based approach, despite producing an average result that is slightly worse than the other approach. Our approach is also simpler, making it easier to understand and experiment with. In the future, our proposed modified GWO may be further improved to produce significantly better results. Besides, GWO is relatively new to the field, providing researchers with plentiful opportunities to improve the algorithm. Modifying the equations of our modified GWO, such as the decay rate of α, is one avenue in which researchers may take to build upon our study.

Bibliography

[1] Discover GAMS.

[2] Quadratic Assignment Problem — NEOS.

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

[4] tournament selection in genetic algorithm - Stack Overflow, 2015.

[5] André R.S. Amaral. On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2):508–518, 2006.

[6] 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.

[7] 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.

[8] 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.

[9] Mariem Besbes, Marc Zolghadri, Roberta Costa Affonso, Faouzi Masmoudi, and Mohamed Haddar. A methodology for solving facility layout problem considering barriers: genetic algorithm coupled with A* search. *Journal of Intelligent Manufacturing*, 31(3):615–640, 2020.

[10] F. Burkard, R.E., Cela, E., Karisch, S.E., Rendl. QAPLIB Problem Instances and Solutions, 2002.

[11] 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.

[12] Subham Datta. Branch and Bound Algorithm — Baeldung on Computer Science, 2020.

[13] 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.

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

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

[16] Gunter Dueck. New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel, 1993.

[17] Irina Dumitrescu and Thomas Stützle. Combinations of local search and exact algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2611:211–223, 2003.

[18] Matthias Ehrgott, Xavier Gandibleux, and Anthony Przybylski. Exact methods for multi-objective combinatorial optimisation. *International Series in Operations Research and Management Science*, 233:817–850, 2016.

[19] 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.

[20] Andries Engelbrecht. Particle swarm optimization: Velocity initialization. *2012 IEEE Congress on Evolutionary Computation, CEC 2012*, (2):10–15, 2012.

[21] 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.

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

[23] L. Garcia-Hernandez, L. Salas-Morera, C. Carmona-Munoz, A. Abraham, and S. Salcedo-Sanz. A Hybrid Coral Reefs Optimization-Variable Neighborhood Search Approach for the Unequal Area Facility Layout Problem. *IEEE Access*, 8(1):134042–134050, 2020.

[24] L. Garcia-Hernandez, L. Salas-Morera, J. A. Garcia-Hernandez, S. Salcedo-Sanz, and J. Valente de Oliveira. Applying the coral reefs optimization algorithm for solving unequal area facility layout problems. *Expert Systems with Applications*, 138:112819, 2019.

[25] 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.

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

[27] Fred Glover and Kenneth SÃČÂ¶rensen. Metaheuristics. *Scholarpedia*, 10(4):6532, 2015.

[28] 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.

[29] 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.

[30] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[31] Ranjan Kumar Hasda, Rajib Kumar Bhattacharjya, and Fouad Bennis. Modified genetic algorithms for solving facility layout problems. *International Journal on Interactive Design and Manufacturing*, 11(3):713–725, 2017.

[32] 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.

[33] 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.

[34] 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.

[35] Fariborz Jolai, Reza Tavakkoli-Moghaddam, and Mohammad Taghipour. A multi-objective particle swarm optimisation algorithm for unequal sized dynamic facility layout problem with pickup/drop-off locations. *International Journal of Production Research*, 50(15):4279–4293, 2012.

[36] L. Jourdan, M. Basseur, and E. G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, dec 2009.

[37] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, feb 2021.

[38] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[39] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, may 1983.

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

[41] K. K. Lai and Jimmy W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32(1):115–127, 1997.

[42] Young Hae Lee and Moon Hwan Lee. A shape-based block layout approach to facility layout problems using hybrid genetic algorithm. *Computers and Industrial Engineering*, 42(2-4):237–248, 2002.

[43] 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.

[44] 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.

[45] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[46] Alan R. McKendall and Jin Shang. Hybrid ant systems for the dynamic facility layout problem. *Computers and Operations Research*, 33(3):790–803, mar 2006.

[47] Alan R. McKendall, Jin Shang, and Saravanan Kuppusamy. Simulated annealing heuristics for the dynamic facility layout problem. *Computers and Operations Research*, 33(8):2431–2444, 2006.

[48] 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.

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

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

[51] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Handbook of Heuristics*, 1-2(1):759–787, 1997.

[52] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, feb 2016.

[53] Hasan Hosseini Nasab and Fatemeh Mobasheri. A simulated annealing heuristic for the facility location problem. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(3):210–224, 2013.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] Hani Pourvaziri and B. Naderi. A hybrid multi-population genetic algorithm for the dynamic facility layout problem. *Applied Soft Computing Journal*, 24:457–469, 2014.

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

[60] 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.

[61] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. A. Portilla-Figueras. The Coral Reefs Optimization Algorithm: A Novel Metaheuristic for Efficiently Solving Optimization Problems. *Scientific World Journal*, 2014, 2014.

[62] Bruno Seixas Gomes de Almeida and Victor Coppo Leite. Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems. *Swarm Intelligence - Recent Advances, New Perspectives and Applications*, pages 1–21, 2019.

[63] 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.

[64] Safiye Turgay. Multi objective simulated annealing approach for facility layout design. *International Journal of Mathematical, Engineering and Management Sciences*, 3(4):365–380, 2018.

[65] Md Sanuwar Uddin. Hybrid Genetic Algorithm and Variable Neighborhood Search for Dynamic Facility Layout Problem. *Open Journal of Optimization*, 04(04):156–167, 2015.

[66] Gerhard J. Woeginger. Exact algorithms for NP-hard problems: A survey, 2003.

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

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