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

September 2021

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.

ASST. 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

In 1623, an English poet by the name of John Donne has written an essay containing the widely quoted excerpt, ”No man is an island.”. These words have proven to be true throughout history and manifests itself in the human experience in general. This work of ours is obviously not an exception to this, and has benefited from the wisdom and inputs of different people from different walks of life.

Due gratitude and acknowledgments are dedicated to my thesis advisor, Prof. Victor M. Romero II, for his guidance, encouragements, and wisdom that enabled me to complete this thesis.

Special thanks to my friends in ”Mga Loyal sa Komsai”, Kenneth Lanante, Bea Santiago, Babes Ngoho, Aerol Nebril, and Cylwyn Creer for their comaraderie and general support in accomplishing this work, to Kate Young for semi-regularly asking me for the progress of my thesis, and to my friends in the Hangouts Int'l Discord server, notably, Shann Ripalda for providing inputs that helped me improve this work, and Denz Merin, Julian Yu, Julyanna Huang, Michael Omisol, and Elyzah Parcon for their *encouragements* and support. Special thanks to Rina Falculan and Charles Webb for helping me with practicing for the defense.

Lastly, I would like to thank God, my family, my godparent, Ramil Perez, and my friends for supporting and aiding me in accomplishing this thesis study.

Abstract

The unequal area static facility layout problem (UA-SFLP) deals with arranging a set of buildings of varying sizes in a region for a long period of time based on certain objectives. This problem is well-researched, with most researches solving instances of the problem, and the general facility layout problem (FLP), using traditional algorithms such as genetic algorithms, simulated annealing, and particle swarm optimization. However, newer algorithms have been introduced and may produce better solutions than previous studies. In this study, we are using the grey wolf optimization algorithm to solve the UA-SFLP. We have modified the algorithm in order for it to produce feasible solutions to the problem. We conducted experiments that vary the value of the c parameter and the population size. Our experiments show that a larger population size produces better results, and the proper c value is dependent on the population size and the problem being solved. We also compared our GWO approach against a hybrid GA approach and a PSO approach. We have discovered that the hybrid GA approach produces the best solutions on average but scales poorly when the number of buildings increase, with PSO producing the worst solutions on average but is the fastest. Our GWO approach is the second best on average in solution quality and speed, and was found to scale better than the hybrid GA approach. Hence, our approach provides a balance between speed and solution quality. Future studies can be done to improve the performance of GWO in solving FLPs.

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 [69]. 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]. Figure 1.1 provides an illustration of a facility layout problem. There are other types of facility layout problems, which we will discuss later. The illustration only showcases a continuous type of FLP.

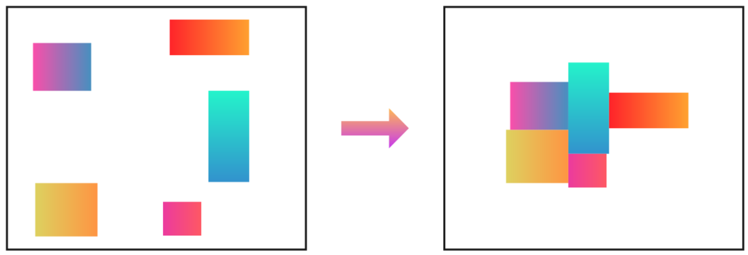


Figure 1.1: An illustration of the facility layout problem (FLP), which seeks to arrange a set of facilities or assets based on some criteria. Note that this illustration only showcases a continuous version of the facility layout problem. There are multiple types of facility layout problems.

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 [60], 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 [68]. Mixed integer programming, when in the context of optimization, is also known to be NP-Hard [60]. 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 [55]) 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 [61]. In works that derive from this formula, such as in [21], [64], and [56], 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 [58]. 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 [57]. 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) [14] 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.

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

Figure 1.2: 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.

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 [51] [29]. The algorithm works by letting the wolves gradually move towards a prey/local optimum solution. At the start, the wolves will be scattered and may likely be too far from a prey. The movement of the wolves, which will be guided by the alpha, beta, and delta wolves (which are closest to a prey/local optimum), will eventually allow them to be in an area where the prey/local optimum is located. As an interesting side note, the inspiration for Grey Wolf Optimization initially came from The Grey [50], 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 [51].

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 [51]. 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 [51]. Each wolf updates their positions based on the following equations.

$$\vec{D}_\alpha = \left|\left(\vec{C}_1 \cdot \vec{X}_\alpha\right) - \vec{X}\right| \quad (1.2.1)$$

$$\vec{D}_\beta = \left|\left(\vec{C}_2 \cdot \vec{X}_\beta\right) - \vec{X}\right| \quad (1.2.2)$$

$$\vec{D}_\delta = \left|\left(\vec{C}_3 \cdot \vec{X}_\delta\right) - \vec{X}\right| \quad (1.2.3)$$

$$\vec{X}_1' = \vec{X}_\alpha(t) - \vec{A}_\alpha \cdot \vec{D}_\alpha \quad (1.2.4)$$

$$\vec{X}_2' = \vec{X}_\beta(t) - \vec{A}_\beta \cdot \vec{D}_\beta \quad (1.2.5)$$

$$\vec{X}_3' = \vec{X}_\delta(t) - \vec{A}_\delta \cdot \vec{D}_\delta \quad (1.2.6)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1' + \vec{X}_2' + \vec{X}_3'}{3} \quad (1.2.7)$$

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.

1.2.2 Interpretation of GWO in the Abstract Search Space

In the perspective of an n-dimensional abstract search space, we can view preys as local optima (and the value of each prey can be seen as their nutritional value), and the wolves being positioned in some point in the search space. Assuming that we are dealing with the facility layout problem, we can view areas that have infeasible solutions as having higher elevations than those areas with feasible solutions (Note that since it is difficult to visualize and imagine higher-dimensional objects, we are using terms from a three-dimensional plane.). The problem can then be viewed as finding the lowest valley in the search space. Movement of the wolf would be seen in the search space as the traversal of the aforementioned terrain (though the wolves will not struggle in doing so no matter how, from our perspective, difficult the terrain features are to traverse). Wolves in a position where the area is in a high enough elevation would be considered to be having an infeasible solution. When viewed in the FLP space, buildings here may be intersecting with other buildings, among other infeasibility factors. Conversely, those wolves positioned in a low enough area would be considered to have feasible solutions. Buildings in this type of area are not necessarily arranged in a compact manner, but they, at the least, do not violate feasibility criteria. Additionally, given how the GWO is modelled, wolves would not be bothered nor punished when moving towards high elevation/infeasible areas despite coming from a lower/feasible area. This does allow the algorithm to move towards a local optimum faster. To aid understanding of how GWO behaves in the abstract search space, Figure 1.3 provides a visualization.

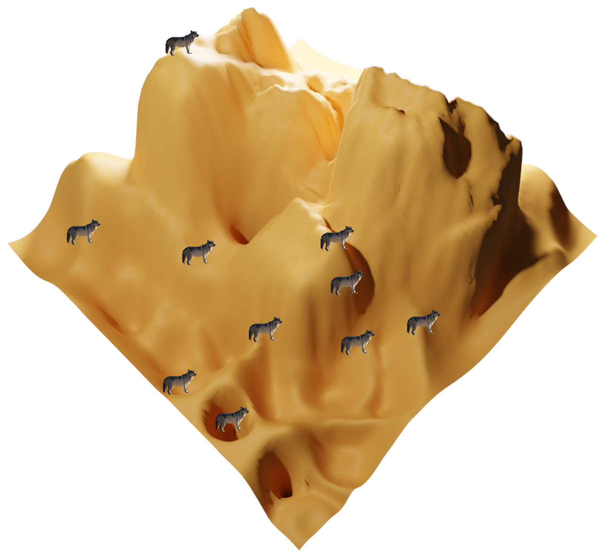


Figure 1.3: A visualization of how GWO behaves in an n-dimensional search space. Throughout the course of the execution of GWO, the wolves will be traversing the terrain (without much difficult, most probably unlike the first author of this manuscript). The topmost wolf has the worst solution, while the bottommost one has the best.

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 [67].

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 [53]. 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 [64].

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 [56]. 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 [52], 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 [66].

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 [59].

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 [65]. 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 [54]. 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 [69].

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 [63]. 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 [62]. 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 [51]. 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.

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

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:

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

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

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:

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

The aforementioned equations may be usable as is for other problems. However, as we have discovered through our prior 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 primarily to the formula, $\vec{D} = \left|\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)\right|$. To understand why the aforementioned formula contributes to the behaviour we have discussed earlier, we should understand what the formula means. It is helpful to

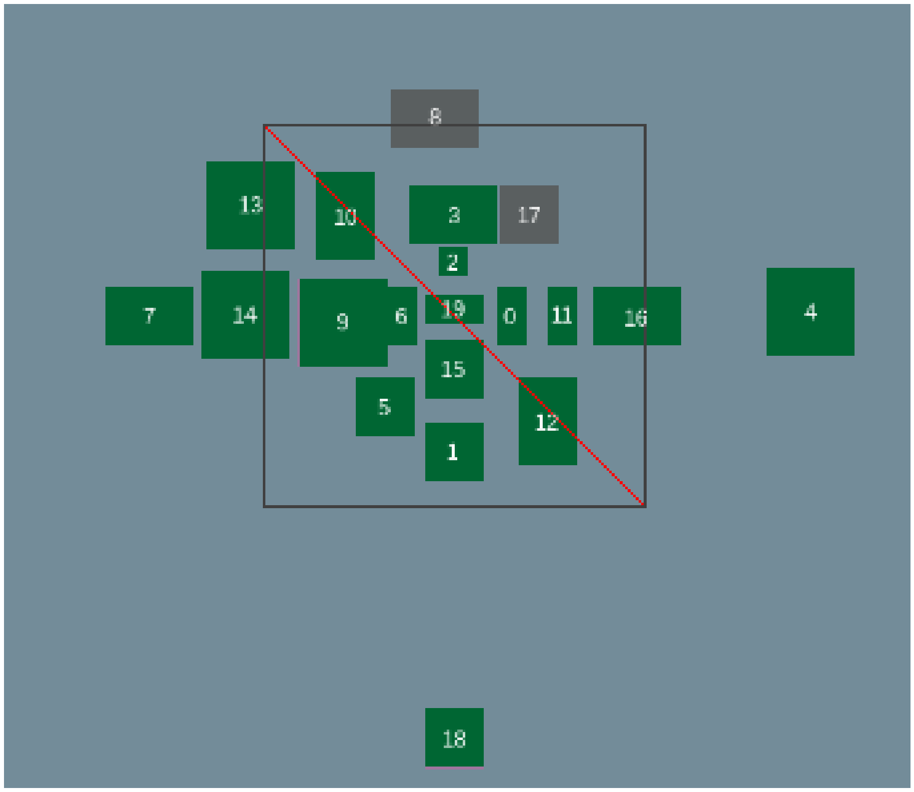


Figure 5.2: A visualization of a solution where the buildings tend to move towards the axes, with many already being restricted to them.

simply consider that only 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 a dot product, but it is actually 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.

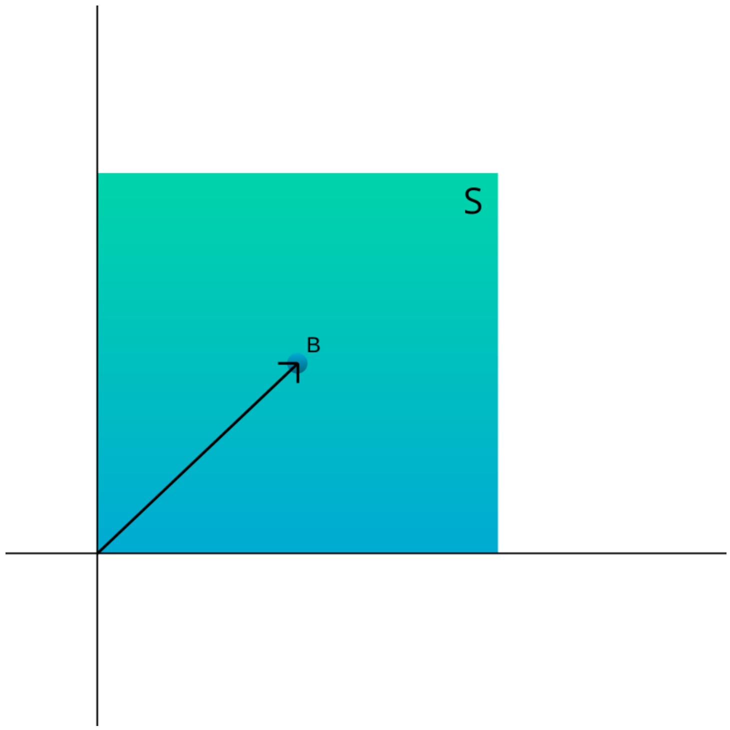


Figure 5.3: In K, C simply scales the x and y positions and angles of buildings. Assuming that the point B represents the x and y positions of a building, the region S is where B may be repositioned based on the values of C.

Let us also take note, $\vec{A} \cdot \vec{D}$. 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. Considering equations 5.3.1 to 5.3.3, $\vec{A}$ influences the distance of a building from its counterpart in the leading wolves. This would mean that as the number of iterations increase in a run, buildings will eventually follow the placements of the leading wolves.

Let us now return back to $\vec{C} \cdot \vec{X}_l(t)$. Over the course of iterations, this equation

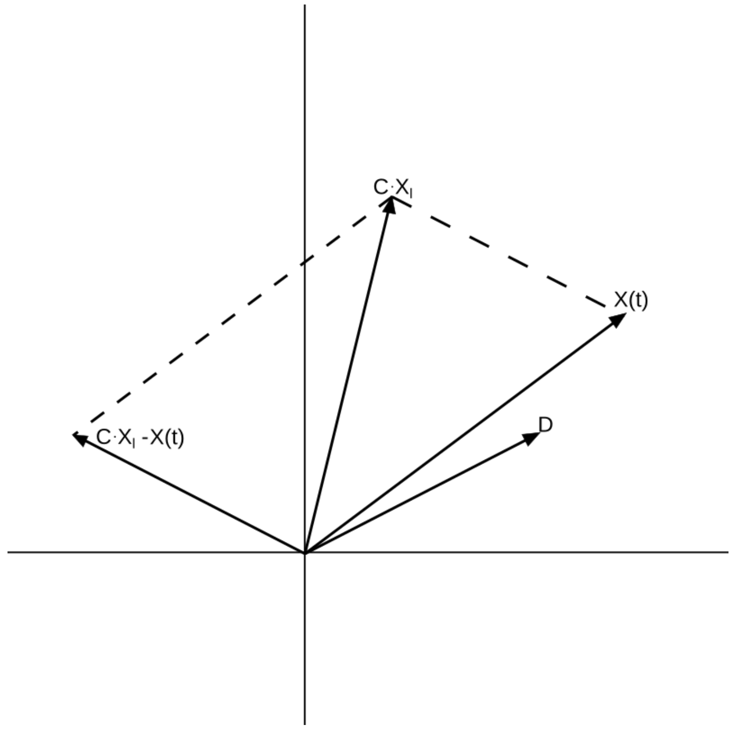


Figure 5.4: A visualization of how D is computed and its inherent meaning.

will make it difficult for a building to change its position. Around 50% of the time (due to the fact that $\vec{r_2}$ is a *uniform* random vector), the value of $\vec{C}$ will be less than 1. The position of the buildings will be moved towards the axes. Since $\vec{C}$ is a scaling factor, it will be difficult for a building position to move away from an axis. This affects all solutions, and noting that the leading wolves guide the entire population, the movement towards an axis will be propagated towards the entire population, especially with the fact that the $\vec{A}$ reduces the difference between the leading wolves/solutions and the rest of the solutions as the number of iterations increase in a run. 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, and to the scaling phenomenom caused by $\vec{C}$. 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 aforementioned formula makes the 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. [51] 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.

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.

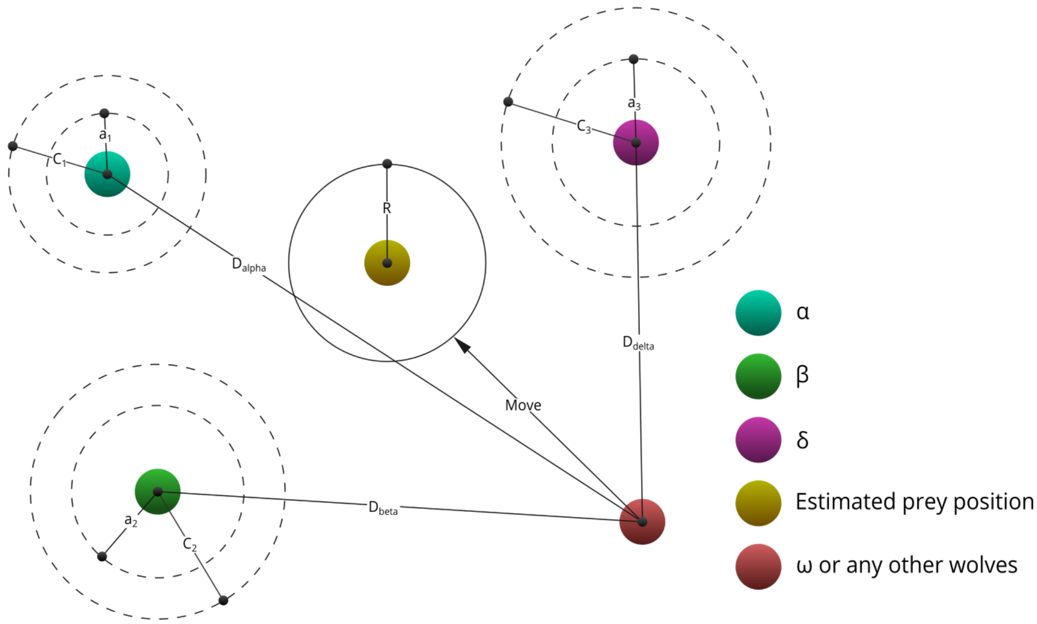


Figure 5.5: Visualization of how wolves in GWO update their positions. An ω wolf will move towards a random point inside the circle of the estimated prey position.

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.

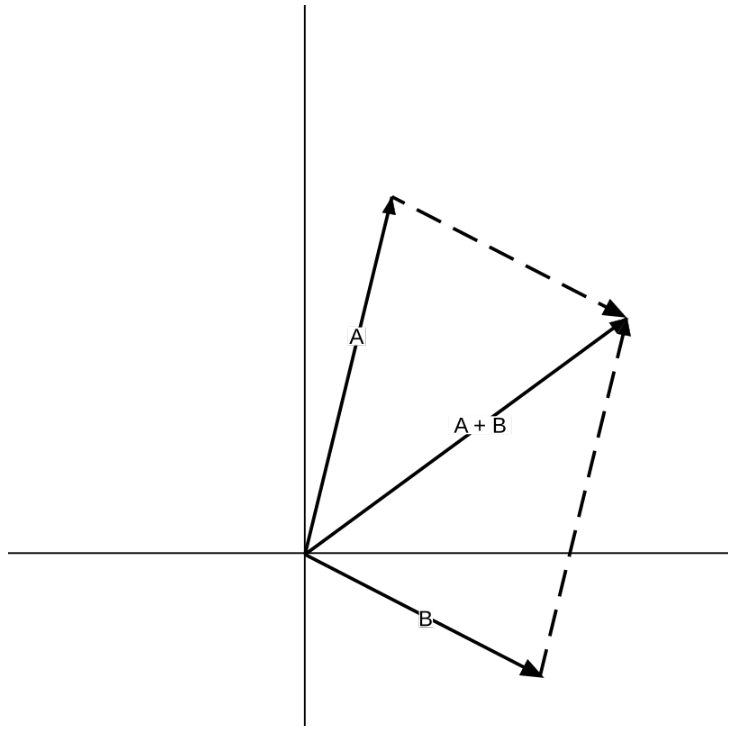


Figure 5.6: Vector addition pushes the point represented by $\vec{A}$ towards the direction of $\vec{B}$ by the magnitude of the same vector.

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.

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

We also have another small but still important modification to the classical GWO. In each iteration, the global best generated by our algorithm is tracked by our approach. The global best, however, does not replace the alpha wolf of any iteration, even if the alpha wolf has already become infeasible. The algorithm continues as though no tracking is being conducted in the first place.

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 problems.

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

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

$$\vec{D}_\alpha = \left| \left(\vec{C_1} + \vec{X_\alpha(t)} \right) - \vec{X(t)} \right| \tag{5.3.12}$$

$$\vec{D}_\beta = \left| \left(\vec{C_2} + \vec{X_\beta(t)} \right) - \vec{X(t)} \right| \tag{5.3.13}$$

$$\vec{D}_\delta = \left| \left(\vec{C_3} + \vec{X_\delta(t)} \right) - \vec{X(t)} \right| \tag{5.3.14}$$

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

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

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

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

Algorithm 2 Pseudocode for the proposed modified GWO.

Set T to be the maximum number of iterations.
Initialize the grey wolf population $\vec{X}_i (i = 1, 2, \ldots, n)$
Initialize c, and $a = 2$.
Calculate the fitness of each wolf.
Set $\vec{X}_\alpha$ to be the fittest wolf.
Set $\vec{X}_\beta$ to be the second fittest wolf.
Set $\vec{X}_\delta$ to be the third fittest wolf.
Set $\vec{X}_{best}$ to be the global best wolf.
while t < T **do**
    for each wolf $\vec{X}_i$ **do**
        Initialize $\vec{A}_1$, $\vec{A}_2$, $\vec{A}_3$, $\vec{C}_1$, $\vec{C}_2$, and $\vec{C}_3$.
        Update the position of the current wolf $\vec{X}_i$ using Equations 5.3.12 to 5.3.18.
        Set the orientation of each building using Equation 5.3.9.
        Clamp the buildings in $\vec{X}_i$ using Equations 5.3.7 and 5.3.8.
    end for
    Calculate the fitness of each wolf.
    Update $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$.
    if $\vec{X}_{best} ==$ null or $f(\vec{X}_\alpha) < f(\vec{X}_{best})$ **then**
        $\vec{X}_{best} = \vec{X}_\alpha$
    end if
    $a = 2 - \frac{2t}{T}$
    $t = t + 1$
end while
return $\vec{X}_{best}$

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 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 mKra30a, 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.

Let S be the collection of generated solutions.
Set S_{curr} be the current solution.
Add S_{curr} to S.
Set N_B be the maximum number of buildings.
for i = 0 until $N_B - 2$ **do**
    for j = i + 1 until $N_B - 1$ **do**
        Building i's orientation in S_{curr} is changed to the other orientation,
        $\hookrightarrow$ and the resulting new solution is added to S
        Building j's orientation in S_{curr} is changed to the other orientation,
        $\hookrightarrow$ and the resulting new solution is added to S
        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
        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.
        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.
        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.
        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.
    end for
end for
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.

Set $X^{(1)}$ to be the first best parent.
Set $X^{(2)}$ to be the second best parent.
Set P to be the population of solutions.
Set k to be the tournament size.
Set $f(X)$ to be the fitness function.
$X^{(1)} = \text{null}$
$X^{(2)} = \text{null}$
for $i = 1, 2, \ldots, k$ **do**
    $p = U(1, |P|)$
    if $X^{(1)} == \text{null}$ or $f(P_p) < f(X^{(1)}$ **then**
        $X_{(2)} = X^{(1)}$
        $X_{(1)} = P_p$
    else if $X^{(2)} == \text{null}$ or $f(P_p) < f(X^{(2)}$ **then**
        $X_{(2)} = P_p$
    end if
end for
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.

Set X to be the offspring.
Set N to be the number of genes a solution has.
Set $X^{(1)}$ to be the first parent.
Set $X^{(2)}$ to be the second parent.
for $i = 1, 2, \ldots, N$ **do**
  rand $= U(0, 1)$
  if rand < 0.5 **then**
    $X_i = X_i^{(1)}$
  else
    $X_i = X_i^{(2)}$
  end if
end for
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 [61]. 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.

Set S to be a collection of solutions.
Set S_{curr} to be the solution being optimized.
Add S_{curr} to S.
Set N_B be the maximum number of buildings.
Set N_A be the maximum number of activities.
for i = 0 until $N_B - 1$ **do**
    for a = 0 until $N_B - 1$ **do**
        Perform activity a with building i in S_{curr} and save the new solution in S.
        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.
    end for
end for
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.

Set S to be a collection of solutions.
Set S_{curr} to be the solution being optimized.
Add S_{curr} to S.
Set N_B be the maximum number of buildings.
Set N_A be the maximum number of activities.
for i = 0 until $N_B - 2$ **do**
  for a = 0 until $N_B - 1$ **do**
    Perform activity a with building i in S_{curr} and save the new solution in S.
    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.
    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.
    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.
  end for
end for
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.

Initialize population P.
Calculate the fitness of each solution in P.
Set T to be the number of generations.
Set $X^{(best)}$ to be the best solution found.
Set E_N to be the number of elite solutions that will be kept in the next generation.
Sort P from lowest to highest fitness value.
for $t = 1, 2, \ldots T$ **do**
    if $t \leq 100$ **then**
        Apply swapping method.
    end if
    for $i = E_N + 1, \ldots, |P|$ **do**
        Select parents $X^{(1)}$ and $X^{(2)}$ using tournament selection.
        Crossover parents and produce offspring $X^{(o)}$.
        Mutate $X^{(o)}$ if mutation probability allows.
        $P_i = X^{(o)}$.
    end for
    Sort P from lowest to highest fitness value.
    Apply Local Search 1 to the best solution in P.
    if $t \geq T - 50$ **then**
        Apply Local Search 2 to the best solution in P.
    end if
end for
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.

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + c_1\vec{r}_1^t\left(p\vec{best}_i - \vec{X}_i^t\right) + c_2\vec{r_2}^t\left(g\vec{best}_i - \vec{X}_i^t\right) \tag{6.2.2}$$

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

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.

Set T to be the maximum number of iterations.
Initialize w, c_1, and c_2.
Initialize the swarm $\vec{X}_i (i = 1, 2, \ldots, n)$
Set the velocity $\vec{V}_i^t$ of each particle i to 0.
Calculate the fitness of each particle.
Set the current solution of each particle i as their personal best pbest_i.
Set the gbest to be the best solution in the swarm.
while t < T **do**
  for each particle $\vec{X}_i$ **do**
    Initialize $\vec{r}_1$ and $\vec{r}_1$.
    Update the position of the current particle $\vec{X}_i$ using Equations 6.2.2 and 6.2.3.
    Set the orientation of each building in $\vec{X}_i$ using Equation 6.2.4.
    Clamp the buildings in particle i using Equations 5.3.7 and 5.3.8.
    Calculate the fitness of $\vec{X}_i$.
    if $f(\vec{X}_i) < f(\text{pbest}_i)$ **then**
      $\text{pbest}_i = \vec{X}_i$
    end if
    if $f(\vec{X}_i) < f(\text{gbest})$ **then**
      $\text{gbest} = \vec{X}_i$
    end if
  end for
  $t = t + 1$
end while
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 mKra30a. 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**: AMD Ryzen 5 5600X
  - **GPU**: NVIDIA GeForce GTX 1050 Ti
  - **RAM**: Crucial Ballistix RGB 3600 MHz DDR4 16 GB (8 GB x 2) CL16
- Software
  - **OS**: elementaryOS 5.1.7 Hera
  - **Linux Kernel Version**: 5.4.0-99-generic

7.2 Experiments

We have conducted two sets of experiments in order for us to evaluate the performance of our proposed GWO approach. The first set of experiments varies the GWO parameter, c, and the population size. It shows the impact of the parameters to the algorithm. The second set are experiments with the competing approaches. It shows how our proposed approach compares against other approaches. A population size of 50 is used for all the experiment runs in the second set. This set will then be compared to the GWO experiment runs that have a population of the same size.

7.2.1 Results with Different GWO Parameter Values

Our proposed GWO approach only has one parameter, other than the population size, and number of iterations, the c value. We used four values for the parameter: 2, 4, 8, and 12. We also varied the population size for this experiment set. The population sizes we used are: 25, 50, and 75. Tables 7.1 to 7.12 show the results. We will refer to each GWO experiment as $G_{n,c}$, where n is the population size, and c is the value of the c parameter in each experiment. So, for example, the GWO experiment with a population size of 25 and $c = 2$ will be referred to as $G_{25,2}$, and so on. Each experiment for each parameter configuration combination has been run 30 times.

Let us first discuss the results with the SFLP-II problem. Table 7.13 provides a summary of the results of the experiments performed for the problem using the GWO approach. Figure 7.2, on the other hand, shows a line graph that displays the average fitness of solutions of each population size when solving the SFLP-II problem as the c value increases. For the problem, $G_{50,2}$ has the best average compared to the

Problem	GWO (c = 2, Pop. Size of 25)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	226.149871	328.546146	287.749326366667	24.7018482581174	6.03333333333333
mSFLP-III	50874.238564	60974.998642	55624.0061857667	2544.70550235336	22.4333333333333
mKra30a	94640.759514	120397.403767	107874.742523333	6706.6049593287	35.9666666666667

Table 7.1: Results obtained from our proposed GWO approach with $c = 2$ and a population of 25.

Problem	GWO (c = 4, Pop. Size of 25)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	228.710226	373.604858	298.836421533333	34.1522620177737	5.9
mSFLP-III	50825.278824	59194.486832	55236.4286594	2301.71070299619	21.3333333333333
mKra30a	87110.57618	116746.599121	104270.487286567	8041.05756072353	36.3333333333333

Table 7.2: Results obtained from our proposed GWO approach with $c = 4$ and a population of 25.

other configurations with a value of 283.795019233333. The worst average belonged to $G_{50,12}$ with a value of 315.831642166667. The configuration with the best solution produced would be $G_{75,2}$ with the solution having a value of 205.666955. Figure 7.3 shows the progress of the solution as the number of iterations increase. On the other hand, the worst solution was produced by $G_{75,12}$ with a value of 413.874466. Figure 7.1 shows what these solutions look like. The average runtime of the experiments increase as the population size increases. In a similar fashion, the experiments with population sizes of 25 and 50, their average fitness value worsens (increases) as the value of c increases. However, with a population size of 75, the same trend is reflected until $c = 12$, where the average fitness improves.

As with the mSFLP-III problem, of which Table 7.14 provides the summary of the experimental results, the best average was produced by $G_{75,8}$ with a value of 51801.8837937333. $G_{25,2}$ produced the worst average with a value of 55624.0061857667.

Problem	GWO (c = 8, Pop. Size of 25)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	249.624192	368.435807	313.640670633333	29.7958232730557	6.23333333333333
mSFLP-III	51250.638187	57894.186882	54206.6467008333	1334.40810225102	23.6
mKra30a	95254.061554	118719.490257	105760.743434367	6557.69877516131	37.0666666666667

Table 7.3: Results obtained from our proposed GWO approach with $c = 8$ and a population of 25.

Problem	GWO (c = 12, Pop. Size of 25)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	255.639347	358.033844	315.207093933333	26.4399229476443	6.36666666666667
mSFLP-III	51328.437737	62758.004044	55331.2312675333	2540.54370041386	21.2333333333333
mKra30a	93525.765816	124181.531029	108251.9895637	8151.19724950765	37.3

Table 7.4: Results obtained from our proposed GWO approach with $c = 12$ and a population of 25.

For this problem, the best solution produced has a fitness of 47597.794662, and is generated by $G_{50,2}$. Figure 7.6 shows the progress of this solution as the number of iterations increase. Interestingly, $G_{50,2}$ has also generated the worst solution with a fitness value of 62827.738159. These solutions are visualized by Figure 7.4. Parallel to the observed behaviour with the SFLP-II problem, the average runtime of the experiments also increases as the size of the population increases. A trend is also observable as the c value increase that applies to all population sizes used. Increasing the c value shows an improvement in the fitness value (value decreases). Unfortunately, this behaviour changes when $c = 12$, where the fitness worsens. To supplement Table 7.14, we are also providing Figure 7.5, which presents a line graph version of the results displaying the relationship between the c value and the average fitness of the experiments solving the problem in every population size we are using.

Lastly, we present a brief overview of the results we obtained for the mKra30a

Problem	GWO (c = 2, Pop. Size of 50)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	221.18019	341.568304	283.795019233333	28.9742518867792	13.2666666666667
mSFLP-III	47597.794662	62827.738159	54495.2676476667	3328.18744058766	40.1333333333333
mKra30a	88657.824898	121124.59779	102742.1803823	7156.18271087496	73.3333333333333

Table 7.5: Results obtained from our proposed GWO approach with $c = 2$ and a population of 50.

Problem	GWO (c = 4, Pop. Size of 50)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	241.862298	389.711568	294.2461242	33.7641257773172	13.1333333333333
mSFLP-III	50250.080536	57916.673454	53421.9267731333	2239.06725435468	41.9666666666667
mKra30a	90599.06601	122300.269909	102855.4831497	8820.10238434929	70.9666666666667

Table 7.6: Results obtained from our proposed GWO approach with $c = 4$ and a population of 50.

problem. Table 7.15 shows the summary of the experimental results for the problem, and Figure 7.8 provides a graph version of the results showcasing the impact of the value of c on the average fitness of the experiments solving the problem for each population size we are using. The best average for this problem was produced by $G_{75,12}$ with a value of 98108.9092933667. On the contrary, the worst average was produced by $G_{25,12}$ with a value of 108251.9895637. The best solution has a fitness value of 84929.672058, and is generated by $G_{50,8}$. Figure 7.9 shows its progress over iterations as it solves the mKra30a problem. The worse solution, on the other hand, with a value of 128598.716599, was produced by $G_{50,12}$. These solutions are visualized by Figure 7.7. In the same vein as the two aforementioned problems, the average runtime of the experiments increase as the population size increased. As for the values of the average fitness values with respect to the population size and c values, no common trend can be observed for the three different population sizes, unlike

Problem	GWO (c = 8, Pop. Size of 50)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	240.638127	413.077936	299.553292466667	40.469222577263	12.7333333333333
mSFLP-III	48844.175789	59710.615997	52699.5983075667	2062.76885562279	41.2666666666667
mKra30a	84929.672058	112251.415863	101570.163644533	6490.61277032704	72.1

Table 7.7: Results obtained from our proposed GWO approach with $c = 8$ and a population of 50.

Problem	GWO (c = 12, Pop. Size of 50)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	261.869799	381.586061	315.831642166667	31.7204847308938	13.6666666666667
mSFLP-III	48920.979538	56477.689476	52837.3591700333	1980.22102755171	42.2333333333333
mKra30a	92563.720146	128598.716599	105348.4949903	9267.72959691125	69.5666666666667

Table 7.8: Results obtained from our proposed GWO approach with $c = 12$ and a population of 50.

with the previous problems. Problems with population sizes of 50 and 75 share a common trend from $c = 2$ to $c = 8$, where the increasing the c value first worsens the average fitness, but the fitness then improves. However, when increasing the c value from 8 to 12, the behaviour differs. With a population size of 50, the average fitness worsens. On the other hand, with a population size of 75, the average fitness improves instead. Lastly, the configuration with a population size of 25 acts differently from the configurations with the aforementioned population sizes. With a population size of 25, the average fitness value when increasing the c value from 2 to 4 initially shows an improvement of the average fitness. However, increasing the c value further results in worsening average fitness values.

Each data set used in the experiments uses differently-sized bounding regions. For SFLP-II, a $12x12$ bounding region is used. mSFLP-III uses a $260x260$ bounding region, while mKra30a uses a $250x250$ one. Notice that, for the SFLP-II problem,

Problem	GWO (c = 2, Pop. Size of 75)				
	Best	Worst	Avg.	Std. Dev.	Avg. Runtime (s)
SFLP-II	205.666955	386.356476	290.063388433333	38.0436802516604	18.8666666666667
mSFLP-III	50179.684898	57659.66436	54015.3749653	2050.7167136713	61.1
mKra30a	88740.484344	117173.305939	99149.0616948	5833.24082935413	104.7

Table 7.9: Results obtained from our proposed GWO approach with $c = 2$ and a population of 75.

Problem	GWO (c = 4, Pop. Size of 75)				
	Best	Worst	Avg.	Std. Dev.	Avg. Runtime (s)
SFLP-II	239.258536	406.790997	302.005947566667	39.1344289013742	18.9
mSFLP-III	48752.443314	56156.624268	52037.6629276	2313.4004317195	61.5333333333333
mKra30a	89197.608078	121878.61042	99482.2352776	7069.6968084522	107.966666666667

Table 7.10: Results obtained from our proposed GWO approach with $c = 4$ and a population of 75.

configurations using $c = 2$ in each population size produce the best average fitness. For the mSFLP-III problems, regardless of population size, using $c = 8$ produces the best average fitness. This suggests to us that, for certain problems with at least less than 20 buildings, the ideal c value to be used with our GWO approach have some correlation with the size of the bounding box. Smaller c values are more likely to be better for problems with smaller bounding boxes. Similarly, larger c values are more likely to better fit problems with larger bounding boxes. However, too high of a value for c may produce worse results on average. This is shown to us by our results with the mSFLP-III problem, where, regardless of population size, configurations with $c = 8$ consistently perform better on average than those with $c = 12$. We can attribute this behaviour to the fact that the c parameter determines how much a building can be shifted away in the formulas of D_α, D_β, and D_δ (see equations 5.3.10 to 5.3.18 in Methodology). A smaller c value introduces a smaller shift, while a larger value

Problem	GWO (c = 8, Pop. Size of 75)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	246.916627	393.744452	309.957982766667	32.0156308267039	19.5
mSFLP-III	49276.248596	54977.558044	51801.8837937333	1419.1918023338	63.0333333333333
mKra30a	87299.715054	113760.078079	98968.6223121	6443.60722715266	108.7

Table 7.11: Results obtained from our proposed GWO approach with $c = 8$ and a population of 75.

Problem	GWO (c = 12, Pop. Size of 75)				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	243.386427	413.874466	307.3213231	36.2239957721235	19.3333333333333
mSFLP-III	49644.232903	55524.684891	51837.8766529667	1430.57988385005	61.7666666666667
mKra30a	86942.304199	108422.175175	98108.9092933667	6511.43059062118	111.266666666667

Table 7.12: Results obtained from our proposed GWO approach with $c = 12$ and a population of 75.

will shift the buildings further. In smaller sized bounding regions, a smaller shift is important due to the limited space available. Larger shifts in such a space will make it harder for buildings to reach feasibility. On the other hand, a larger shift is more appropriate in a larger space since it will allow buildings to move closer to each other faster. Moreover, such a larger amount of available space can be better utilized. A larger space will allow buildings to more easily move away from intersections (and, consequently, infeasibility). However, as shown by our experiments with the mSFLP-III problem, a configuration with the ability to shift too much (e.g. having a c value of 12) can perform poorer than one with a slightly lower amount of shifting. This is due to the fact that shifts that are too large can push and orient buildings in positions and orientations where, over time, it would become more difficult for them to move towards a position where they are close to other buildings as much as possible yet not intersecting with any of them. Buildings may actually move further away from

Parameters		**SFLP-II**				
Pop. Size	**c**	**Best**	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
25	2	226.149871	328.546146	287.749326366667	24.7018482581174	6.03333333333333
	4	228.710226	373.604858	298.836421533333	34.1522620177737	5.9
	8	249.624192	368.435807	313.640670633333	29.7958232730557	6.23333333333333
	12	255.639347	358.033844	315.207093933333	26.4399229476443	6.36666666666667
50	2	221.18019	341.568304	283.795019233333	28.9742518867792	13.2666666666667
	4	241.862298	389.711568	294.2461242	33.7641257773172	13.1333333333333
	8	240.638127	413.077936	299.553292466667	40.469222577263	12.7333333333333
	12	261.869799	381.586061	315.831642166667	31.7204847308938	13.6666666666667
75	2	205.666955	386.356476	290.063388433333	38.0436802516604	18.8666666666667
	4	239.258536	406.790997	302.005947566667	39.1344289013742	18.9
	8	246.916627	393.744452	309.957982766667	32.0156308267039	19.5
	12	243.386427	413.874466	307.3213231	36.2239957721235	19.3333333333333

Table 7.13: Summary of the experiments using the GWO approach with the SFLP-II problem.

Parameters		mSFLP-III				
Pop. Size	**c**	**Best**	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
25	2	50874.238564	60974.998642	55624.0061857667	2544.7055023536	22.4333333333333
	4	50825.278824	59194.486832	55236.4286594	2301.71070299619	21.3333333333333
	8	51250.638187	57894.186882	54206.6467008333	1334.40810225102	23.6
	12	51328.437737	62758.004044	55331.2312675333	2540.54370041386	21.2333333333333
50	2	47597.794662	62827.738159	54495.2676476667	3328.18744058766	40.1333333333333
	4	50250.080536	57916.673454	53421.9267731333	2239.06725435468	41.9666666666667
	8	48844.175789	59710.615997	52699.5983075667	2062.76885562279	41.2666666666667
	12	48920.979538	56477.689476	52837.3591700333	1980.22102755171	42.2333333333333
75	2	50179.684898	57659.66436	54015.3749653	2050.7167136713	61.1
	4	48752.443314	56156.624268	52037.6629276	2313.4004317195	61.5333333333333
	8	49276.248596	54977.558044	51801.8837937333	1419.1918023338	63.0333333333333
	12	49644.232903	55524.684891	51837.8766529667	1430.5798385005	61.7666666666667

Table 7.14: Summary of the experiments using the GWO approach with the mSFLP-III problem.

Parameters		mKra30a				
Pop. Size	**c**	**Best**	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
25	2	94640.759514	120397.403767	107874.742523333	6706.6049593287	35.9666666666667
	4	87110.57618	116746.599121	104270.487286567	8041.05756072353	36.3333333333333
	8	95254.061554	118719.490257	105760.743434367	6557.69877516131	37.0666666666667
	12	93525.765816	124181.531029	108251.9895637	8151.19724950765	37.3
50	2	88657.824898	121124.59779	102742.1803823	7156.18271087496	73.3333333333333
	4	90599.06601	122300.269909	102855.4831497	8820.10238434929	70.9666666666667
	8	84929.672058	112251.415863	101570.163644533	6490.61277032704	72.1
	12	92563.720146	128598.716599	105348.4949903	9267.72959691125	69.5666666666667
75	2	88740.484344	117173.305939	99149.0616948	5833.24082935413	104.7
	4	89197.608078	121878.61042	99482.2352776	7069.6968084522	107.966666666667
	8	87299.715054	113760.078079	98968.6223121	6443.60722715266	108.7
	12	86942.304199	108422.175175	98108.9092933667	6511.43059062118	111.266666666667

Table 7.15: Summary of the experiments using the GWO approach with the mKra30a problem.

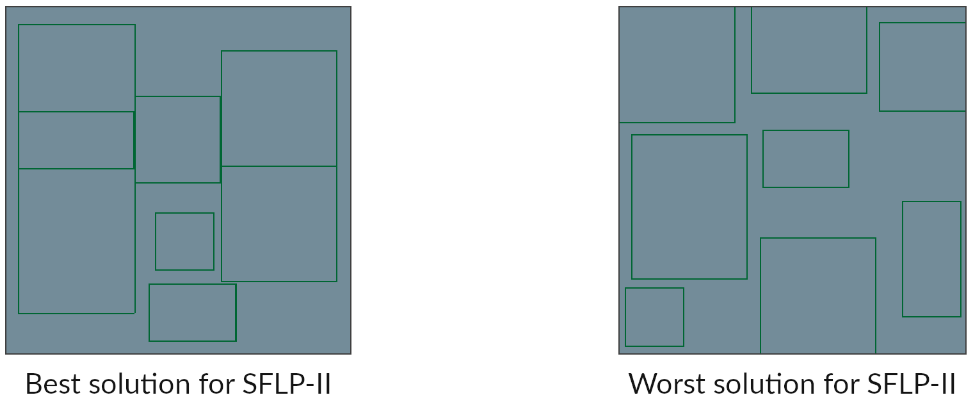


Figure 7.1: Visualizations of the best and worst solutions generated by our GWO approach for the SFLP-II problem. The best solution was generated by $G_{75,2}$, and the worst generated by $G_{75,12}$.

or move in such a way that hinders them from progressing towards these better positions due to this amount of shifting, contributing to the difficulty. Other causes further exacerbate this behaviour. The gradual reduction of the amount of shifting buildings can do as time progresses (see Equation 5.3.10, which is the factor for this gradual reduction), makes it harder for buildings to shift towards a superior position and influencing them to only move within a gradually smaller local area. Another contributing factor to the behaviour is that buildings cluster together over time in our approach. This increases the risk of building intersections, especially those building that are nearer to the ”inside” of a cluster.

Intriguingly, for the mKra30a problem (which has 30 buildings), the ideal c value

Figure 7.2: The average fitness of the solutions produced by our GWO approach as the c value increases when solving the SFLP-II problem. Each line uses a different population with the blue line representing experiments using a population size (n) of 25, the yellow lines representing those with $n = 50$, and the green lines representing those with $n = 75$.

seems to have a correlation with the population size used. As the population size increases, the c value must also be increased to be able to produce the best solutions possible. This is in contrary with the trend suggested by the experiments dealing with the first two problems. This may suggest that the parameters of a problem are also factors in determining the ideal value for c. However, additional experiments are needed to determine if this behaviour with the mKra30a problem is not a merely quirk caused by the random nature of metaheuristics. If ever it is found out to be an expected behaviour of our GWO approach when dealing with the mKra30a problem or a problem of similar or greater parameters, additional experiments should be able to provide insights as to why our approach has this behaviour.

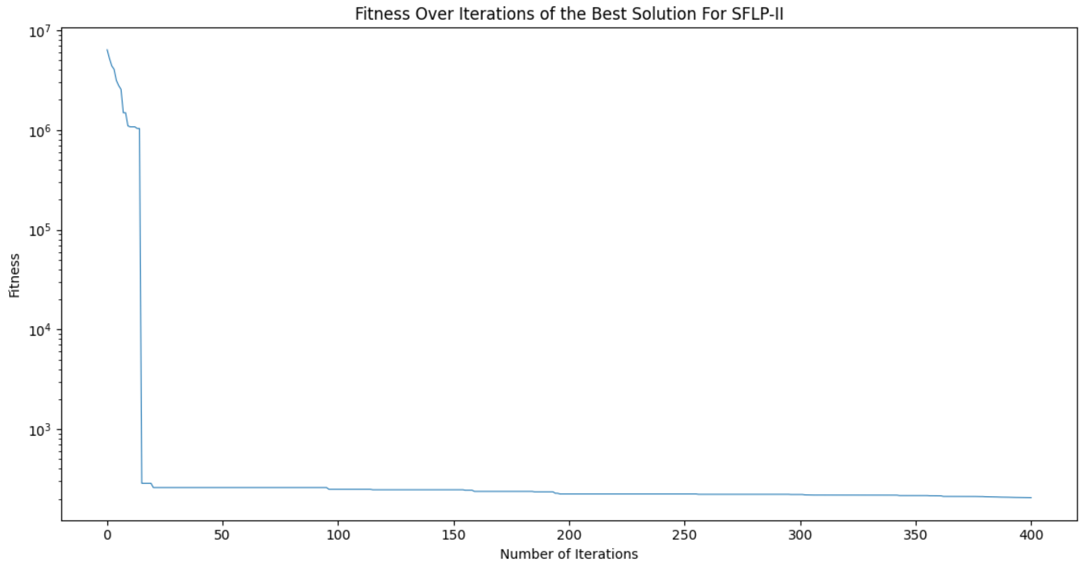
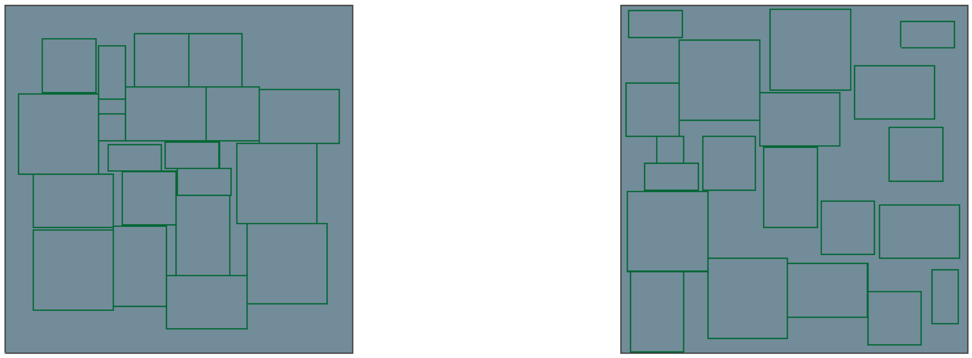


Figure 7.3: The fitness of the best solution generated by $G_{75,2}$ as the number of iterations increase while solving the SFLP-II problem.

It should also be mentioned that the average runtime of each experiment setup in each population size category are generally almost equal to each other. However, the experiments dealing with the mKra30a problem with population sizes of 50 and 75 have larger deviations from each other. It is of interesting note that for the experiment configurations with a population size of 75 solving the mKra30a problem, the average runtime increases as the c value increases.

We also observe that the population size has an impact on the performance of our GWO approach. For larger-sized problems with large bounding boxes such as mSFLP-III and mKra30a, our experiments show that, on average, a larger population size is more ideal. We argue that this is due to the larger amount of space the wolves/solutions cover in the abstract search space, which becomes larger as the size of the problem increases. A diverse set of initial solutions resulting from the larger



Best solution for mSFLP-III

Worst solution for mSFLP-III

Figure 7.4: Visualizations of the best and worst solutions generated by our GWO approach for the mSFLP-III problem. The best and worst solutions were generated by $G_{50,2}$.

population size allows for this larger amount of covered space. This naturally allows us to more easily find the best possible solution for a problem within a reasonable amount of time compared to with a smaller population size. On the contrary and quite interestingly, for small-sized problems with smaller-sized bounding boxes, such as SFLP-II, our experiments show that the higher population sizes do not necessarily translate to better solutions on average. It is suggested by our experiments that a medium-sized population with the right c value produces the best possible solutions for these small-sized problems compared to using the other two population sizes, with small-sized populations performing surprisingly better than large-sized ones, again with both using the best c value for the population size. This is rather odd given

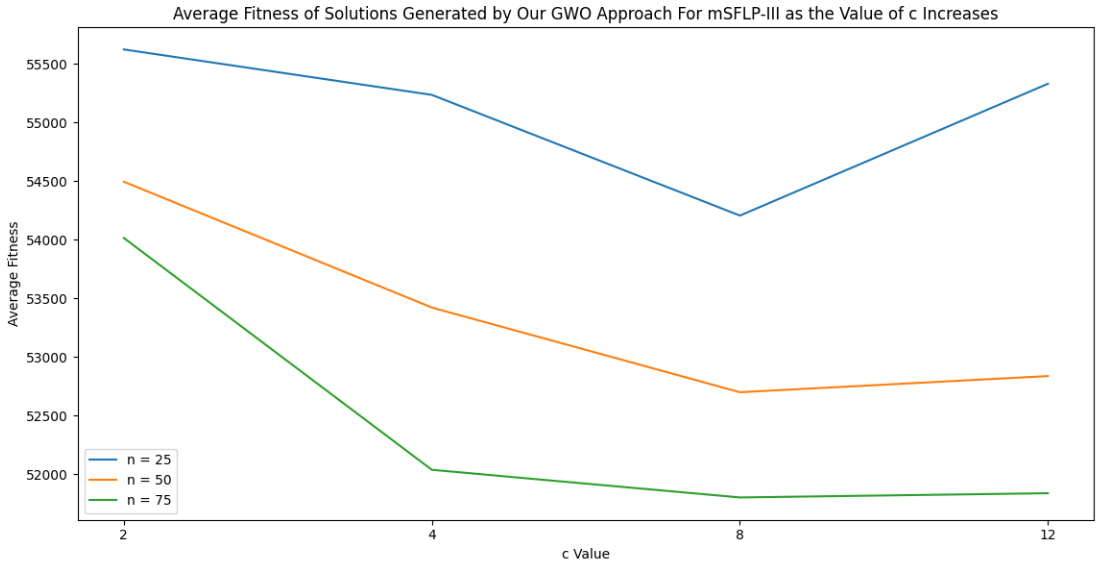


Figure 7.5: The average fitness of the solutions produced by our GWO approach as the c value increases when solving the mSFLP-III problem. Each line uses a different population with the blue line representing experiments using a population size (n) of 25, the yellow lines representing those with $n = 50$, and the green lines representing those with $n = 75$.

the advantage of having a larger population. It would make sense for experiments with medium-sized populations and the right c value to perform than their small-sized population counterparts. However, it does not immediately make sense why the large-sized population experiments would perform poorly than the small-sized ones. In any case, they should perform better, especially when against the medium-sized population. A possible explanation to this is that a larger population size for small-sized problems may result in achieving local optimum too fast. As per the behaviour of GWO itself, wolves/solutions would gradually cluster around this local optimum until a new better local optimum is found to cluster towards to. We argue that finding another local optimum would be more difficult, since SFLP-II has smaller bounding

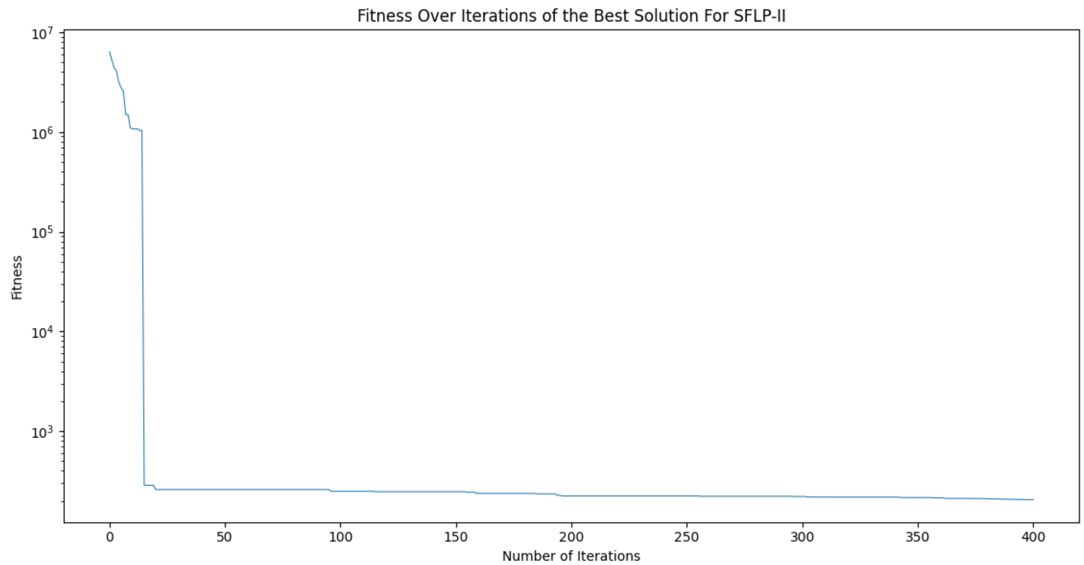


Figure 7.6: The fitness of the best solution generated by $G_{50,2}$ as the number of iterations increasee while solving the mSFLP-III problem.

region size. Intersections between buildings will occur more frequently compared to when using a larger bounding region. Hence, our approach finds it harder to find a better local optimum, resulting in worse average results. Further studies would be needed to gain a better understanding of this behaviour, to confirm our hypothesis, and to determine whether this is simply a quirk of randomness or not.

Later research may also want to focus on determining the ideal c value and population size for a certain problem based on the problem parameters. Figuring out whether the c value can be mathematically modelled rather than being a parameter is another possible avenue for research.

Run	GWO (c = 2, Pop. Size of 25)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	284.274332	5	54517.212997	20	113444.272751	34
2	279.224484	6	55100.110451	25	108634.12896	35
3	290.766824	7	54424.891159	20	101057.257011	36
4	255.89164	6	52281.025963	21	106933.341133	36
5	303.41135	6	56333.281975	24	110198.061668	37
6	307.247426	6	56191.257446	23	103150.565979	35
7	297.32725	6	57228.114368	26	100159.379417	34
8	298.901143	6	54501.928665	25	94640.759514	36
9	290.924892	6	60974.998642	22	101369.649643	35
10	328.546146	5	52108.460861	23	114400.855766	34
11	310.501635	6	55957.510098	21	108862.321945	39
12	260.87289	5	55527.02594	24	106595.717178	39
13	282.531157	5	58757.132118	20	115065.923737	34
14	260.606181	7	60825.496319	22	109135.383606	35
15	315.86567	6	52600.283459	20	120397.403767	35
16	277.906986	7	59222.901581	24	114673.857002	39
17	313.385026	6	53981.477821	24	96989.450607	36
18	289.163818	6	57612.964134	23	104600.307762	37
19	275.85824	5	58171.204727	20	104245.289139	36
20	317.423512	7	57049.199783	20	114823.94133	36
21	288.800416	6	52996.785149	23	108857.258942	34
22	311.202401	6	54449.246704	23	102069.259422	36
23	226.149871	7	53806.471344	21	95024.52264	35
24	258.801216	5	58588.215469	22	113719.784393	36
25	268.787345	7	53098.863617	22	105550.2397	36
26	290.496099	6	50874.238564	21	108321.213905	35
27	318.591564	7	54556.366089	25	110759.756439	35
28	253.100093	6	55594.022305	22	119703.695831	37
29	257.999482	6	54408.740974	25	108653.213264	39
30	317.920702	6	56980.756851	22	114205.463249	38
Average	287.749326366667	6.03333333333333	55624.0061857667	22.4333333333333	107874.742523333	35.9666666666667
Std. Dev	24.7018482581174	0.668675135459372	2544.70550235336	1.81342376380328	6706.6049593287	1.56432938883779

Table 7.16: The entire experiment data we have collected using our GWO approach with $c = 2$ and a population of 25.

Run	GWO (c = 4, Pop. Size of 25)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	295.290119	5	57590.204044	21	99250.338013	36
2	364.30381	6	55786.4907	19	115392.5942	36
3	283.958384	6	56478.70369	19	100622.106056	34
4	308.185705	6	59149.522892	21	100370.279572	37
5	279.100698	5	52092.835846	24	109341.424805	40
6	254.694568	7	54517.030029	23	111078.602165	34
7	299.122021	5	57337.25061	25	87110.57618	38
8	308.245055	5	56083.913353	21	116746.599121	36
9	310.631233	6	56986.768219	19	112465.605843	37
10	293.218722	6	53868.101875	21	110742.670616	37
11	290.615551	7	55411.816826	21	102156.222603	39
12	310.314302	6	50825.278824	24	112943.161697	34
13	284.625909	6	51448.845734	21	96147.747742	34
14	330.488841	6	57670.047066	22	106707.319443	37
15	271.799747	6	57224.530266	21	106312.161995	41
16	308.069251	7	53504.969574	21	109508.693184	39
17	307.75226	6	54594.552979	22	98866.16114	34
18	241.580049	6	56112.939285	19	103499.234871	33
19	228.710226	6	56761.539284	23	88510.688866	35
20	291.547085	6	56455.574303	22	97130.734337	38
21	373.604858	5	59194.486832	23	115666.25502	32
22	288.904785	7	51962.636154	20	100993.999092	37
23	351.234066	5	54615.561333	20	108507.541492	35
24	271.269831	6	52326.159271	20	102196.612885	35
25	345.786381	7	55869.035088	21	106251.775101	40
26	251.753436	6	53282.332359	20	94795.74794	36
27	316.554537	5	53018.602661	22	110122.478073	36
28	321.758455	5	58369.596535	20	94860.329987	42
29	263.652587	6	55446.598206	21	95583.080429	33
30	318.320174	6	53106.935944	24	114233.876129	35
Average	298.836421533333	5.9	55236.4286594	21.3333333333333	104270.487286567	36.3333333333333
Std. Dev	34.1522620177737	0.661763578993857	2301.71070299619	1.62593916273628	8041.05756072353	2.48211996894386

Table 7.17: The entire experiment data we have collected using our GWO approach with $c = 4$ and a population of 25.

Run	GWO (c = 8, Pop. Size of 25)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	262.707255	7	55466.377586	20	104697.724777	37
2	311.009165	6	52879.782005	20	114784.794098	41
3	291.360188	6	54291.537682	22	106627.593742	34
4	362.348913	5	51966.878838	22	118225.260605	36
5	305.725908	7	57894.186882	22	107491.358063	37
6	344.391756	5	54056.77681	24	118719.490257	36
7	337.822041	6	53829.962364	24	100109.010674	33
8	319.080408	7	53949.482574	23	96509.502808	38
9	249.624192	7	55062.295784	22	95254.061554	38
10	317.502579	8	54612.372421	22	111357.790108	34
11	347.870933	6	55273.090096	25	99750.327438	37
12	294.693992	7	54738.413105	25	106767.40667	35
13	282.647407	7	54133.133018	25	111462.286797	34
14	292.732835	6	54017.253056	24	107217.431229	39
15	283.872915	5	53602.699944	26	106198.506428	44
16	321.749494	7	54680.752884	23	97637.581284	36
17	310.098661	5	53654.185509	23	97815.004234	34
18	292.552563	5	54514.880905	26	104283.744904	39
19	330.776446	6	54886.626793	23	108294.5578	47
20	368.435807	5	51250.638187	25	101209.694763	35
21	333.208825	5	53291.312008	22	104063.487877	39
22	307.813316	6	54018.854462	26	106063.043762	34
23	326.428337	6	55304.592007	26	99203.966431	37
24	358.536934	5	56567.198547	25	114660.357567	36
25	299.608566	6	52027.762772	26	101201.104996	38
26	288.695351	8	55043.384789	23	100895.692963	38
27	320.946527	7	54383.856281	22	110844.852486	40
28	363.365827	9	53825.792953	23	104716.713264	33
29	289.365061	6	54623.883316	25	100086.231628	35
30	294.247917	6	52351.437447	24	116673.723824	38
Average	313.640670633333	6.23333333333333	54206.6467008333	23.6	105760.743434367	37.0666666666667
Std. Dev	29.7958232730557	1.04000442085709	1334.40810225102	1.7340405277052	6557.69877516131	3.12865877767558

Table 7.18: The entire experiment data we have collected using our GWO approach with $c = 8$ and a population of 25.

Run	GWO (c = 12, Pop. Size of 25)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	280.100084	6	54382.193893	22	110420.231773	35
2	326.000024	6	56657.682411	19	107836.336555	39
3	339.59429	6	56128.932846	20	105101.680847	34
4	291.153993	6	56149.892471	22	99260.377151	36
5	324.951407	6	57060.862228	22	97755.446457	39
6	322.430838	6	62758.004044	20	114021.624924	38
7	325.058099	6	54246.820145	20	100348.282265	36
8	301.274588	6	54187.122147	20	97531.603271	40
9	341.434416	8	53573.432259	19	108106.979767	37
10	357.480109	7	56897.296356	23	116657.657318	37
11	345.915623	7	51811.845238	21	109709.232689	33
12	321.136856	6	53061.549057	24	100745.31324	40
13	327.757973	6	52248.577141	24	107016.641022	36
14	338.574615	7	55548.896332	26	105929.138687	39
15	354.648388	6	53266.573128	23	110405.852989	36
16	305.538115	7	54944.865646	22	121917.463058	36
17	274.450367	6	52603.816597	20	118235.825241	37
18	299.440072	7	54888.153915	22	116887.253868	36
19	320.987788	8	56474.669212	20	102034.555359	37
20	317.335714	7	59091.504066	18	93525.765816	38
21	358.033844	5	53225.246391	20	105195.959213	37
22	296.381484	6	56367.015659	22	107206.72065	39
23	340.041138	6	55241.328114	21	122687.700439	39
24	307.116007	7	54713.23642	20	105925.402702	37
25	255.639347	7	58269.998817	20	124181.531029	35
26	296.414082	5	56489.390518	18	99701.346191	36
27	264.475464	6	51328.437737	22	98267.529518	40
28	304.718431	6	60483.979401	26	116802.36039	36
29	305.344853	6	53979.122162	20	112001.866364	38
30	312.784809	7	53856.493675	21	112142.008118	43
Average	315.20709393333	6.36666666666667	55331.2312675333	21.2333333333333	108251.9895637	37.3
Std. Dev	26.4399229476443	0.718395402284138	2540.54370041386	2.01174711054387	8151.19724950765	2.07031565142896

Table 7.19: The entire experiment data we have collected using our GWO approach with $c = 12$ and a population of 25.

Run	GWO (c = 2, Pop. 50)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	252.959127	13	54847.391106	39	115453.917747	71
2	310.451593	13	53073.76783	40	105585.975105	71
3	324.7657	14	52564.010193	39	96569.906059	73
4	302.253564	12	62827.738159	42	88657.824898	69
5	287.59426	15	50270.968304	38	99648.274292	72
6	246.890955	12	51021.870609	39	121124.59779	68
7	329.037109	11	53698.933483	39	109386.855927	74
8	270.753061	12	48938.991478	37	100284.722839	68
9	297.364378	14	51457.838638	39	102671.318954	72
10	297.396819	14	55029.144333	40	100138.233009	72
11	264.955059	13	54034.453362	44	94063.746262	78
12	239.019053	13	54843.412094	39	107732.370728	76
13	269.378189	15	60974.241631	40	100197.140274	78
14	270.275415	13	53123.451248	41	112261.793114	75
15	245.398331	13	56983.71854	42	112999.618805	72
16	288.427367	13	55207.566151	43	105895.490013	74
17	233.585352	13	57260.047424	39	98834.100918	75
18	280.347926	16	58842.573586	39	101081.708069	76
19	307.761939	14	52738.607117	39	99121.191231	73
20	284.018757	13	53894.438091	42	99270.817657	75
21	280.025665	14	53150.059753	39	113995.070282	72
22	281.777046	13	47597.794662	42	99077.873138	73
23	221.18019	14	53971.207901	39	103581.175766	81
24	308.542643	14	53825.842094	38	100780.563087	69
25	302.184198	13	59349.959862	40	94936.180672	73
26	293.481107	12	56370.238155	43	97659.466835	69
27	300.88667	13	51595.317627	43	93650.395554	76
28	308.646566	13	57071.090172	37	103402.549973	74
29	272.924234	13	55864.520546	41	103434.275482	77
30	341.568304	13	54428.835281	42	100768.256989	74
Average	283.795019233333	13.2666666666667	54495.2676476667	40.1333333333333	102742.1803823	73.3333333333333
Std. Dev	28.9742518867792	1.01483252680985	3328.18744058766	1.87052147133163	7156.18271087496	3.110974271758

Table 7.20: The entire experiment data we have collected using our GWO approach with $c = 2$ and a population of 50.

Run	GWO (c = 4, Pop. 50)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	312.340316	12	53062.125809	43	91596.268288	75
2	292.071671	14	51271.728905	41	118655.595436	74
3	329.653256	12	55060.696541	41	102449.075523	72
4	306.763128	16	57916.673454	43	104931.357796	71
5	252.772092	12	50881.148052	45	122300.269909	79
6	351.977179	14	53539.74173	42	93781.166649	67
7	330.62476	15	54803.86525	38	108907.169487	71
8	389.711568	11	55295.007942	39	95312.145584	68
9	297.27606	13	53283.531433	38	90647.4767	67
10	277.617715	14	54563.107239	40	93732.829643	78
11	266.562111	12	52247.270363	43	109087.634438	72
12	300.58802	13	53049.529968	43	95105.831413	71
13	263.685979	14	56017.20726	44	110646.846546	69
14	301.834932	12	50960.144783	37	101672.54467	72
15	295.790956	15	54146.704651	38	116249.322273	74
16	268.800616	13	50250.080536	42	90599.06601	70
17	287.1534	12	51163.562675	39	112135.693672	69
18	294.190074	12	56055.587646	40	102759.575096	70
19	257.069981	12	52603.173042	43	99055.68399	76
20	247.273698	13	54079.614361	38	106487.916885	73
21	241.862298	13	54716.540192	44	114099.514786	70
22	314.056355	12	51900.551003	46	101358.406052	68
23	304.7226	14	50370.735947	43	91041.143768	69
24	325.064058	12	51801.091309	45	95614.268036	67
25	298.000466	13	56852.344414	46	102250.452179	70
26	243.562194	15	54890.816574	45	111223.215179	70
27	275.810019	12	57815.042976	45	104611.12146	71
28	286.811285	14	50459.697983	39	93398.191818	67
29	338.195192	15	53087.217148	47	102300.34594	70
30	275.541747	13	50513.264008	42	103654.365265	69
Average	294.2461242	13.1333333333333	53421.9267731333	41.9666666666667	102855.4831497	70.9666666666667
Std. Dev	33.7641257773172	1.25212463115859	2239.06725435468	2.83431355593084	8820.10238434929	3.12369512986908

Table 7.21: The entire experiment data we have collected using our GWO approach with $c = 4$ and a population of 50.

Run	GWO (c = 8, Pop. 50)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	259.516314	12	54177.974075	38	94851.361023	69
2	324.636963	15	52181.587135	41	108371.091469	75
3	269.794718	13	59710.615997	39	105659.087982	72
4	242.578002	13	54392.617035	41	95788.281204	71
5	277.550035	13	51893.418335	36	104205.210045	70
6	268.449108	12	52690.644062	39	108677.065521	72
7	295.814127	12	51080.219994	40	94337.464722	79
8	302.657652	11	52444.058189	40	98890.396248	77
9	251.909466	13	48844.175789	44	103281.15654	67
10	310.410302	13	52941.323196	42	94691.424179	69
11	300.186725	13	50015.423599	40	111466.642097	70
12	271.950485	12	53271.293297	41	100257.817604	76
13	328.766632	12	50271.230789	42	103823.807419	69
14	318.317183	12	52223.022621	44	92512.471001	73
15	286.950144	12	52495.684402	41	104429.74781	70
16	319.746528	13	53486.564278	40	107520.309296	77
17	401.791988	15	52635.489326	44	104163.873627	68
18	269.226305	14	54915.506729	40	100961.08773	71
19	278.087246	13	52754.325737	41	98450.999786	75
20	240.638127	12	52209.049438	42	109202.223549	72
21	329.440071	12	50330.414803	40	108166.858864	70
22	280.454088	12	49972.284843	44	97141.092979	69
23	413.077936	12	55224.250671	39	102738.584953	71
24	298.426238	12	53442.302917	41	95437.557976	77
25	362.974943	13	52942.692451	43	109236.066574	76
26	279.583288	11	52528.093407	42	112251.415863	71
27	314.159356	14	54601.247261	43	102901.634148	69
28	279.102985	14	54642.370102	46	96261.835789	73
29	296.096529	13	51280.921593	43	84929.672058	71
30	314.30529	14	51389.147156	42	96498.67128	74
Average	299.553292466667	12.7333333333333	52699.5983075667	41.2666666666667	101570.163644533	72.1
Std. Dev	40.469222577263	1.01483252680985	2062.76885562279	2.09980842038002	6490.61277032704	3.14423391250583

Table 7.22: The entire experiment data we have collected using our GWO approach with $c = 8$ and a population of 50.

Run	GWO (c = 12, Pop. Size of 50)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	330.967894	13	51707.706673	43	106886.297325	69
2	352.688213	14	52477.265709	42	102138.644287	70
3	326.198721	13	54071.188835	38	127234.147255	71
4	316.33189	14	51498.448105	46	109019.598618	71
5	290.179763	12	54326.654686	44	103742.514435	65
6	358.362449	13	51799.437698	42	110403.709251	68
7	312.598053	14	55847.500244	47	99383.761021	66
8	280.485977	13	51694.484573	47	112262.345184	65
9	346.624847	17	56477.689476	46	99664.26989	68
10	307.655417	15	55350.96994	45	103295.469814	69
11	318.316489	13	52508.811115	42	120704.796318	72
12	313.9823	16	50773.498726	45	112647.377289	73
13	286.658271	14	52073.785706	40	97144.798492	69
14	285.225881	14	51173.391113	40	128598.716599	65
15	266.707759	14	53180.308609	40	93067.630051	65
16	324.595953	14	49284.544159	40	105843.944901	69
17	380.423649	13	53861.001526	49	94020.885498	69
18	323.238861	13	49585.285339	40	93486.066071	68
19	357.150067	13	48920.979538	42	102269.656761	75
20	303.404414	14	53799.463486	43	97102.425781	69
21	277.139452	13	51834.894127	45	98196.685638	71
22	261.869799	13	53773.521938	42	106061.312057	72
23	342.69634	13	53797.093964	40	106350.569275	68
24	328.36665	12	49982.629631	41	108921.795296	70
25	267.253918	15	53048.273376	40	94603.179474	72
26	315.065236	13	56021.777359	39	92563.720146	70
27	381.586061	16	53813.8256	39	103393.719864	70
28	299.772833	13	53787.978058	40	107972.391861	75
29	315.251937	14	54349.40506	40	109941.11869	71
30	304.150171	12	54298.960732	40	113533.302567	72
Average	315.831642166667	13.6666666666667	52837.3591700333	42.2333333333333	105348.4949903	69.5666666666667
Std. Dev	31.7204847308938	1.18418699983352	1980.22102755171	2.86095394427529	9267.72959691125	2.69972327232375

Table 7.23: The entire experiment data we have collected using our GWO approach with $c = 12$ and a population of 50.

Run	GWO (c = 2, Pop. 75)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	298.520938	20	53141.273369	58	101530.62925	107
2	247.876691	17	53633.342445	62	89274.679817	102
3	381.271954	18	50851.902184	62	95287.577271	101
4	255.177597	22	53696.3871	60	101152.594696	103
5	294.831714	23	50902.627678	61	104005.611374	107
6	205.666955	19	57068.075058	57	105304.380791	99
7	302.115958	19	56211.556992	62	88740.484344	103
8	314.284962	17	52866.921768	62	101882.857147	107
9	266.953485	20	54239.673317	62	93008.283638	102
10	308.163814	17	53867.634491	62	99406.877457	99
11	320.826071	19	50179.684898	62	101618.364464	110
12	298.084522	18	54544.619156	61	100320.698097	102
13	308.806339	17	54300.903725	63	107151.828575	107
14	282.147528	18	55166.191742	66	97411.479393	103
15	298.474119	18	53445.058487	68	89903.97784	106
16	291.704501	19	50309.586052	61	98993.719925	108
17	270.690811	18	57659.66436	64	117173.305939	112
18	386.356476	17	56617.206116	59	95635.212196	110
19	256.02258	19	52217.748032	64	102206.51368	105
20	250.591168	17	52619.802849	59	98741.184265	103
21	297.072077	18	53319.960991	61	96027.794487	104
22	306.206314	19	57618.996025	59	103661.234585	107
23	245.324803	19	53255.321243	58	97855.144623	99
24	300.076243	23	53322.684452	62	100661.008976	109
25	341.507297	22	54297.791504	61	100964.315514	109
26	256.656206	17	54966.582184	60	105386.165337	102
27	262.987007	18	57045.288116	59	97001.08107	101
28	297.544054	20	52647.000465	61	94089.306458	103
29	258.89821	20	54702.205833	57	93904.516655	109
30	297.061259	18	55745.558327	60	96171.02298	102
Average	290.063388433333	18.8666666666667	54015.3749653	61.1	99149.0616948	104.7
Std. Dev	38.0436802516604	1.75643316732074	2050.7167136713	2.44032219466879	5833.24082935413	3.62129715757327

Table 7.24: The entire experiment data we have collected using our GWO approach with $c = 2$ and a population of 75.

Run	GWO (c = 4, Pop. 75)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	289.002406	19	52142.283493	60	98677.828976	106
2	277.052858	19	49880.523918	60	92124.010956	105
3	335.462648	23	53328.857437	62	103596.954788	106
4	377.541344	17	49640.026917	61	103328.22345	106
5	239.258536	19	51617.14695	57	96614.5653	106
6	325.637233	19	53735.167488	62	91132.62336	105
7	288.088882	17	49864.33725	70	97078.708122	108
8	285.496838	19	55286.558464	61	109522.060501	101
9	323.256618	20	51300.184471	59	105238.711536	103
10	287.471357	17	49627.393288	61	96470.319412	104
11	265.513964	20	55577.590172	62	107900.23465	104
12	278.710724	18	48762.911812	58	92805.567352	100
13	262.309272	18	53302.249931	61	106591.955357	107
14	288.189661	21	51965.122269	64	101124.411228	110
15	291.547712	19	48824.390396	63	98799.394608	114
16	267.058788	17	49248.646812	61	90563.94965	107
17	265.283893	19	56156.624268	62	99513.980186	113
18	365.171428	18	48752.443314	60	94829.357605	118
19	406.790997	18	51401.382912	59	89197.608078	112
20	295.505686	18	54511.452782	63	89742.548553	115
21	329.868032	19	50549.009491	59	121878.61042	108
22	263.242765	18	56138.764824	66	100696.10968	101
23	260.896212	18	51281.46727	64	105140.868805	107
24	332.599415	19	53231.359383	61	98453.998238	115
25	355.053306	19	56071.344505	58	94238.087372	110
26	310.583911	22	51826.126656	63	95764.557251	109
27	261.280996	18	51597.582993	62	98053.700539	115
28	305.331404	18	52095.70153	62	97999.711472	108
29	305.927029	24	50398.410248	63	98793.284912	112
30	321.044512	17	53014.826584	62	108595.115971	104
Average	302.005947566667	18.9	52037.6629276	61.5333333333333	99482.2352776	107.966666666667
Std. Dev	39.1344289013742	1.70900253223113	2313.4004317195	2.54251210736389	7069.6968084522	4.62737978538189

Table 7.25: The entire experiment data we have collected using our GWO approach with $c = 4$ and a population of 75.

Run	GWO (c = 8, Pop. 75)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	319.414222	19	53533.50576	64	89710.066864	115
2	328.165713	20	50595.545044	62	104972.593979	108
3	306.412776	20	50424.89698	61	99951.159401	110
4	376.979948	19	52540.922691	61	100989.191406	109
5	300.174307	18	51199.14978	60	89169.450142	112
6	279.19583	18	50648.013687	60	113760.078079	112
7	312.371843	19	52296.237488	62	99162.493286	108
8	260.155633	20	51729.699924	61	101318.449509	106
9	309.446858	19	50810.120876	65	98910.665161	106
10	300.916353	17	52049.903336	64	104835.718559	105
11	302.721873	20	53897.510803	66	106039.799339	111
12	321.177588	19	53753.174721	63	87299.715054	117
13	285.325578	18	54977.558044	64	91432.348927	112
14	312.951696	18	49276.248596	66	92447.678101	108
15	342.242961	19	51142.870007	66	102956.340828	116
16	246.916627	20	53668.493233	61	103917.16843	106
17	288.087718	20	53224.681351	62	96208.562988	106
18	292.023611	21	49689.811363	63	91004.287216	110
19	306.479581	21	50058.597248	61	94997.346939	107
20	307.767267	19	51180.542244	62	92252.138172	102
21	281.911149	22	50156.850952	63	101269.8825	106
22	365.744095	21	52686.439079	65	103977.26178	113
23	316.998798	18	51797.785599	61	112025.688389	107
24	293.730703	20	51566.563713	68	98071.09903	105
25	281.707827	20	51542.716793	61	98420.038445	108
26	358.463471	20	53318.161484	64	92146.580208	105
27	393.744452	20	50811.437714	66	101289.536804	116
28	304.854541	19	53189.377037	63	101786.061531	107
29	299.836358	20	50725.26915	64	100487.72049	104
30	302.820106	21	51564.429115	62	98249.547806	104
Average	309.957982766667	19.5	51801.8837937333	63.0333333333333	98968.6223121	108.7
Std. Dev	32.0156308267039	1.13714706536836	1419.1918023338	2.07586015962031	6443.60722715266	3.94924698191161

Table 7.26: The entire experiment data we have collected using our GWO approach with $c = 8$ and a population of 75.

Run	GWO (c = 12, Pop. 75)					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	311.266407	18	50203.176975	60	104539.105591	108
2	313.716906	21	49644.232903	62	96584.774765	113
3	288.457501	19	50234.59729	66	86942.304199	108
4	321.340706	18	51165.054924	60	98779.496071	115
5	272.457565	19	55524.684891	60	108422.175175	111
6	288.239145	20	54490.212311	61	93106.942497	107
7	263.701073	18	50293.834431	58	105715.713463	109
8	309.448008	20	51443.199959	65	95049.082172	122
9	287.407971	20	51524.595062	59	89120.334511	111
10	293.245684	20	52417.916771	65	91566.401367	117
11	284.921627	18	50844.285988	63	104532.725182	118
12	334.502174	21	51783.373306	58	99873.148727	113
13	335.122006	18	51240.802536	65	87224.183014	118
14	342.360371	20	51226.104279	58	94446.328514	119
15	386.133494	17	53765.692108	62	99027.918419	111
16	243.386427	19	51797.691086	67	105792.314682	103
17	278.076036	18	50207.38295	60	103782.227905	109
18	342.271969	19	51941.683334	66	93258.144852	108
19	286.475345	18	52165.466835	61	95130.892349	110
20	290.548313	20	50326.696167	63	98527.791496	113
21	291.333399	19	54636.728546	61	101805.933556	105
22	331.904596	20	50238.804039	61	97895.015785	116
23	264.303243	23	53149.875965	59	92221.346848	110
24	348.721866	20	52677.171204	60	90693.638336	113
25	413.874466	20	52407.621346	60	107340.909904	111
26	282.0196	21	51187.63504	67	100840.685257	113
27	294.260799	20	52864.498299	60	107007.960152	101
28	322.869648	19	51778.318222	62	89651.275627	111
29	303.557749	19	51775.92057	64	107554.843956	108
30	293.715599	18	52179.042252	60	96833.664429	107
Average	307.3213231	19.3333333333333	51837.8766529667	61.7666666666667	98108.9092933667	111.266666666667
Std. Dev	36.2239957721235	1.26854065851231	1430.57988385005	2.72514894668332	6511.43059062118	4.75563791241753

Table 7.27: The entire experiment data we have collected using our GWO approach with $c = 12$ and a population of 75.

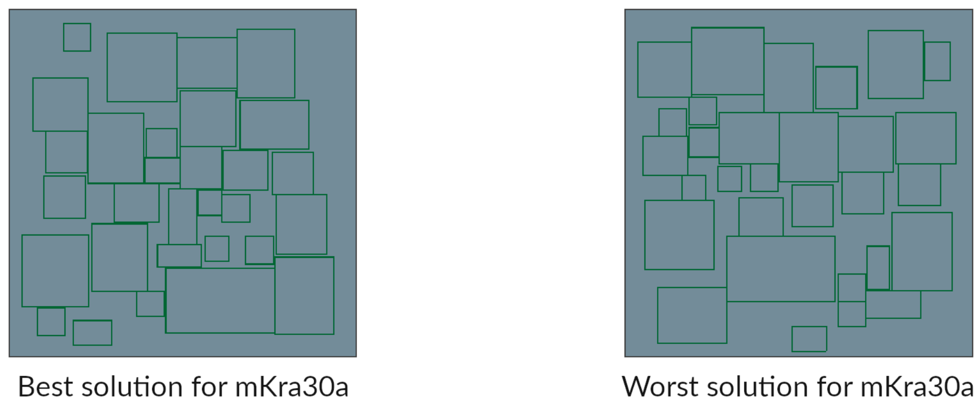


Figure 7.7: Visualizations of the best and worst solutions generated by our GWO approach for the mKra30a problem. The best solution was generated by $G_{50,8}$, with the worst generated by $G_{50,12}$.

7.2.2 Results of Other Approaches

Each approach has their own parameters, and the values we have set for those parameters are shown in Table 7.28. 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]. Our proposed GWO approach is not included in the following results since we have already discussed them in the previous subsection. However, since we are using a population size of 50 for the competing approaches, we will be using the GWO experiments with a population size of 50 to represent our GWO approach and determine its performance compared to the two other approaches.

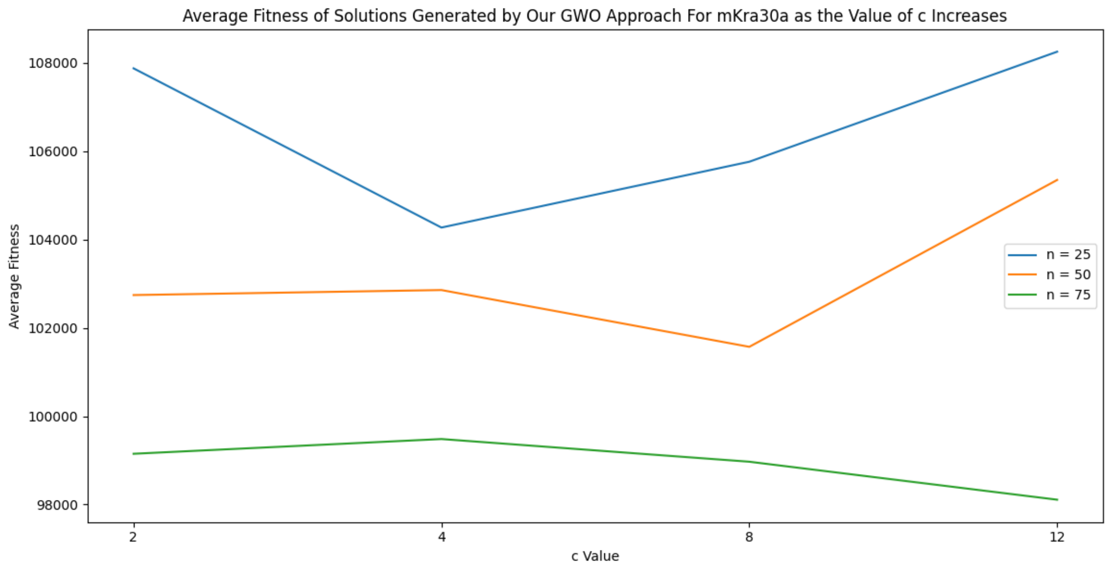


Figure 7.8: The average fitness of the solutions produced by our GWO approach as the c value increases when solving the mKra30a problem. Each line uses a different population with the blue line representing experiments using a population size (n) of 25, the yellow lines representing those with $n = 50$, and the green lines representing those with $n = 75$.

The results obtained for each approach is shown in Tables 7.29 and 7.30, respectively. As what the tables show, the competing genetic algorithm approach produces a solution that is better than our proposed GWO approach (with any value of c) and the PSO approach, with a fitness averages of 277.8299637, 50309.7310666, and 88945.0482127333 for the SFLP-II, mSFLP-III, and mKra30a problem configurations respectively. This is compared to our proposed approach's fitness averages. Fortunately for our approach, the PSO approach obtained the fitness averages of 322.6801845, 64734.7002284667, and 120530.7167285, proving that our GWO is not the worst approach. For SFLP-II, the best and worst solutions have fitnesses of 236.266584 and 351.084812 for the GA, respectively, compared to the PSO approach

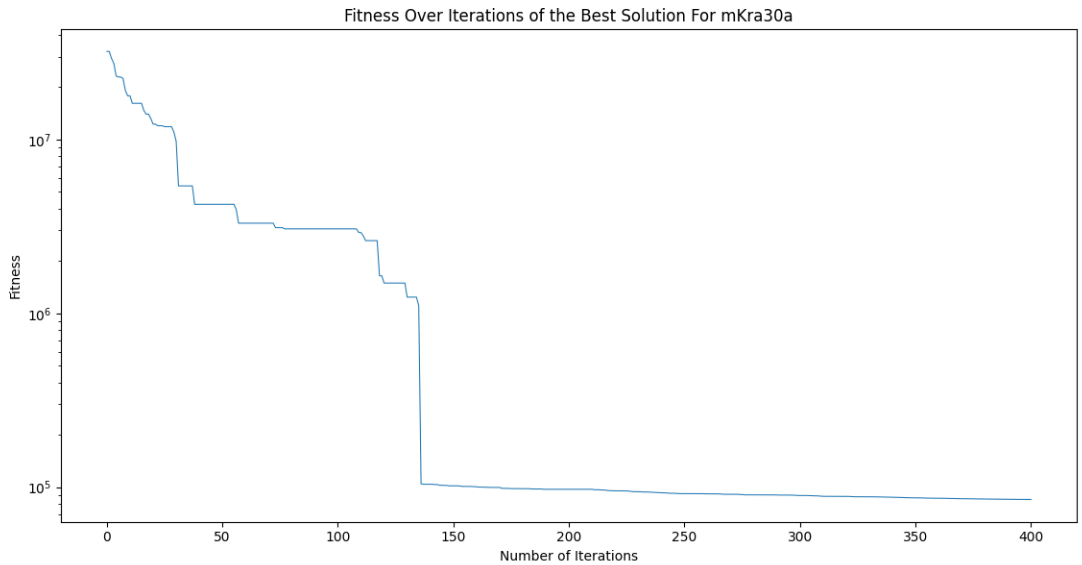


Figure 7.9: The fitness of the best solution generated by $G_{50,8}$ as the number of iterations increasee while solving the mKra30a problem.

which obtained 261.579758 and 371.217026. For mSFLP-III, the best and worst solutions have a fitness of 47803.047028 and 52113.727844 for the GA, respectively, compared to the PSO approach's 60132.200264 and 68087.431686. Lastly, for mKra30a, the best and worst solutions have fitnesses of 79772.279457 and 97487.293617, respectively, for the GA. PSO produces the poorest best and worst solutions with fitnesses of 105473.485001 and 136518.533897. This behaviour of producing the best average solution of the GA approach is attributed to the local search methods, which relatively exhaustively finds a better solution in a small area near the best solution found so far in each iteration. These local search methods intensifies the exploitation phase of the approach. Our GWO approach also exploits the local area, but it is not as intensive as the GA approach and only occurs at a later time in a run, similar to how the PSO approach behaves. Notice as well how PSO produces the worst solutions on

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

Table 7.28: Parameter values of the GWO, GA, and PSO approaches.

average among the three. This can be explained with how particles in the PSO approach are equally influenced by their personal best position and their swarm's global best position. This reduces the chances of particles from exploiting the area around the global best position. This behaviour also explains the observation we have with PSO during our experiemnts where the approach struggles to produce good results for the mKra30a data set. It requires multiple runs just to produce a single feasible solution. This is unlike our GWO approach where all wolves are influenced/led by the best three wolves, enabling them to exploit the space around the best found solution. In future studies, different behaviour may be observed when the PSO parameters are tweaked to different values.

The genetic algorithm approach is consistently the slowest among the three problems, with PSO being the fastest. Our GWO approach was found to be the second fastest algorithm. When solving the SFLP-II, GA had an average run time of 16.3666666666667s, compared to our approach's time and the PSO approach's 5.96666666666667s. With mSFLP-III, the amount of time GA took to solve the problem on average was now 180.633333333333s, while the PSO approach took 22.7666666666667s only. Moving to mKra30a, GA further increases the amount of time now takes 557.666666666667s on average, and the PSO approach just took 44.7s on average.

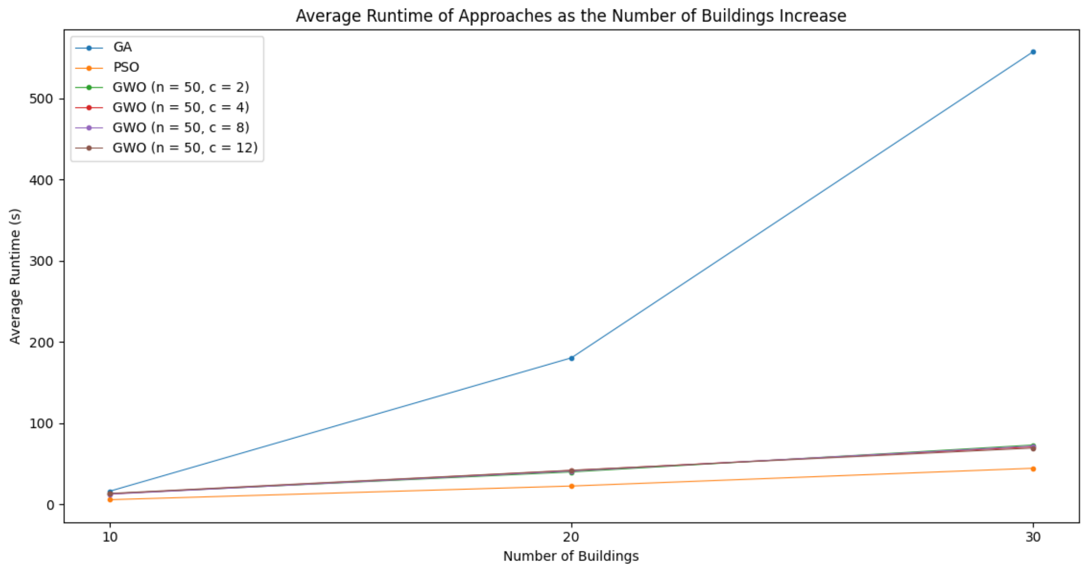


Figure 7.10: The average runtime (s) of each of the approaches as the number of buildings in a data set increase.

Figure 7.10 shows the impact of the number of buildings to the amount of time the approaches need on average to completely solve a problem. We can attribute this faster increase in average runtime as the number of buildings increase in the GA approach to what enables it produce better solutions on average—its local search methods. Since the local search methods perform a relatively exhaustive search in order to find a better solution, the GA will take more time to finish execution. Hence, we observe this phenomenon. This is not the case with GWO and PSO, due to the lack of local search methods. GWO may have taken a longer time compared to PSO due to the higher amount of operations that are performed in the metaheuristic. Better implementations, especially those that utilize SIMD operations, for both approaches may reduce the gap in terms of average run time between the two. However, basing from the equations in both metaheuristics, it is likely that PSO will remain faster than GWO. Further studies, however, are required to exactly determine how well each approach scales with regards to the number of buildings.

Problem	**Genetic Algorithm**				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	236.266584	351.084812	277.8299637	28.733389801383	16.3666666666667
mSFLP-III	47803.047028	52113.727844	50309.7310666	1045.05795299065	180.633333333333
mKra30a	79772.279457	97487.293617	88945.0482127333	5158.91507488963	557.666666666667

Table 7.29: Results obtained from using the competing GA approach.

Problem	**Particle Swarm Optimization**				
	Best	**Worst**	**Avg.**	**Std. Dev.**	**Avg. Runtime (s)**
SFLP-II	261.579758	371.217026	322.6801845	32.4426073658366	5.96666666666667
mSFLP-III	60132.200264	68087.431686	64734.7002284667	1890.32624601396	22.7666666666667
mKra30a	105473.485001	136518.533897	120530.7167285	8523.84757756761	44.7

Table 7.30: Results obtained from our proposed PSO approach.

We can further obtain insights from our results, by looking at the best solutions generated by each approach. Figures 7.11 to 7.13 show the fitness graphs of the best solutions using the SFLP-II, mSFLP-III, and mKra30a data sets. We can observe that in the early stages of all approaches, explorations is being performed. Gradually, exploitation takes over exploration to find the best possible solution to the problems.

Basing from the graphs we have, GA and GWO continuously exploit their abstract search space. We can attribute this behaviour to how each approach is designed. With the GA approach, remember that a local search algorithm (see Algorithm 7) is always being applied to the best solution the algorithm has produced (see Algorithm 9 for details on the GA approach). Another local search algorithm (see Algorithm 8) is also being applied to the best solution, but only during the last 50 iterations of the algorithm. This results in the algorithm constantly exploiting the area the best solution occupies in the abstract search space and therefore further improving the best solution. Hence, the behaviour we observe in the graphs. It should be noted that we argue that the local search algorithms is what allows the GA approach to produce the best solutions on average. GWO, on the other hand, does not depend on another algorithm to fuel its exploitation. This is also a reason why it performs faster than GA. In GWO, buildings are constantly shifted around even if the number of iterations is almost near zero. However, the amount of shifting gradually reduces as the number of iterations increase (see Subsection 1.2.1.2, while also taking Subsection 5.3.2 into account). This still allows buildings to find better position while reducing the risk of intersecting with other buildings. This design of our GWO approach, just with the GA approach, allows our approach to exploit the area around a local optimum and further improve the best solution it has found (along with the other solutions our approach has generated).

PSO, on the contrary, has its fitness becoming and remaining relatively stable as the number of iterations increase. This results in an almost flat line for most of the iterations in the graphs. This suggests to us that it struggles to exploit the region around its local optimum. At the start, however, it is able to explore and find good solutions. We can attribute this difficulty of the PSO approach to the fact that the amount of shifting buildings are subjected to is more likely to be large, which can be too much for later iterations during an execution. As described in Subsection 6.2.1.6, the movement for the search of a better local optimum of each particle/solution in PSO is influenced by the particle's previous velocity, its personal best solution found, and the global best. It is these last two factors that has the most impact in the building shifting in our PSO approach, as they have their c_1 and c_2 parameters set to

2.00, which results in the higher likelihood of the building shifting to be too much. Hence, the behaviour we are observing in the line graph for the PSO approach. Note that the aforementioned parameters affect how much a particle is influenced to move towards the direction of the particle's best solution and the global best. This does suggest that tweaking the parameters values for our PSO approach may yield better results.

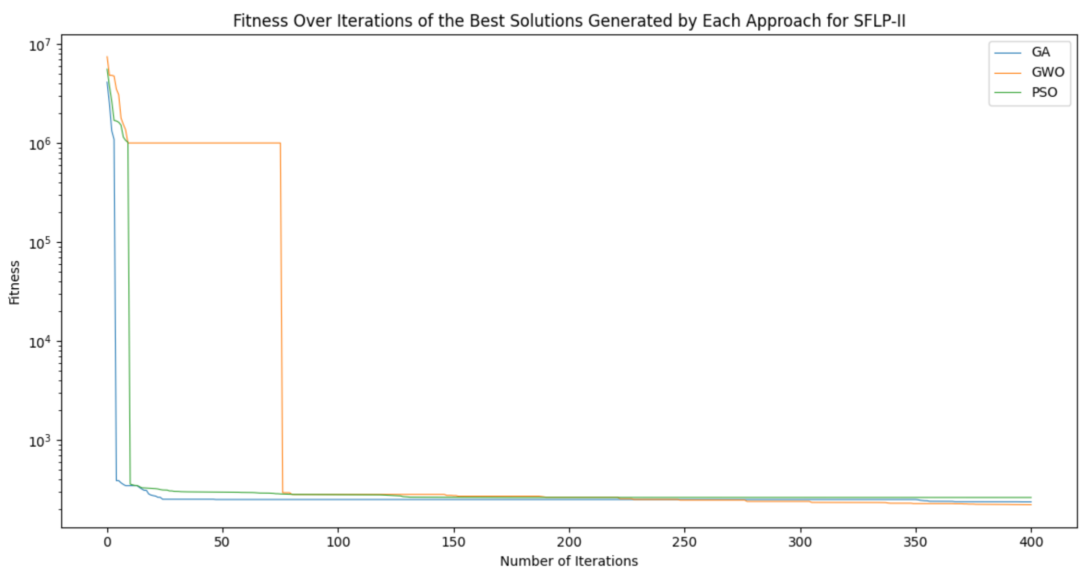


Figure 7.11: Fitness over time of the best solutions for the SFLP-II produced by the GA, GWO, and PSO approaches.

Another avenue we can use to gather insights is through the visualization of the results produced by the approaches mentioned in this study. Figures 7.14 to 7.16 show a visualization of the best results. Notice that with the hybrid GA approach and our GWO approach, the buildings tend to clump together, which is what we want to happen, based on our objective function. For our hybrid GA approach, we can attribute the result to the local search method as well as the mutation operators

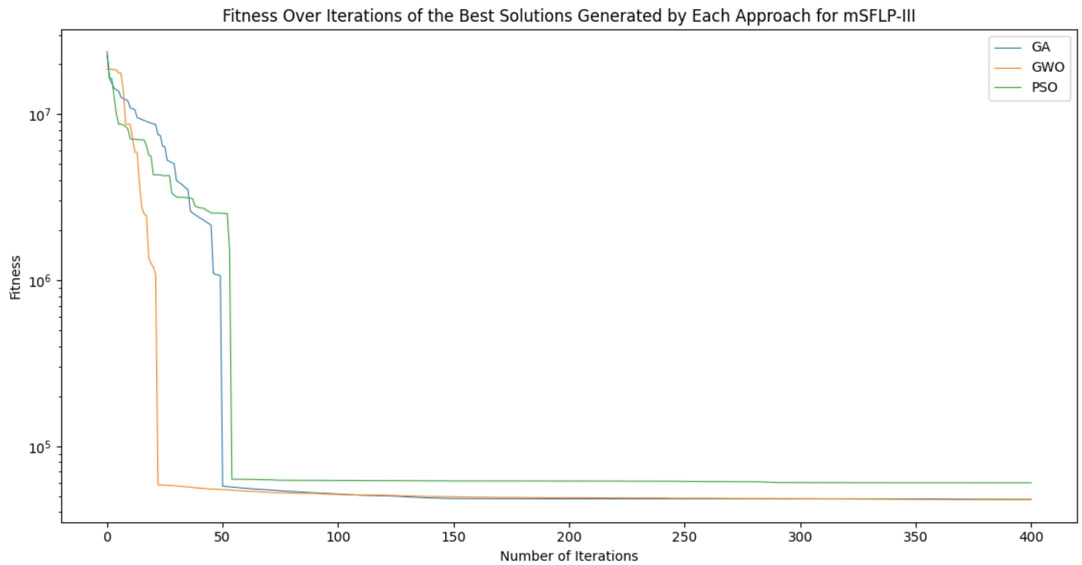


Figure 7.12: Fitness over time of the best solutions for the mSFLP-III produced by the GA, GWO, and PSO approaches.

as they were key to ensure that the buildings are close to each other. The crossover operator is also instrumental in achieving this result by finding combinations that will lead to the result. Our GWO approach also makes buildings clump together but not to the same degree as the GA approach, as can be observed from one building being far from the rest of the buildings in mKra30a data set in Figure 7.15. The clumping ability of our approach is attributable to how solutions are allowed to perturb their buildings to positions relatively far from the buildings positions in the best three solution initially. Eventually, our approach will decrease the distance of the buildings in a solution from the leading solutions. Note that the leading solutions eventually become similar to each other, which help drive the reduction of the degree of building shifting. This gradually decreasing shifting of the buildings will lead to intersections from being resolved and reducing the distance of buildings from each

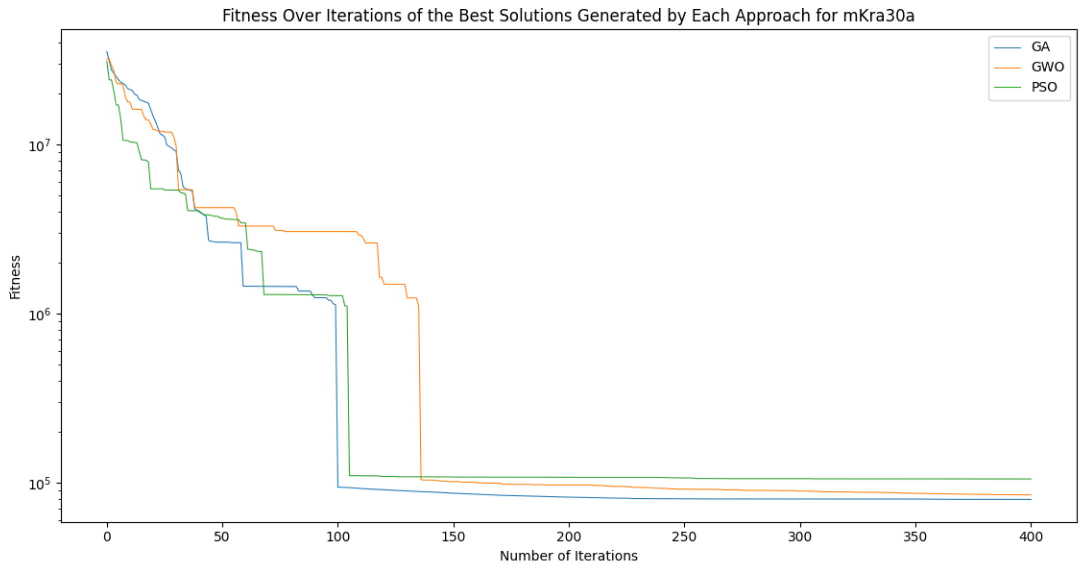


Figure 7.13: Fitness over time of the best solutions for the mKra30a produced by the GA, GWO, and PSO approaches.

other. The intersections are resolved by reducing the chances of buildings being to moved to a relatively further position where they would still intersect with another building, and gradually pushing intersecting buildings away from each other towards non-intersection. Note that the objective function has a lower penalty for solutions with buildings that do not significantly intersect. The decreasing shifting also encourages buildings to move towards each other due to the fact that smaller shifts have lower probability of causing buildings to intersect with one another too deeply or at all, which allows buildings to move to positions that are closer to the other buildings but without any intersections. Finally, as one can notice in Figure 7.16, the PSO approach struggles to produce a solution where the buildings are clumped together. This deficiency is not necessarily clear with a small number of buildings, but it does as the number increases. This behaviour is due to how our PSO approach, as discussed earlier in the previous paragraph, makes it easier for building to shift too much and cause building intersections to be more likely.

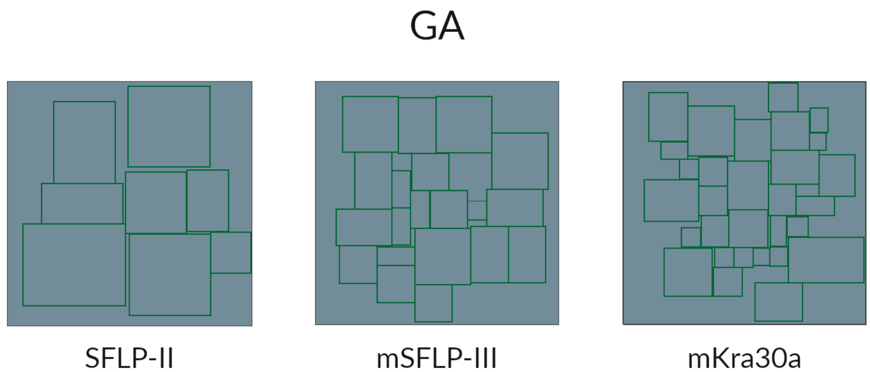


Figure 7.14: Visualization of the best solutions produced by the hybrid GA approach for the three data sets used in this study.

For reference, Tables 7.31 and 7.32 provide the detailed numbers we have obtained in our experiments for the GA and PSO approaches, respectively. Results of the GWO approach using a population size of 50 used as the representative data for our GWO approach for comparison against the GA and PSO approaches are already provided by Tables 7.20 to 7.23.

The performance of the GA approach in this study is definitely noteworthy. It produces the best solutions on average among the three approaches. However, based on the results, the GA approach does not scale well as the number of buildings increase, compared to our approach and the PSO approach. PSO definitely shows

Run	GA					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	266.528195	16	49947.365097	182	91150.52594	547
2	275.47149	16	51340.576492	178	86806.55534	551
3	266.351613	17	49861.753159	179	84091.301003	550
4	308.826011	16	49252.501534	177	97487.293617	550
5	263.646755	16	50780.747742	181	94015.955956	556
6	294.516963	17	49587.689148	182	95859.984146	557
7	272.558493	17	48958.183441	180	89946.6082	551
8	330.859349	16	48662.193741	180	88766.381279	555
9	308.087156	17	47803.047028	180	91838.419716	548
10	250.671959	16	49296.756157	178	79772.279457	544
11	303.593238	17	51669.300724	179	85323.813148	538
12	275.874591	17	50343.115303	183	96152.109024	542
13	285.168494	16	51452.603043	180	93274.746521	553
14	268.42481	17	50330.552826	183	87103.569084	551
15	256.26808	16	50043.851532	179	91260.246647	553
16	248.508814	16	50312.167366	183	93187.412254	557
17	256.90695	17	49781.92263	185	86379.197144	571
18	281.037801	16	50783.211159	183	92814.148727	562
19	253.327052	16	48951.801483	180	89999.578728	568
20	291.173308	16	50682.090927	180	90016.975861	552
21	236.845447	16	51750.123131	181	94116.278717	566
22	255.045933	17	52113.727844	183	93903.068489	558
23	351.084812	16	51356.503441	182	91290.029549	564
24	288.641643	16	49658.713455	177	81967.175171	582
25	265.659544	16	50867.704594	182	88435.975075	575
26	236.266584	16	50367.296349	179	81396.486565	569
27	324.972096	17	49485.836605	180	79991.033119	577
28	311.776283	16	50957.352654	181	81334.934044	554
29	264.748742	17	51506.650543	180	89664.76149	563
30	242.056705	16	51386.59285	182	81004.602371	566
Average	277.8299637	16.3666666666667	50309.7310666	180.633333333333	88945.0482127333	557.666666666667
Std. Dev	28.733389801383	0.490132517853561	1045.05795299065	1.95612800747016	5158.91507488963	10.6393327938762

Table 7.31: The entire experiment data we have collected using our hybrid GA approach.

Run	PSO					
	SFLP-II	Elapsed Time (s)	mSFLP-III	Elapsed Time (s)	mKra30a	Elapsed Time (s)
1	349.921185	5	64278.204849	21	115460.666122	46
2	365.357545	5	63842.174843	22	105473.485001	45
3	353.241308	6	67228.944778	22	117889.501312	43
4	296.976458	6	67293.847954	25	123930.586349	44
5	337.563747	7	64717.147179	24	117817.663696	48
6	319.264219	6	63914.649506	21	129141.190292	44
7	340.752566	5	63146.369568	24	131232.40274	41
8	337	5	68087.431686	22	118068.045341	44
9	369.166254	6	64523.811813	23	127204.096298	47
10	317.500725	6	64391.846161	22	127170.164375	44
11	267.553256	5	65543.256248	23	118574.181587	43
12	371.217026	5	63490.423851	24	133711.182159	45
13	303.141264	6	65320.857124	23	116703.547356	44
14	319.166245	5	65809.173042	23	126081.738121	43
15	325.424345	5	63372.697357	21	111884.375015	45
16	337.414429	8	65486.100739	24	111221.815865	41
17	328.956743	6	67511.591209	21	115048.387337	49
18	271.596369	6	61698.533489	23	120007.382874	46
19	307.163927	5	65688.555557	21	122245.078575	42
20	291.097535	8	61948.927948	25	135302.486481	46
21	294.035434	6	65192.457092	24	135998.582253	44
22	271.983335	6	66683.379341	23	105845.64386	46
23	369.234533	9	64707.545151	22	111971.493881	48
24	358.052214	6	60132.200264	23	116962.85556	47
25	322.684204	7	64733.312561	24	116335.920261	44
26	313.111456	5	63888.932556	25	117395.851891	44
27	261.579758	7	68010.421631	20	136518.533897	44
28	362.433082	5	62982.527634	22	121072.119308	46
29	287.482981	7	62928.84359	23	118000.14296	44
30	330.333392	5	65486.842133	23	111652.381088	44
Average	322.6801845	5.96666666666667	64734.7002284667	22.7666666666667	120530.7167285	44.7
Std. Dev	32.4426073658366	1.06619961038982	1890.32624601396	1.33088856325993	8523.84757756761	1.93248098531931

Table 7.32: The entire experiment data we have collected using our PSO approach.

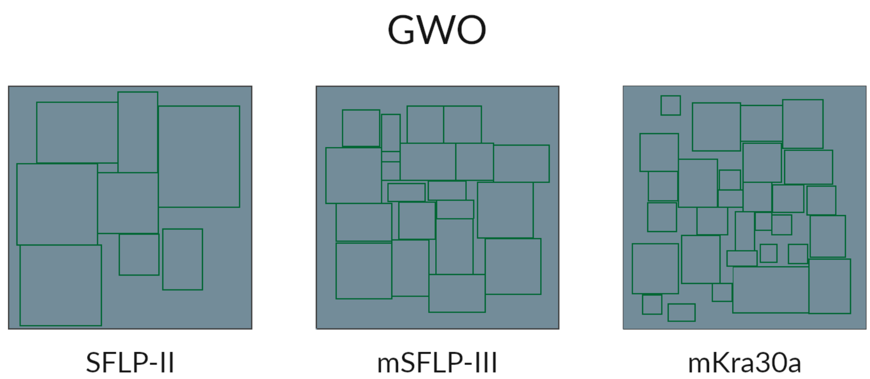


Figure 7.15: Visualization of the best solutions produced by our GWO approach for the three data sets used in this study.

the best average runtimes. However, it produces the worst average fitness. For faster speed, we traded performance. This is where our approach shines. Our approach is consistently the second best when it comes to solution quality and speed. This shows to us that our GWO approach provides a balance between speed and performance. Our approach also requires only a few parameters. We argue that this will simplify and speed up experimental setups and configuration in later studies and applications. Importantly, the results also indicate that there is promise in further exploring the applicability of the grey wolf optimization algorithm in solving the facility layout problem.

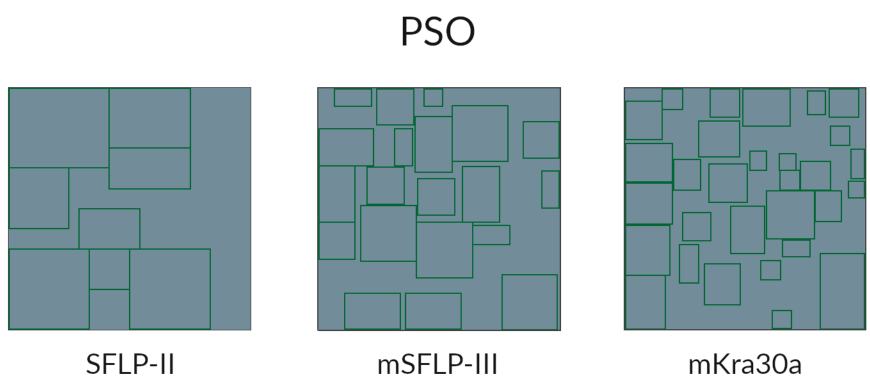


Figure 7.16: Visualization of the best solutions produced by the PSO approach for the three data sets used in this study.

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, among other approaches, genetic algorithms and particle swarm optimization. Our approach utilizes the grey wolf optimization to solve the problem. We have introduced modifications to this meta-heuristic in order for it to be able to produce feasible solutions. We have conducted experiments varying the value of the c parameter and population size of our proposed GWO approach, and compared this approach against a GA-based hybrid approach and a PSO approach. Results from our experiment indicate that a larger population size produces the best possible results. We also found that the value of c impacts the performance of our approach, and that the appropriate value for the parameter depends on the population size and the problem being solved. Additionally, we have found that the GA-based approach produces the best solutions on average compared to our modified GWO approach and the PSO approach. However, our results showed that there is promise in GWO as a viable algorithm for solving FLPs. The GA approach was shown to take longer to finish as the number of buildings increase. The PSO approach is the fastest among the three, but produces the worst solutions on average. Our approach, on the other hand, is the second best in both speed and solution quality. Hence, it provides a balance in speed and balance. 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. Additionally, 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. Another avenue is to identify whether the c parameter's value can be mathematically modelled instead of being a parameter. Subjecting GWO to different problems will also be an interesting endeavour to pursue as it can help with determining the impacts of the parameters on the performance of the algorithm.

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] Seyeda Mirjalili. The inspirations for Grey Wolf Optimizer - YouTube, 2020.

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

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

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] 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.

[63] 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.

[64] 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.

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

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

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

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

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