

CONSITEPLAN – A Multi-Objective Construction Site Utilization Planning Tool

by

Karthick Alagarsamy

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 7, 2012

Keywords: Construction site utilization planning, BIM, Genetic algorithms

Copyright 2012 by Karthick Alagarsamy

Approved by

Abhijeet Deshpande, Chair, Assistant Professor of Civil Engineering
Larry Crowley, Associate Professor of Civil Engineering
Wesley Zech, Associate Professor of Civil Engineering

Abstract

Construction Site Utilization Planning (CSUP) involves identifying, sizing, and positioning the temporary construction facilities required during different stages of the project within the available site boundaries and using available offsite facilities. CSUP has a significant positive impact on worker productivity, costs, and duration of construction. An effectively planned layout could potentially result in improvements in quality of work and safety of operations. This research work presents a temporary facility layout planning tool aimed at building on existing research on site layout planning. A user friendly, mathematically robust optimization tool that enables the user to model geometric, temporal and project specific requirements is presented in this thesis. This tool uses an optimization engine based on genetic algorithms. The tool provides an option to conventional distance measurement formulas (i.e., Euclidean and Manhattan). The tool is designed to enable the user to specify project specific constraints such as unusable areas, use of offsite locations for temporary facilities, eliminating double handling of materials by locating the required temporary facilities (e.g., laydown areas) within the reach of a user defined crane location. The tool also has the capability to use data extracted from REVIT Building Information Modeling (BIM) models. This tool is designed to be user friendly, mathematically robust and practical.

Acknowledgments

I would like to thank my advisor, Dr. Abhijeet Deshpande, for his guidance and support throughout the research work. I would also like to thank Dr. Larry Crowley and Dr. Wesley Zech for their inputs on my thesis work and for serving on my committee.

Table of Contents

Abstract.....	ii
Acknowledgments.....	iii
List of Tables	ix
List of Figures.....	x
List of Abbreviations	xii
1 Introduction.....	1
1.1 Construction Site Utilization Planning	1
1.2 Problem Statement.....	4
1.3 Research Objectives.....	6
1.4 Organization of the Study	6
2 Literature Review.....	9
2.1 Research in Layout Planning	9
2.2 Research in Construction Site Utilization Planning.....	10
2.2.1 Classification of Construction Site Layout Research	10
2.2.2 Optimization Techniques.....	10

2.2.3	Layout Optimization Using Population Based Meta-Heuristics.....	13
2.2.4	Layout Optimization Using Swarm Intelligence	15
2.2.5	Layout Optimization Using Trajectory based Meta-Heuristics	16
2.2.6	Graphical Site Layout Methods	17
2.2.7	Dynamic (Temporal) Site Layouts.....	18
2.2.8	Distance between Facilities: Euclidean or Manhattan	19
2.3	Review Summary.....	22
3	Evolutionary Algorithms	26
3.1	Genetic Algorithms.....	26
3.1.1	Trail-and-Error Optimization.....	26
3.1.2	Temporal Optimization.....	26
3.1.3	Continuous Optimization.....	27
3.1.4	Constrained Optimization	27
3.1.5	Minimum Seeking Optimization.....	27
3.2	Traditional Optimization Algorithms for Minimization	29
3.2.1	Exhaustive Search	29
3.2.2	Analytical Optimization	30
3.2.3	Nelder-Mead Downhill Simplex Method.....	31

3.3	Genetic Algorithms	33
3.4	Genetic Operations.....	36
3.4.1	Selection	36
3.4.2	Mating	39
3.4.3	Mutation	40
3.4.4	Convergence.....	41
3.5	Summary	42
4	Site Utilization Planning Algorithms.....	43
4.1	Introduction.....	43
4.2	Algorithms	43
4.2.1	Distance Measurement Algorithm.....	43
4.2.2	Optimization Algorithms	51
4.2.2.1	Initial Solutions.....	52
4.2.2.2	Solutions Evaluation	60
4.2.2.3	Genetic Operations Algorithms: Crossover and Mutation.....	71
4.3	Summary	75
5	CONSITPLAN – Architecture Review	76
5.1	Introduction.....	76

5.2	CONSITPLAN – Process Flow.....	76
5.3	CONSITPLAN – Input.....	79
5.3.1	Site Parameters.....	79
5.3.2	Layout Parameters	81
5.4	CONSITPLAN – Optimization Components.....	89
5.4.1	SplitBoundary	89
5.4.2	ExtractFreeArea	89
5.4.3	ComputeDistance.....	89
5.4.4	InitialPopulation.....	90
5.4.5	EvaluateSolution	90
5.4.6	GenerateSolutions	90
5.4.7	DrawDimensions.....	91
5.5	Summary	91
6	CONSITPLAN – Illustrative Example.....	92
6.1	Introduction.....	92
6.2	Illustrative Example	94
6.3	Analysis of the Impact of Optimization Parameters	99
6.3.1	Impact of the Number of Iterations on Objective Function Value	99

6.3.2	Impact of the % of Probability of Mutation on Objective Function Value	100
6.3.3	Impact of the selected Grid Size on the Objective Function Value	101
6.3.4	Number of Iterations, Grid Size vs Computation Time	102
6.4	Summary	104
7	Summary and Scope for Future Research.....	105
7.1	Introduction.....	105
7.2	Conclusion	111
	References.....	113

List of Tables

Table 1.1 No. of fatal accidents in Construction from '05-'09.....	3
Table 2.1 Literature Summary	25
Table 3.1 Rank Weighting	38
Table 4.1 Representation of Solution Chromosome	56
Table 4.2 Objective Functions used in Literature	61
Table 4.3 Preferred Proximity Relationship Mapping	63
Table 4.4.a Parent Chromosome I.....	72
Table 4.4.b Parent Chromosome II.....	72
Table 4.5.a Offspring Chromosome II.....	72
Table 4.5.b Offspring Chromosome II.....	72
Table 6.1 Case Study Example Temporary Facilities.....	95
Table 6.2 Case Study Example Preferred Proximity	96
Table 6.3 Case Study Example Optimization Parameters	96

List of Figures

Figure 1.1 Time/Safety Influence Curve	4
Figure 2.1 Distance Measurement Formulation.....	20
Figure 3.1 Flowchart for binary GAs.....	35
Figure 3.2 Single-Point Crossover Example.....	40
Figure 4.1 Euclidean Distance Measurement Approach.....	44
Figure 4.2 Manhattan Distance Measurement Approach (applicable scenario).....	45
Figure 4.3 Manhattan Distance Measurement Approach (inapplicable scenario).....	46
Figure 4.4 Improvised Distance Measurement Approach	47
Figure 4.5 Distance Measurement Strategy	48
Figure 4.6.a Horizontal Alignment	53
Figure 4.6.b Vertical Alignment	54
Figure 4.7 Illustrative Example.....	55
Figure 5.1 CONSITEPLAN Architecture Flow.....	77
Figure 5.2 CONSITEPLAN GAs Optimization Process Flowchart.....	78

Figure 5.3 Site Space Conceptualization	83
Figure 5.4 CONSITEPLAN – Primary Screen	84
Figure 5.5 CONSITEPLAN – Facilities Detail	85
Figure 5.6 CONSITEPLAN – Unusable Areas	86
Figure 5.7 CONSITEPLAN – Cost of Interaction among Facilities	87
Figure 5.8 CONSITEPLAN – Preferred Proximity among Facilities	88
Figure 6.1 Case Study Example Layout	94
Figure 6.2 CONSITEPLAN – Site Utilization Plan 1	97
Figure 6.3 CONSITEPLAN – Site Utilization Plan 2	97
Figure 6.4 CONSITEPLAN – Site Utilization Plan 3	98
Figure 6.5 CONSITEPLAN – Site Utilization Plan 4	98
Figure 6.6 Objective Function Value vs. No. of Iterations.....	100
Figure 6.7 Objective Function Value vs. Percentage of Probability of Mutation.....	101
Figure 6.8 Objective Function Value vs. Grid Size	102
Figure 6.9 Number of Solution Iterations vs. Computation Time	103
Figure 6.10 Grid Size vs. Computation Time.....	103

List of Abbreviations

AI	Artificial Intelligence
BIM	Building Information Modeling
CSUP	Construction Site Utilization Planning
GA	Genetic Algorithms
GIS	Geographic Information System
GSA	General Services Administration
PTF	Primary Time Frame

Chapter 1: Introduction

1.1 Construction Site Utilization Planning

Construction Site Utilization Planning (CSUP) is the determination of the layout of temporary facilities, and is an integral and critical component of a project execution plan. Temporary facilities are those facilities which occupy physical space within the site boundaries, are not part of the permanent buildings/structure, and are only used to support the construction process. CSUP can also be defined as a decision making process for positioning the temporary facilities, which involves identifying problems and opportunities, developing solutions, choosing the best alternative and implementing the plan (Ning et al. 2010a). Typical temporary facilities in a construction project include material laydown areas, fabrication yards, concrete batching plants, equipment storage areas, and temporary site offices used by various contractors (Mawdesley et al. 2002).

Practitioners and researchers have recognized CSUP as a vital process in construction planning, which essentially is a problem of utilizing the available site space efficiently. It is widely accepted that the available site space on a construction project varies greatly based on several project specific requirements (Ning et al. 2010a). Many construction projects performed in urban areas have very limited working space at the site and do not have sufficient space for positioning all the temporary facilities required for construction. Therefore, planning should essentially be aimed at effectively utilizing the available smaller spaces within the site boundaries to execute the construction activities efficiently. In construction projects on large

sites, the project managers might tend to not focus on site layout planning at the beginning of construction phase due to abundant availability of space. In such cases, the layout of temporary facilities is done on an ad-hoc basis with no concern for optimality or layout efficiency. This results in increased costs of transportation of materials, and inefficient labor movement. For example, in extreme cases, decision on where to store the materials might only be made after the delivery of the materials to the project site (Mawdesley et al. 2002). Evidently, the careful planning of the layout of temporary facilities has a bearing on the success of construction projects whether the site is confined with limited space, or it is a very large site where travel between various facilities could be time consuming and slow down production (Li and Love 1998).

The location of temporary facilities and equipment is coupled with the sequence of construction activities at the site. Poor planning of the temporary facilities layout could potentially lead to work delays, temporary material storage, multiple handling of materials resulting in reduced labor productivity, schedule delays, loss of time and money, and unsafe working conditions (Mincks and Johnston 2010). For the construction process to flow smoothly and efficiently, utilization of site space needs to be determined with great care. Site spaces could be efficiently used by developing the site layout plan(s) which involve, i) defining the site boundaries ii) identifying the temporary construction facilities to be laid out iii) identifying the constraints between the facilities iv) determining the relative positions of the facilities that satisfy the relationships between them, and allowing them to function efficiently (Zouein et al. 2002). Consequently, a complete site layout should include the following aspects, i) identification of temporary facilities that need to be established ii) locations of the temporary facilities and, iii)

timing of the establishment and removal of specific facilities during the project (Mawdesley et al. 2002).

According to Szymberski (1997), consideration of safety in earlier project stages has better potential to influence the outcome and its overall ability to influence safety decreases as the project progresses from conceptual stage to project startup, which can be seen in Figure 1.1. By extension, this theory is pertinent for individual project phases (e.g., construction) as well, where the project manager can influence safety to a greater extent at the beginning of the construction process through well planned layouts.

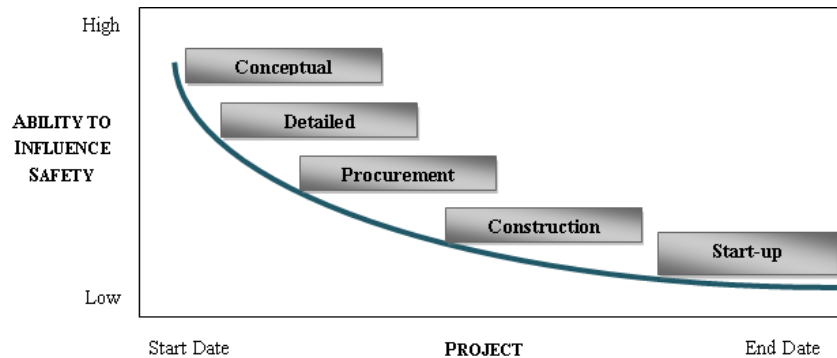


Figure 1.1: Time/Safety Influence Curve (Szymberski 1997)

Thus, CSUP which involves identifying the temporary construction facilities that are used to support the project is a vital activity that greatly contributes in achieving the project objectives. An efficiently planned site layout will result in one, some, or all of the following: i) reduction in project cost ii) improvement in quality of work iii) improvement in safety of operations on the project iv) improvement in project site environment (e.g., sequenced facilities positioning, elimination of untidy material storage) (Anumba and Bishop 1997; Mawdesley et al. 2002).

Layout Problem: Manufacturing, Architecture and Very Large Scale Integration (VLSI)

The layout planning problem involves the allocation of activities to space such that a set of constraints (for example, area requirement, adjacency requirements) are met and an objective function is optimized (usually transportation distance or cost) (Liggett 2000). It is encountered in several industries which include architecture, manufacturing, chip design, and construction. In manufacturing, a layout problem is typically concerned with an arrangement of physical facilities (i.e., departments, machines) primarily accomplishing two optimization objectives: minimizing the material handling cost, and maximizing some measure of closeness ratings (Rosenblatt 1986). In architecture, layout planning is characteristically defined as the assignment of discrete space elements to their corresponding locations while the space elements have relationships (topology, geometry) among each other. In solving this problem, the objective is defined as identifying an optimal plan by considering the interaction and travel cost between the space elements in the plan (Jo and Gero 1998).

VLSI is a technology for producing complex electronic circuits by combining a very large number of transistors efficiently in a very small area. The layout planning in VLSI involves identifying a layout that satisfies the required neighborhood relations (i.e., provide sufficient interfaces, route the corresponding connections between transistors) among the component rectangles and results in the smallest area, given a set of constraints on these rectangles (Heller et al. 1982; Kozminski and Kinnen 1984).

1.2 Problem Statement

The layout of temporary facilities is unique in every construction project due to the changing requirements on account of different site and building configurations, project specific

location of adjacent roads and buildings, subsurface conditions, types of foundation systems planned, construction execution approach, location of underground utilities at the site, schedule requirements, types of construction methods being used, material delivery and storage requirements, and the safety requirements at the site. Layout planning in construction can be conceptualized as a multi-objective problem where an optimal layout of the temporary facilities needs to be identified. This layout should incorporate the project specifications and constraints while simultaneously minimizing the cost of resource flow and improving the construction safety.

Moreover, the layout planning problem in construction also has a temporal aspect to it, in which a particular layout may need to change over the course of the project to meet the unique requirements of the construction process. In addition, several key, unique considerations need to be accounted for in solving this problem, further compounding the complexity of the problem. These considerations include offsite storage, worker safety, and areas that can't be used to locate temporary facilities. While addressing this multi-objective problem, a set of constraints specific to the construction project also need to be considered. These considerations could include, positioning the temporary facilities within the reachable radius of tower cranes, and avoiding close proximity between the temporary facilities that store materials of hazardous nature.

Thus, the problem/complexity in solving the layout planning problem in construction is that i) it needs to be defined as a multi-objective problem where an optimal layout to position the temporary facilities should be identified, while addressing the safety requirements of the project ii) temporal aspect of the construction projects should be accounted in solving the problem iii) other identified project specific constraints (i.e., usage of tower crane, unusable areas, and offsite

locations of temporary facilities) should also be imposed in addressing the layout planning problem in construction.

1.3 Research Objectives

The primary objective of the research is to develop a multi-objective temporary facilities layout planning tool that:

1. Makes a structured attempt to create an optimal site utilization plan based on the user requirements and project specific requirements
2. Measures the potential travel distance among the temporary facilities in the construction site
3. Considers project specific constraints
4. Provides a mechanism to address safety requirements in the layout planning process
5. Accounts for the temporal nature of the construction layout
6. Provides an interface with the BIM data from a given REVIT model

1.4 Organization of the Study

This thesis presents the development of a multi-objective optimization tool to address the layout planning problem in construction. The optimization strategy presented in this tool is based on Genetic Algorithms. This tool also addresses several shortcomings of the earlier studies which include measuring potential travel distance, addressing safety, temporal aspect, tower crane usage, and unusable areas while addressing the problem. Discussion of this thesis document is organized as follows:

Chapter 1 – Introduction: This chapter provides detailed discussion about the CSUP, its significance in successfully accomplishing the projects objectives, and when it should be done. Subsequently, a brief introduction about the layout planning problem encountered in several other sectors, and how this problem in construction is more unique and complex as compared to other sectors is reviewed along with the problem statement and objectives of this research study.

Chapter 2 – Literature Review: Several research studies have been performed during the last two decades with the focus of addressing the layout planning problem. This chapter reviews these studies on layout planning and provides a comprehensive, critical review of the optimization models presented in the literature.

Chapter 3 - Evolutionary Algorithms: This chapter provides an overview about the several evolutionary algorithms used in the past to address the layout planning problem. Since Genetic Algorithms is used in this study and is also prevalently used in the earlier studies addressing layout planning problem, a detailed discussion about Genetic Algorithms is also provided in this chapter.

Chapter 4 – Site Utilization Planning Algorithm: This chapter provides an overview of the optimization methodology incorporated in the tool developed in this study to solve the problem under discussion.

Chapter 5 – CONSITEPLAN – Architecture Review: This chapter provides information about the architecture of the programming developed in this study. It discusses the several coding functionalities developed in this study and how they are organized to solve this construction utilization planning problem.

Chapter 6 – CONSITEPLAN: Illustrative Example: This chapter discusses the implementation of the developed optimization tool in a case study project and its results.

Chapter 7 – Summary and Scope for Future Research: An epilogue that summarizes the research study, its unique attributes, and scope for further research is discussed in this chapter.

In this research, CONSITEPLAN, a windows based tool is developed to overcome the limitations of the current layout planning tools. The main objective behind developing CONSITEPLAN is to create a tool which addresses the practical requirements of layout planning without sacrificing the mathematical robustness of the underlying optimization algorithm. The main features of the tool include, space conceptualization, robust optimization engine, improved efficiency of the genetic optimization algorithm, development of an improved distance measurement algorithm, consideration of offsite temporary facilities, consideration of unusable areas, consideration of tower crane access, implementation of safety constraints, consideration of temporal nature of the CSUP problem, and ability to seamlessly interact with REVIT BIM files.

Chapter 2: Literature Review

2.1 Research in Layout Planning

Facility layout planning is encountered in several industries (e.g., architecture, manufacturing, chip design, construction), which essentially is concerned with the allocation of activities to space such that a set of constraints (e.g., area requirement, adjacency requirements) are met and an objective function is optimized (usually transportation distance or cost) (Liggett 2000). It is a classic problem of combinatorial optimization. On account of the complexity and importance of the layout problem, several researchers have significantly contributed in addressing this planning problem by channelizing the problem to specific industrial sectors.

For example, in the manufacturing industry, due to high market demand for variegated products, short product life cycles, and uncertain demand; layout planning has a significant impact on the system efficiency (Rajasekharan et al. 1998). Many studies have been done by several researchers since 1960s and numerous computerized packages have been developed to solve the plant layout problem e.g. CRAFT (Buffa et al. 1964), MATCH (Montreuil et al. 1987), FLEX-BAY (Tate and Smith 1995), FACOPT (Balakrishnana et al. 2003). A body of the published research in facility layout planning in the manufacturing, architecture sectors has also been summarized and surveyed by many researchers. For example, (Singh 2006) provides a detailed review of optimization methods proposed in the manufacturing sector, while (Liggett 2000) provides an excellent review of the algorithms proposed to solve the facility layout problems in architecture.

2.2 Research in Construction Site Layout Planning

2.2.1 Classification of Construction Site Layout Research

The research in construction site layout planning can be classified based on i) the mathematical optimization algorithm used (e.g., population based evolutionary algorithms, trajectory algorithms, hybrid algorithms, harmony optimization, dynamic programming, mixed integer programming, etc.), ii) formulation of the problem (e.g., space as discrete objects, space as area and shape), as suggested by (Liggett 2000), iii) whether temporal nature of layout planning is considered to generate more than one layout for a project over the construction phases (static vs. dynamic nature), iv) type of objective function used (e.g., single objective or multi-objective) and v) types of constraints addressed. Due to the overlap in these categories, the literature review is loosely organized on the basis of optimization techniques used and the conceptualization of the layout problem.

2.2.2 Optimization Techniques

A substantial amount of research effort has been directed at developing robust methods to identify optimal location of temporary facilities at construction sites. The conventional optimization algorithms such as linear programming or integer programming can't be applied to the problem because of the complex, discrete nature of the problem. The layout problem is usually characterized by a very large search space which also makes it infeasible to use deterministic exact algorithms, such as 'branch and bound' to find a perfectly optimal solution (Gholizadeh et al. 2010). The layout problems are known to be NP-hard (Nondeterministic polynomial time hard) wherein no known polynomial time algorithms exists and the identification of a perfectly optimal solution may require exponential computation time

(Arikaran et al. 2010). Due to these reasons, researchers have proposed several meta-heuristic methods to obtain acceptable solutions to the layout problems.

The solution strategies to solve the layout optimization problem can be classified into the following categories, i) constructive initial placement strategies ii) iterative improvement strategies iii) trajectory methods iv) population based strategies iv) hybrid strategies (Liggett 2000). A constructive initial placement strategy, similar to the strategy based on decision trees, locates facilities one by one, building a solution from scratch in a step-by-step fashion. This approach can be used to solve simple, direct optimization problems, but is of little practical value in solving complex layout optimization problems. The iterative improvement, trajectory based and population based optimization strategies belong to a class of optimization strategies known as meta-heuristics.

Meta-heuristic Optimization Methods:

Meta-heuristics are designed to search the solution space efficiently by generating candidate solutions and evaluating the performance of these solutions with respect to a predefined objective function. These algorithms are typically designed to enable the underlying problem specific heuristic to escape local optima and efficiently search a very large solution space by introducing an intelligently designed bias instead of randomly generating candidate solutions in the iterative cycles. This bias can be defined in the form of a memory bias (based on previously made decisions) or experience bias (based on prior performance of a candidate solution) or as a descent bias introduced in the objective function (Stützle 1999).

Iterative Local Search Method:

An iterative improvement solution strategy begins with an initial layout and attempts to improve it incrementally (Liggett 2000). This strategy basically involves a pair-wise exchange

between the current layout and a layout selected from the current neighborhood that demonstrates improvement based on some predetermined objective function. The performance of these algorithms is heavily influenced by the selection of initial population, the initial neighborhood, and the objective function. These strategies have the tendency to get trapped in local optima.

Trajectory Methods:

Other trajectory methods such as Simulated Annealing and Tabu search have been designed to enable the algorithms to escape local optima. Simulated Annealing allows for the movements that result in worse layout than the current layout, thus increasing the probability of escaping the local optima. Tabu Search, on the other hand uses short term memory in the form of Tabu lists to enable the escape from local optima. Other meta-heuristic explorative local search methods include Greedy Randomized Adaptive Search Procedure (GRASP), Guided Local Search, and Variable Neighborhood Search (Hansen and Mladenovic 1997; Pitsoulis and Resend 2002; Voudouris and Tsang 1999).

Population based Methods:

As compared to the previous strategies, population based search methods such as Genetic Algorithms (GAs) deal with a set of layouts in each iteration instead of a single layout. In GAs, an initial population of layouts is generated at the beginning and these layouts are manipulated using strategies inspired by evolutionary processes to generate another, improved population of layouts (Goldberg 1989; Holland 1992). Ant Colony Optimization is another population based search method where artificial ants represent various layouts in an ant colony. These ants are then guided to better layouts using pheromone trails.

The optimization techniques used in the studies reviewed in this article include various variants and hybrids of genetic algorithm, particle swarm algorithm, ant colony algorithm variants, simulated annealing, Tabu search algorithm, and harmony search algorithm. In addition, researchers have also attempted to solve the problem by applying Geographic Information System (GIS) functionalities and artificial intelligence as well. The remainder of this chapter provides a comprehensive, critical review of the optimization models presented in research literature in solving the CSUP problem, barriers to application of the algorithms in practice and a review summary.

2.2.3 Layout Optimization Using Population Based Meta-Heuristics

Li and Love (1998) proposed a pioneering research in which a solution for the site-level facilities layout problem using GAs was developed for the first time. They conceptualized the CSUP problem as laying out predetermined facilities in predetermined locations, (e.g., laying out 11 facilities in 11 locations). This formulation has been adopted in most of the subsequent studies (Gholizadeh et al. 2010), (Lam et al. 2009), (Tam et al. 2002), (Zhang and Wang 2008). The GA optimization model developed in this study uses ‘Strings’ to represent the facilities locations, over which the GA operations, edge recombination crossover operation (to preserve genetic information from the parental strings) and symmetric gene exchange mutation operation (to prevent the loss of good genes in crossover) are performed to identify an optimal solution. (Li and Love 2000) later acknowledged that the representation of the problem as an equal-area problem limited the utility of the algorithm and proposed an alternate model. In the latter study, the problem is defined as an unequal-area problem by creating a subset of available locations (which are smaller than rest of the locations, say L_p) and facilities (which cannot be accommodated in L_p), and imposing a genetic constraint to govern the non-assignment

requirement between those subsets. In both the studies, the authors' investigated the effect of the size of initial population on convergence and recommended that an initial population of 100, and 90-100 GA iterations could be used for effective optimization.

Due to the different functions of the temporary facilities and the space/topographic conditions, the CSUP problem is often an unequal-area facility layout where the facilities and locations have different sizes/areas. Harmanani et al. (2000) and Zouein et al. (2002) have developed an algorithm in which several GA operators (in addition to the traditional crossover and mutation) were used to solve the site layout problem with unequal size and constrained facilities. These operators include inversion, swap, move, rotation, flip2edge, add missing blocks, fix blocks, and aging. In this study, the layout problem is characterized as positioning a few rectangular blocks (fixed dimensions) within the available site, where the layout objects can take two orientations (0° , 90°) while satisfying the proximity, overlap, and orientation constraints. A function, Find-a-Set-of-Possible-Positions (Find-ASPP), is used to generate a set of feasible positions for a temporary facility in the neighborhood of a randomly selected location, and is invoked by several GA operators to evolve an initial population. Find-ASPP takes a given location as input and returns a set of four rectangles or less that describe feasible positions of the center-point of the temporary facilities. The authors tested the tool's performance for different total object-to-size-area ratios (OSAR) in differing cases. The analysis results showed that the tool performed well for smaller total object-to-size-area ratios (55% or less).

Hegazy and Elbeltagi (1999) have developed Evosite, a spread sheet based site utilization planning tool, which allows the user to define the space requirement for each facility as a group of unit areas (grids). These grids combine to take a user-specified shape (not limited to rectangular or square), allowing for more flexibility in the placement of the facilities and a

realistic representation of the site geometry. This tool also provides users the option of having unusable site areas by defining them as fixed facilities and thus excluding those spaces from allocation of facilities. The objective function developed is designed to facilitate effective placement of facilities within the site, while governing the desired closeness/proximity relationships (project manager's preference in positioning the facilities). The desired proximity relationships are determined using a qualitative method in which a pair-wise assessment of numerical proximity weights is assigned to the facilities that are being considered for layout.

2.2.4 Layout Optimization Using Swarm Intelligence

Many efficient decentralized, self-organized systems that exist in nature (e.g., colonies of ants, bee hives, schools of fish, etc). demonstrate an uncanny natural ability to find optimal paths. Over the last three decades, researchers have developed optimization algorithms that mimic these natural behaviors (Kennedy and Eberhart 2001). In particular, researchers have developed the Ant Colony Optimization (ACO) algorithm which is designed to mimic ants' foraging behavior and Particle Swarm Optimization (PSO) algorithm which simulates schooling behavior of birds and fish. Researchers have proposed site layout optimization solutions using these algorithms.

Zhang and Wang (2008) have developed a solution to the unequal-area layout planning problem using PSO algorithm that simulates social behaviors such as bird flocking together. In the PSO based algorithm, a solution approach is developed using priority-based particle representation of the layout solutions, transformation to the specific layout in consideration of non-overlay and geometric constraints, and framework for implementation. A comparative study with the solutions of GA based approach was also performed, and the results demonstrated that

PSO adopted solutions required fewer iterations to find the optimal solution than GA based solutions.

Lam et al. (2009) used a hybrid max-min ant system-genetic algorithm (MMAS-GA) to solve the layout planning problem, in which (ACO) and GAs were integrated. In this hybrid approach, the MMAS is used to generate a better initial population than the population randomly generated in GA and can lead to better optimal solutions with the objective of minimizing the transportation flows among the facilities.

2.2.5 Layout Optimization Using Trajectory based Meta-Heuristics

The main drawback of hill climbing optimization methods is that they tend to get trapped in a local optimum. Simulated Annealing and Tabu Search algorithms are two methods commonly used to enable the escape from a local optimum. Simulated Annealing is a probabilistic optimization method which works by emulating the physical processes whereby a solid is slowly cooled so that its structure is finalized at a minimum energy configuration. Yeh (1995) used annealed neural networks to solve the site layout optimization problem. Liang and Chao (2008) proposed an algorithm based on Tabu Search (TS) to layout temporary facilities. They conceptualized it as a problem of allocating 'n' facilities to 'n' locations with the objective of minimizing the travel distance among the facilities. To avoid local optima, the TS include two main processes, intensification (to explore the solution space from similarity) and diversification (to modify the move strategy to become more efficient in the solution search). Much like the contemporary formulation of the problem, these studies formulated the optimization as a static, one-to one (i.e., space as discrete objects) problem with the single objective of minimizing the cost / distance traveled.

2.2.6 Graphical Site Layout Methods

The layout of temporary facilities is inherently graphical in nature due to the fact that the site boundaries, existing permanent structures, the structure to be constructed, and temporary facility locations all occupy space in three dimensions. Layout planning can be done effectively graphically because of inherent human ability to conceptualize space and inter-relationships between spaces in two or three dimensions. Two notable studies have proposed solutions that utilize this idea.

ArcSite, developed by Cheng and O'Connor (1997), utilized the capabilities of GIS integrated with Database Management Systems (DBMS) to assist designers in solving the layout planning problem. Based on the knowledge based information provided by the user, GIS functionalities (e.g., buffer, erase, overlay) can be performed in the coverage files that identifies a feasible layout to solve the problem. The key benefit of the tool proposed in this study is that knowledgeable users can easily modify the project attributes and perform ‘What-If’ analysis with little effort.

Osman et al. (2003) have developed a novel hybrid approach in which GA is integrated within the computer aided design AutoCAD® environment to optimize the location of temporary facilities onsite. While most systems that use mathematical techniques aim at achieving one or more goals (typically minimization of cost) under problem-specific constraints, this hybrid system, in contrast, performs the optimization based on the geometric data provided in the AutoCAD® drawing (e.g., site boundaries, permanent facilities, obstacles). The drawing is used to detect space and ensure constraint satisfaction (facilities placed within site boundaries and non-overlap). While the first task (space detection) is performed only once to identify the space

available for placing temporary facilities, the latter task (constraint satisfaction) is performed in every cycle of the optimization process.

2.2.7 Dynamic (Temporal) Site Layouts

By its very nature, construction is a dynamic process that requires different temporary facilities over its life cycle. Thus, the layout planning problem in construction has a temporal aspect to it, in which a particular layout may need to change over the course of the project to meet the unique requirements of the construction process. In many of the earlier solutions for CSUP, the layout is considered to be static through the construction phase. But in ideal practice, the space available for laying out temporary facilities may change during the construction phase. For example, less space might be available during the substructure construction phase. As the project progresses and superstructure is constructed, more space may become available. In some cases, the building structure may be used to house some of the temporary facilities. This increases the complexity and the computational effort in the optimization process.

Zouein and Tommelein (1999) addressed the temporal nature of the problem by dividing the entire project duration into smaller time frames, termed as Primary Time Frames (PTFs), and generating a sequence of layouts, each spanning a time interval corresponding to a PTF and in combination corresponding to the duration of the project. Constraint Satisfaction and Propagation algorithm (to identify the feasible positions) and Linear Programming (to determine/evaluate the optimality of the identified layouts) were used for optimization in this study. Mahachi (2001) proposed a solution that used the PTFs (Zouein and Tommelein 1999) in conjunction with GAs to identify multiple layouts through the life of the project.

Mawdesley et al. (2002, 2003) recommended balancing the benefits and cost of moving a facility from one location to another in different phases of construction using an appropriate

fitness function. They modeled the dynamic nature of a construction project by associating site layout with the phase of construction and including facility setup and removal cost in the fitness evaluation function for the GA. Elbeltagi et al. (2004) proposed a GA based solution that accounted for the temporal nature of site layouts, which they defined as the reorganization of the location/areas of the needed temporary facilities (TFs) on-site at various schedule intervals, along project duration to suit the activities dynamic need of TFs. This was also the first study to incorporate the safety aspects in the solution approach by defining safety related temporary facilities needed on-site, defining safety zones around construction space, considering safety in determining the optimum layout, and changing the TFs' uses to alleviate site congestion.

Ning et al. (2010a, 2010b) presented a site layout algorithm based on ACO, in which the planning problem is conceptualized as a dynamic multi-objective optimization problem. The authors chose to address the dynamic aspect of the problem by dividing the project duration into smaller layout intervals and generating a layout for each of the layout interval. The evaluation function is designed to find a layout that minimizes the likelihood of accidents (improve safety) and minimizes the total handling cost of interaction flows between the facilities associated with the construction site layout.

2.2.8 Distance between Facilities: Euclidean or Manhattan

Distance between facilities plays an important role in optimization because the objective function is usually defined as a least cost or a least distance function with some constraints. In facility layout problems, distances are usually measured using Manhattan (measuring the sum of absolute differences of the coordinates of the points under consideration) or Euclidean (measuring the diagonal distance between the points given by the Pythagorean formula) formula

(Mawdesley et al. 2002). For example, in a two dimensional space, the Euclidean distance between two points $p(x_1, y_1)$ and $q(x_2, y_2)$ is measured using the formula:

$$d(p,q) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} \quad (\text{Eq. 1})$$

where,

$d(p,q)$ = distance between the points p and q

x_1 = X coordinate of the point p

x_2 = X coordinate of the point q

y_1 = Y coordinate of the point p

y_2 = Y coordinate of the point q

Whereas, in the case of Manhattan distance, the distance between the two points is measured using the formula:

$$d(p,q) = (|y_2-y_1| + |x_2-x_1|) \quad (\text{Eq. 2})$$

Figure 2.1 graphically illustrates the idea behind measuring distance between two points using Euclidean and Manhattan formula.

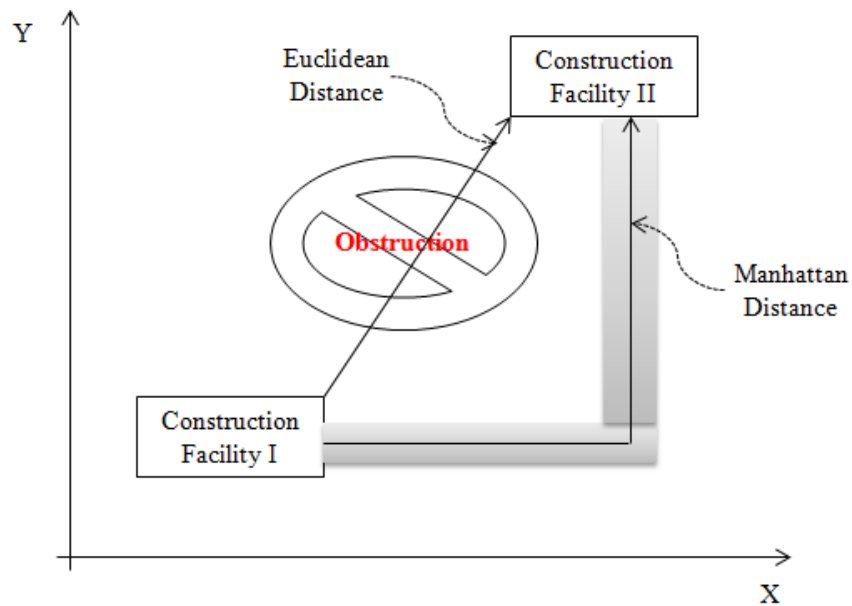


Figure 2.1: Distance Measurement Formulation

In most research studies reviewed in this article, the distance between facilities is calculated using the Euclidean formula. While it may be appropriate to use Euclidean distance in some cases, from a practical point of view, the Manhattan distance could better reflect the actual operation on a construction site than the Euclidean distance. In the cases of facilities between the presence of some interfering activity or object, Euclidean distance measurement does not represent the actual travelling distance (Sanad et al. 2008). Perhaps, in order to model reality at construction sites, it may be effective to incorporate these two types of measurements. It should also be noted that the models based on the Manhattan and Manhattan-squared distances are very efficient, requiring only a few minutes to solve. This is because the Manhattan objective function is piecewise linear and thus much easier to bound and the Manhattan-squared objective function is a convex function. On the other hand, the Euclidean objective function, despite of being convex, is often treated as a convex power function imposed by another concave power function, and therefore the solver may require a significant computational effort to prove the global optimality.

Sanad et al. (2008), acknowledging the lack of precision in measuring Euclidean distance (diagonal distance) between the facilities in evaluating a layout, proposed a multi objective optimization model which measured Manhattan distance between the facilities that aids to address the problem in a better aspect. In this study, Manhattan distance is measured between the facilities by dividing the entire site area into smaller grids and traversing through the grids between the facilities. The size of the grids dictates the number of possible locations for facility placement. If the size of the grid is too large, it may result in a fewer possible combinations while if the size of the grid is too small, it may generate a very large number of potential locations, resulting in significantly increased computational effort.

2.3 Review Summary

In the last two decades, a significant amount of effort has been directed at developing algorithms for identifying optimal construction site layouts. This section provides a brief summary of the literature and the observations.

There are two prevailing schools of thought about formulation of the layout problem. The first one involves making discrete one-to-one assignments where facilities are assigned to previously defined locations and the second one conceptualizes the site space as a continuous domain, theoretically creating infinite layouts. Many researchers have defined the site layout as a one-to-one assignment problem where 'n' facilities are to be located in 'n' predefined locations. While this definition may be limited in scope, it can be used to effectively model a situation where space is limited and a limited number of locations are available for temporary facilities. On the other hand, conceptualization of the site as a continuous space is computationally intensive because it increases the potential locations for locating the facilities exponentially. This has been overcome by dividing the site into grids and using available grids as discrete options for the layout.

The layout planning problem is typically defined as efficiently positioning the temporary facilities within the available site boundaries, while accomplishing an optimization objective. While this definition might be suitable for many projects, construction projects in urban areas tend to have limited working space at the site with fabrication carried out offsite. This aspect of site layout planning has not been adequately addressed in the current research.

It is a common construction practice to use the additional covered areas that tend to become available with the progress of construction for short spells for temporary fabrication and material storage. This aspect has not been addressed in current research on site layout planning.

In order to control the complexity of the problem, most studies consider site layout planning as a single objective problem involving minimization of cost or minimization of distance travelled. Though this characterization can lead to “good” layouts from the process efficiency perspective, it is essential to consider safety implications of site layout right from the beginning of the project. Hence, the problem should be defined as a multi-objective problem where the optimization objectives would be to reduce layout cost while simultaneously meeting safety requirements.

Most of the studies reviewed in this article have interpreted the site layout to be static (single layout for the entire project duration). But by the very nature of the construction process, the space available for laying out temporary facilities may change with time. It is important to consider CSUP as a dynamic/temporal problem. This dynamic aspect of creating a layout has been explicitly addressed by only a few researchers. However, it is worth noting that the existing static algorithms can be given a dynamic nature by generating layouts for different phases, including the facility setup cost, facility takedown cost, and unique unusable areas for every phase.

As the objective function defined in all the studies involves minimizing the distance travelled or transportation cost, the method used to measure distance between facilities plays a critical role in giving validity to the results. A majority of the research studies used the Euclidean formula to calculate the distance between facilities. This approximation may not be accurate in projects where such travel is not possible due to the obstructions. In such cases, Manhattan distance would provide a more realistic estimate of the distance travelled. The use of either of these formulas should not be hard coded in the optimization algorithm. The user should have the ability of choosing either or both based on the site specific conditions. This idea has not

been explored in any of the studies reviewed in this article. The review is summarized in Table 2.1. While several studies addressing the CSUP problem were reviewed in this chapter, the next chapter discusses about the various evolutionary algorithms/optimization techniques used to address this complex layout planning problem in construction.

Table 2.1 – Literature Summary

Study	Algorithm used	Objective	Problem formulation	Layouts over the construction phase	Distance measurement
(Li and Love 1998)	GA	Minimize traveling distance	Space as discrete	Single	Euclidean
(Li and Love 2000)	GA	Minimize traveling distance	Space as area	Single	Euclidean
(Harmanani et al. 2000), (Zouein et al. 2002)	GA	Minimize layout cost	Space as area	Single	Euclidean
(Hegazy and Elbeltagi 1999), (Osman et al. 2003)	GA	Minimize layout cost	Space as area	Single	Euclidean
(Zhang and Wang 2008)	PSO	Minimize layout cost	Space as discrete	Single	Euclidean
(Lam et al. 2009)	ACO, GA	Minimize transportation flow	Space as discrete	Single	Euclidean
(Cheng and O'Connor 1997)	GIS	Minimize traveling distance	Space as area	Single	Euclidean
(Zouein and Tommelein 1999)	CSPA, LP	Minimize layout cost	Space as area	Multiple	Euclidean
(Mahachi 2001), (Mawdesley et al. 2002)	GA	Minimize layout cost	Space as area	Multiple	Euclidean
(Mawdesley and Al-Jibouri 2003)	GA	Minimize layout cost	Space as discrete	Single	Euclidean
(Elbeltagi et al. 2004)	GA	Minimize layout cost, Improve safety	Space as area	Multiple	Euclidean
(Ning et al. 2010a)	ACO	Minimize layout cost, Improve safety	Space as area	Multiple	Euclidean
(Sanad et al. 2008)	GA	Minimize layout cost, Improve safety	Space as area	Multiple	Manhattan
(Wong et al. 2010)	GA, MIP	Minimize layout cost	Space as discrete	Single	Euclidean
(Tommelein and Zouein 1993)	None	Facilitate layout planning	Space as area	Multiple	-
(El-Rayes and Said 2009)	ADP	Minimize layout cost, Improve safety	Space as area	Multiple	Euclidean

Abbreviations:

ACO – Ant Colony Optimization

CSPA – Constraint Satisfaction and Propagation Algorithm

GIS – Geographic Information System

MIP – Mixed Integer Programming

ADP – Approximate Dynamic Programming

GA – Genetic Algorithm

LP – Linear Programming

PSO – Particle Swarm Optimization

Chapter 3: Evolutionary Algorithms

3.1 Genetic Algorithms

As discussed extensively in Chapters 1 and 2, optimization in CSUP involves searching the transportation cost surface until a minimum value is found. Typically, optimization problems can be classified in the following categories.

3.1.1 Trail-and-Error Optimization

Trail-and-error optimization refers to an iterative process in which the variables that have an effect on the output are adjusted without knowing much about the process that produces the output. Several prominent discoveries (for example, discovery and refinement of penicillin as an antibiotic) resulted from the trial-and-error approach to optimization (Haupt and Haupt 2004). On the contrary, optimization based on mathematical formula defines an objective function in function optimization and various mathematical manipulations are performed over the function that leads to the optimal solution.

3.1.2 Temporal Optimization

Optimization problems can also be classified based on whether or not the objective function should have a temporal aspect to it or not. Optimization functions which have temporal aspect associated with it (i.e. output is a function of time) are termed as *dynamic optimization*. Whereas, static optimization problems are those problems whose functions do not have a temporal aspect to it and whose output is independent of time. The static problem is difficult to

solve for the best solutions, but the added dimension of time increases the challenge of solving the dynamic problem (Haupt and Haupt 2004).

3.1.3 Continuous Optimization

Optimization problems can also be distinguished based on their approach in identifying the solutions. Discrete optimization has only a finite number of possible values and searches for optimality within the available set of predetermined solutions. Discrete variable optimization is also known as combinatorial optimization, because the optimum solution consists of a certain combination of variables from the finite pool of all possible variables. On the flipside, continuous optimization has infinite number of possible variable values and does search for optimality considering all possibilities. Thus, typically, optimization where continuous variables are involved is computationally more challenging.

3.1.4 Constrained Optimization

Constrained optimization introduces equality and inequality restrictions to the variables in the cost function. On the contrary, variables in unconstrained optimization problems are allowed to take any value without restrictions. A constrained variable often converts into an unconstrained variable through a transformation of variables (Haupt and Haupt 2004).

3.1.5 Minimum Seeking Optimization

The traditional optimization algorithms (based on calculus) generally try to minimize the cost function by starting from an initial set of variable values. These minimum seekers easily get stuck in local minima but tend to converge fast. In the minimum seeking approach, flow between the variable sets is based on some predetermined sequence of steps. On the other hand, random

methods use some probabilistic calculations to find variable sets. They tend to be slower but have greater success at finding the global minimum (Haupt and Haupt 2004).

The optimization of construction site layouts is a combination of all the categories discussed above. Solving the CSUP problem requires an initial set of solutions which will subsequently be iterated until an optimal layout is identified within the expected threshold values. These initial populations are determined using the trial-and-error approach, in which a set of possible solutions are derived without knowledge of the output and are subsequently optimized with the utilization of minimum seeking optimization strategy. Typically, in most of the construction projects, the available site space varies significantly over the course of the project. This scenario mandates adjustments to the layout of temporary construction facilities to efficiently utilize the available site space during various construction stages. This conceptualization utilizes the temporal aspect of the optimization categories.

CSUP can be conceptualized as i) discrete one-to-one assignments where facilities are assigned to previously defined locations, ii) a continuous domain. While the first definition may be limited in scope, it can be used to effectively model a situation where space is limited and a limited number of locations are available for temporary facilities. The second definition, which theoretically creates an infinite number of layouts, increases the potential locations for locating the facilities exponentially and results in increased optimality. Moreover, the layout planning problem in construction inherently requires that several constraints be imposed in the optimization process (for example, non-overlap constraints). Solving the problem without any optimization constraints restricts the applicability of the optimization algorithm to real-life scenarios. Thus, an efficient CSUP optimization solution requires a constrained optimization strategy.

3.2 Traditional Optimization Algorithms for Minimization

The optimization algorithms can be traced back to the days of Newton, Lagrange, and Cauchy. Cauchy made the first application of the steepest descent method to solve unconstrained optimization problems. The foundations of calculus of variations, dealing with the minimizations of functions, were laid by Bernoulli, Euler, and Lagrange. Over the past several decades, several optimization methods have been developed for finding the minima. The main characteristics of these methods and their weaknesses are discussed in this section.

3.2.1 Exhaustive Search

The traditional approach to optimization iterates through sufficient samples of cost function depending on the problem type to find the global minimum. Usually a list of function values is generated over the sampled variables, and then the list is searched for the minimum value. The exhaustive search goes through the appropriate sampling to produce a precise solution. This approach requires checking an extremely large but finite solution space with the number of combinations of different variable values given by iteration shown in the following equation (Haupt and Haupt 2004).

$$V = \prod_{i=1}^{N_{var}} Q_i \quad \text{E.q. 3.1}$$

where,

V = number of different variable combinations

N_{var} = total number of different variables

Q_i = number of different values that variable i can attain

With the extensive sampling of the variables, this technique does not get trapped in local optimal/minimal solutions and works well for both discrete and continuous conceptualization of the optimization problem. However, the exhaustive sampling results in an extremely long time to find global minimum. Another shortfall of this approach is that the global minimum may be missed due to under-sampling. It is easy to under-sample when the cost function takes a long time to calculate. These shortfalls restrict the applicability of exhaustive search approach to only small number of variables in a limited search space (Haupt and Haupt 2004).

3.2.2 Analytical Optimization

Calculus provides the tools and elegance for finding the minimum of many cost functions. The thought process can be simplified to a single variable for a moment, and then an extremum is found by setting the first derivative of a cost function to zero and solving for the variable value. If the second derivative is greater than zero, the extremum is a minimum, and conversely, if the second derivative is less than zero, the extremum is a maximum. One way to find the extrema of a function of two or more variables is to take the gradient of the function and set it equal to zero, $\nabla f(x,y) = 0$ (Haupt and Haupt 2004).

When compared to exhaustive search approach, the analytical optimization technique is mathematically sound and finds a single minimum solution relatively faster. Yet, it requires a search scheme and continuous functions with analytical derivatives to find the global minimum. Also, the gradient of the cost function serves as the compass head pointing to the steepest downhill path. It works well when the minimum is nearby, but cannot deal well with cliffs or boundaries, where the gradient cannot be calculated. Moreover, if there are too many variables, then it is difficult to find all the extrema, which makes it an unfit strategy to solve the real world

optimization problems. Even though it is impractical, most numerical approaches are based on it. A typical algorithm starts at some random point in the search space, calculates a gradient, and then heads downhill to the bottom. These analytical approaches head downhill fast; however, they often result in local optima instead of the global optima and do not work well with discrete variables (Haupt and Haupt 2004).

3.2.3 Nelder-Mead Downhill Simplex Method

A downhill simplex method which does not require the calculation of derivatives was studied by (Nelder and R.Mead 1965). A simplex is the most elementary geometrical figure that can be formed in dimension N and has $N + 1$ sides (for example, a triangle in two dimensional space). The downhill simplex starts at $N + 1$ points that form the initial simplex. The goal of this method is to move the simplex until it surrounds the minimum, and then to contract the simplex around the minimum until it is within an acceptable error. As this definition gets stuck in local minima, it can be combined with the random search algorithm to find the minimum (Haupt and Haupt 2004).

Though these optimization strategies apply the fundamental principle of heading downhill from an arbitrary starting point, they differ in deciding in which direction to move and how far to move. Successive improvements increase the speed of the downhill algorithms but do not add to the algorithm's ability to find a global minimum instead of a local minimum.

In the recent times, algorithms which generate new points in the vast solution space by applying operators to current points and statistically move toward more optimal places in the solution space have been developed. These include genetic algorithms (Holland 1975), simulated annealing (Kirkpatrick et al. 1983), particle swarm optimization (Parsopoulos and Vrahatis

2002), ant colony optimization (Dorigo and Gambardella 1997), and evolutionary algorithms (Schwefel 1995). These algorithms depend on an intelligent search of large but finite search space using statistical approaches and do not require taking cost function derivatives, which facilitates to deal with discrete variables and not continuous cost functions.

Of these recent algorithms, Genetic Algorithms (GAs) is an optimization and search technique based on the principles of genetics and natural selection. GAs allow a population composed of many individuals to evolve under a specified selection rules to a state that maximizes the fitness. The natural selection of GAs pilot to several generalizations that lead to the view of its origins and workings that includes i) highly diverse organisms ii) complexity level in the organisms is striking iii) several of the organism features have an apparent usefulness. Some of the key advantages of GAs include:

- The ability to optimize continuous or discrete variables
- Does not require derivative information
- Simultaneously searches from a wide sampling of the cost surface
- The ability to deal with a large number of variables
- Well suited for parallel computers
- Optimizes variables with extremely complex cost surface (they can jump out of a local minimum)
- Provides a list of optimum variables, not just a single solution
- May encode the variables so that the optimization is done with the encoded variables
- Works with numerically generated data, experimental data, or analytical functions (Haupt and Haupt 2004).

GAs represent processes in nature that are remarkably successful at optimizing natural phenomena. Algorithms based on GAs have been successfully applied on several complex civil engineering optimization problems. Hence, this research utilizes GAs to develop an algorithm for site utilization planning. Genetic Algorithms are discussed in detail in this chapter.

3.3 Genetic Algorithms

Genetic Algorithms (GAs) is inspired from the theory explaining the origin of species, in which weak and unfit individuals within their environment are faced with extinction by natural selection, and the stronger ones have greater opportunity to pass their genes to future generations via reproduction. In the long run, species carrying the correct combination in their genes become dominant in their population. Sometimes, during the slow process of evolution, random changes may occur in genes. If these changes provide additional advantages in the challenge for survival, new species evolve from the old ones. Unsuccessful changes are eliminated by natural selection.

Genetic Algorithms is a population based meta-heuristic optimization method that is based on the evolutionary process which allows for the survival of the fittest. The main advantage of using GAs is that the algorithm needs a well-defined objective function and a set of constraints and doesn't require any knowledge of the problem space. In GAs, a solution vector is represented as 'Chromosome'. Each chromosome is made of several units called 'Genes'. Each gene controls one or more features of the chromosome. GAs operates with a collection of different chromosomes called 'Population'. The optimization starts with a population of randomly selected individuals (chromosomes). The initial population contains a number of potential solutions (chromosomes) which are combined and mutated using genetic operators to ensure diversity and avoid premature convergence to local optima. From this generation, the

fitter layouts survive while the less fit are eliminated. Thus, as the optimization evolves, the population includes the fitter solutions and eventually converges to a population dominated with a single solution. The fitness of each individual of the population is evaluated based on some predefined objective function. This process is repeated many times till either a predetermined number of iterations are reached or an acceptable solution is found.

Several operators can be implemented in the GAs during the optimization process to produce new solutions, ensure diversity among the population, and avoid premature convergence to local optima. It includes selection, mating (crossover), mutation, inversion, swap, move, rotation, flip2edge, add missing blocks, fix blocks, and aging (Harmanani et al. 2000; Zouein et al. 2002). Due to their high flexibility and diverse applicability, the commonly used operators include Crossover and Mutation (Deb 2009). Flow chart of a binary GAs process is illustrated in Figure 3.1.

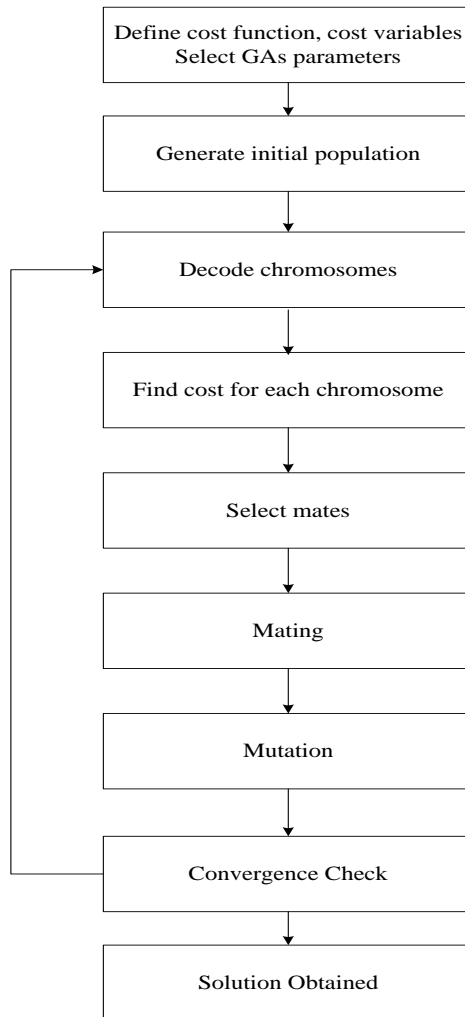


Figure 3.1: Flowchart for binary GAs (Haupt and Haupt 2004)

While the above illustrates the flow of a GAs process, pseudo code of a common GAs optimization process is provided in the section below (Deb 2009).

Step 1: Set $t = 1$. Randomly generate N solutions to form the first/initial population (P_t).

Evaluate the fitness of solutions in P_t .

Step 2: Generate an offspring population Q_t (Crossover) as follows:

2.1. Choose two solutions x and y from P_t based on the fitness values.

2.2. Using a crossover operator, generate offspring and add them to Q_t .

Step 3: Change the attributes of randomly selected solutions in Q_t (Mutation) with a predefined mutation rate.

Step 4: Evaluate and assign a fitness value to each solution in Q_t based on its objective function value and infeasibility.

Step 5: Select N solutions from Q_t based on their fitness and copy them to P_{t+1} .

Step 6: If the stopping criterion is satisfied, terminate the search and return to the current population, else, set $t = t+1$, and iterate the process from Step 2.

3.4 Genetic Operations

3.4.1 Selection

GAs start with a group of chromosomes called population (N_{pop}), of which the chromosomes with worst objective values will be discarded during the iterative process of accomplishing an optimal solution. In this survival of the fittest theory, to retain the best solutions in the mating pool (N_{keep}) and discard the worst solutions, GAs uses the ‘Selection’ operator which chooses two chromosomes from the mating pool (N_{keep}) to produce two new offspring. The selection process is iterated until an offspring population equal to the population of discarded chromosomes is born. Several selection techniques are available to perform this pairing process and are discussed in the following section.

i) Pairing from top to bottom:

This selection technique starts with pairing the top chromosomes from the population pool (N_{pop}) until the top N_{keep} chromosomes are selected for mating. In this top down approach,

the odd rows are paired with the even rows, in which the mother has the odd row numbers (1, 3, 5,...) from the population matrix and the father has the even row numbers (2, 4, 6,...) from the population matrix (Haupt and Haupt 2004).

ii) *Random pairing:*

In this approach, instead of the top down technique for pairing, the chromosomes to be paired are randomly identified from the population matrix (N_{pop}). For example, the row numbers of the parents to be paired from the population pool can be identified as,

$$\text{mother chromosome} = \text{int}[N_{\text{keep}} * \text{random}(1, N_{\text{keep}})]$$

$$\text{father chromosome} = \text{int}[N_{\text{keep}} * \text{random}(1, N_{\text{keep}})]$$

iii) *Weighted random pairing:*

In this selection technique, probabilities (inversely proportional to the objective value) are assigned to the chromosomes in the mating pool. A chromosome with lowest objective value has the highest probability of mating, whereas the chromosome with highest objective value has the lowest probability of mating. A random number determines which chromosome is selected. This type of weighting is often referred as roulette wheel weighting and there are two types in it (Haupt and Haupt 2004).

a) *Rank Weighting:* This approach is problem independent and finds the probability from the rank, n , of the chromosome:

$$P_n = \frac{N_{\text{keep}} - n + 1}{\sum_{n=1}^{N_{\text{keep}}} n} = \frac{4 - n + 1}{1 + 2 + 3 + 4} = \frac{5 - n}{10} \quad \text{Eq. 3.2 (Haupt and Haupt 2004)}$$

For example, consider the example provided in Table 3.1 which shows the results for $N_{\text{keep}} = 4$ chromosomes. The cumulative probabilities listed in column 4 are used in selecting the chromosome. A random number between zero and one is generated. Starting at the top of the list,

the first chromosome with a cumulative probability that is greater than the random number is selected for the mating pool. For example, if the random number (r) is 0.577, then r lies between 0.4 and 0.7, so chromosome2 is selected. If a chromosome is paired with itself, there are several alternatives. First, let it go. It means that there are three of these chromosomes in the next generation.

Table 3.1: Rank Weighting (Haupt and Haupt 2004)

n	Chromosome	P_n	$\sum_{i=1}^{i=n} P_i$
1	00110010001100	0.4	0.4
2	11101100000001	0.3	0.7
3	00101111001000	0.2	0.9
4	00101111000110	0.1	1.0

Second, randomly pick another chromosome. The randomness in this approach is more indicative of nature. Third, pick another chromosome using the same weighting technique. In this approach, small populations have a high probability of selecting the same chromosome and the probabilities only have to be calculated once.

b) *Cost Weighting*: In this approach, the probability of selection is calculated based on the cost of the chromosome, whereas the rank weighting approach is based on a chromosome's rank in the population. A normalized cost is calculated for each chromosome by subtracting the lowest cost of the discarded chromosomes ($C_{N_{\text{keep}+1}}$) from the cost of all the chromosomes in the mating pool.

$$C_n = c_n - C_{N_{\text{keep}+1}} \quad \text{Eq. 3.3 (Haupt and Haupt 2004)}$$

Subtracting $c_{N_{\text{keep}}+1}$ ensures all the costs are negative. This approach tends to weight the top chromosome more when there is a large spread in the cost between the top and bottom chromosome. On the other hand, it tends to weight the chromosomes evenly when all the chromosomes have approximately the same cost. The same issues as discussed in rank weighting apply if a chromosome is selected to mate with itself. In this technique, the probabilities must be recalculated for each generation (Haupt and Haupt 2004).

iv) Tournament selection:

This approach closely mimics mating competition in nature in which a random small subset of the chromosomes is picked from the mating pool, and the chromosome with the lowest cost in the subset becomes parent. The tournament repeats for every parent needed. Thresholding and tournament selection make a nice pair, because the population never needs to be sorted. In large populations, as sorting of population becomes time-consuming, this technique works best for larger population sizes (Haupt and Haupt 2004).

Each of the selection techniques results in a different set of parents, and the composition of the next generation also varies for each selection scheme. Roulette wheel and tournament selection are standard selection techniques in GAs (Haupt and Haupt 2004).

3.4.2 Mating

Mating is the process of creating one or more offspring from the parents selected in the pairing process. The most common form of mating process is to crossover the involved parents to produce offspring (Figure 3.2). A crossover point is randomly chosen from the length of the chromosome and the parent1 passes its solution to the left of that crossover point to offspring1. Similarly, parent2 passes its solution to the left of the same crossover point to offspring2. In a

like manner, solution from the right of parent1 and parent2 from the same crossover point goes to offspring2 and offspring1 respectively. This mating process is termed as single-point crossover (Haupt and Haupt 2004).

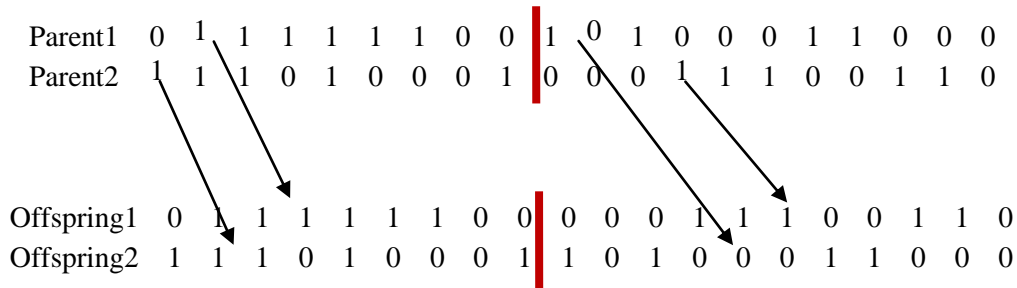


Figure 3.2: Single-Point Crossover example

As the parents selected from the selection techniques are with preference towards better fitness (better objective values), the offspring generated from the parents in the mating process are expected to inherit good genes from the parents which make them fitter. By iteratively applying the mating/crossover operation, genes of good chromosomes are expected to appear more frequently in the population, eventually leading to convergence to an overall good solution (Deb 2009).

3.4.3 Mutation

Mutation introduces random changes into the characteristics of chromosomes at a certain percentage. Mutation is the second way the GAs explore a cost surface. It can introduce traits not in the original population and keeps the GAs from converging too fast before sampling the entire cost surface. Mutation points are randomly selected from the ($N_{pop} * \text{Chromosome length}$) total number of bits in the population matrix, and a single point binary mutation changes a 1 to a 0, and vice versa (equation 3.4). Increasing the number of mutations increase the algorithm's

freedom to search outside the current region of variable space. It also tends to distract the algorithm from converging on a popular solution. As best solutions are designated as elite solutions and destined to propagate unchanged, mutation is typically not allowed on best solutions.

$$\# \text{ of mutations} = \mu * (N_{\text{pop}}-1) * \text{length of chromosome}$$

$$\text{Mutate chromosome} = (\mu * N_{\text{pop}}) + \text{Random}(1, N_{\text{pop}})$$

$$\text{Mutate gene} = \text{Random}(\mu, 1) * N_{\text{pop}}$$

where, μ = mutation probability/rate



Eq. 3.4

In typical GAs implementations, the mutation rate (probability of changing the properties of a gene) is very small and depends on the length of the chromosome. Therefore, the new chromosome produced by mutation will not be very different from the original one (Deb 2009). This random mutation operation shall be implemented through several operators which include swap (exchanges the positions of two randomly selected genes in a chromosome), inversion (flips the attribute of a gene), move (moves the attribute of a randomly selected gene), rotation (rotates the attribute of a gene in either clockwise or anticlockwise direction), and aging (destroys a chromosome whose fitness did not change after a predefined number of iterations) (Harmanani et al. 2000).

3.4.4 Convergence

The number of generations that evolve depends on whether an acceptable solution is reached or a set number of iterations are exceeded. After a while, all the chromosomes and

associated costs (objective value) would become the same if it were not for mutations. At this point the algorithm should be stopped.

3.5 Summary

The layout planning problem is a hybrid of several categories of optimization including temporal optimization, optimization in a continuous domain, constrained and minimum seeking optimization. The traditional optimization algorithms for minimization like exhaustive search, analytical optimization, and simplex methods often seek to identify locally minimal solutions and have limited applicability in solving complex real-time optimization problems. Several optimization strategies have been developed in the recent times and are contemporarily used in engineering optimization problems. Genetic Algorithms (GAs), one of the commonly used contemporary algorithm, is identified as a robust optimization technique that seek to improve performance by sampling areas of the parameters space that are likely to lead better solutions (Goldberg 1989), (Holland 1992). Also, it has the potential to move randomly from one feasible solution to another without getting into to local optima, in which other algorithms often get trapped (Li and Love 1998), and it has the highest potential to efficiently solve several engineering and construction management problems (Elbeltagi et al. 2004).

Chapter 4: Site Utilization Planning Algorithm

4.1 Introduction

Site layout planning in construction is a complex optimization problem, in which the layout of temporary facilities is unique in every construction project due to the changing requirements on account of different site configurations and project requirements. A significant amount of research effort has been made by academic research community over the last two decades to develop a satisfactory optimization algorithm for layout of temporary facilities at construction sites. Building on the existing research, a new and much improved site layout tool has been developed in this research study. This tool overcomes some of the critical shortcomings of the past research (e.g., distance measurement) and provides innovative means to model some of the constraints commonly observed on construction projects. The open and modular architecture of this tool also makes it an ideal platform for future improvements. The important algorithms used in this tool are discussed in this chapter.

4.2 Algorithms

4.2.1 Distance Measurement Algorithm

The core objective of optimization in facility layout planning is to minimize the cost of transportation of materials, workers, and equipment between temporary facilities. In other words, the objective of optimization is essentially to minimize the distance travelled. Hence it is imperative that the actual distance travelled between facilities be measured accurately and realistically. Almost all of the optimization algorithms found in the literature measure distance

between facilities using the Euclidean formula. It is quite possible that the earlier studies have adopted the Euclidean approximation to simplify the complexity of the problem. This may not be accurate in projects where such travel is not possible and is likely to produce sub-optimal layouts (Sanad et al. 2008).

Measuring the Euclidean distance between the temporary facilities uses the Pythagorean formula, which measures the diagonal distance between the points without considering any obstructions. For example, consider the Figure 4.1. In a two dimensional space, the Euclidean distance between the two points $p(x_1, y_1)$ and $q(x_2, y_2)$ is measured using the formula $d(p,q) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$. As illustrated in Figure 4.1, the distance between the points P_1 and P_2 is measured as the diagonal distance between the points without taking into account of the building between them. But such travel, between two temporary facilities, may not be possible in most construction projects due to the presence of the permanent structure or the excavation for the structure. Thus, measuring the Euclidean distance might not represent the actual distance travelled on a construction site.

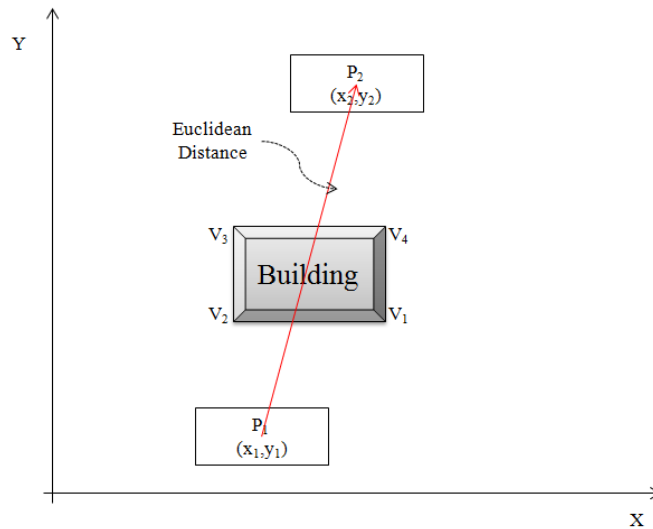


Figure 4.1 Euclidean Distance Measurement Approach

To overcome this shortcoming, many researchers have proposed the use of the Manhattan distance, which measures the sum of absolute differences of the coordinates of the points under consideration. Manhattan distance, also known as Rectilinear Distance, between two points refers to the distance traveled between two points on a grid of streets in a downtown area (e.g., Manhattan). Manhattan distance between any two given points (x_1, y_1) and (x_2, y_2) is measured using the formula:

$$d(p,q) = (|y_2-y_1| + |x_2-x_1|) \quad \text{Eq. 4.1}$$

While this type of distance measurement would provide a better estimate of the actual distance travelled (as compared to Euclidean distance), in some cases, even measuring the Manhattan distance might not represent the actual travel distance between the facilities. For example, consider the Figure 4.2. In this case, measuring the Manhattan distance between the points P_1 and P_2 represents the actual travel distance. But in this scenario the obstruction (i.e., building) does not interfere with the range of X coordinates of the points P_1 and P_2 .

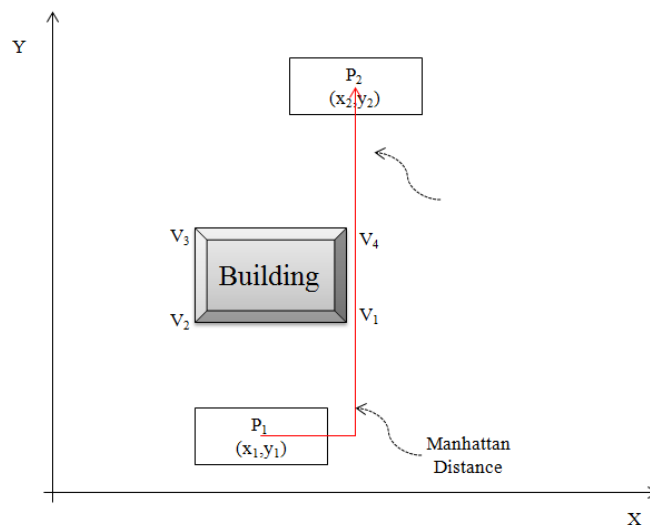


Figure 4.2 Manhattan Distance Measurement Approach (applicable scenario)

Now consider the Figure 4.3 where the obstruction (building) interferes with the range of X coordinates of the points P_1 and P_2 . In this scenario, as illustrated in Figure 4.3, measuring the pure Manhattan distance between the points P_1 and P_2 also does not represent the actual travel distance between the points. Thus, in solving the site utilization planning problem in construction, measuring the pure Manhattan distance among the facilities might not be accurate as it does not represent the accurate distance in some scenarios.

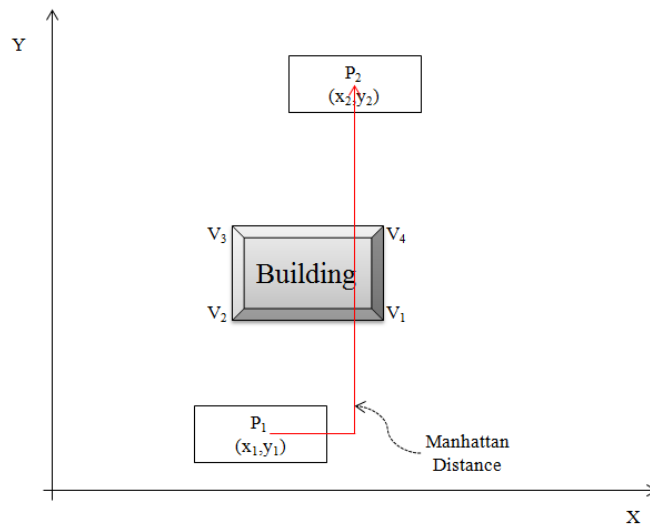


Figure 4.3 Manhattan Distance Measurement Approach (inapplicable scenario)

Hence, to overcome these shortcomings of the currently used Euclidean and Manhattan distance measurement approaches, an improvised distance measurement algorithm is developed in this study which would represent the actual travel distance among the construction facilities in all scenarios. For example, for the scenario illustrated in Figure 4.4, the distance measurement approach works as follows.

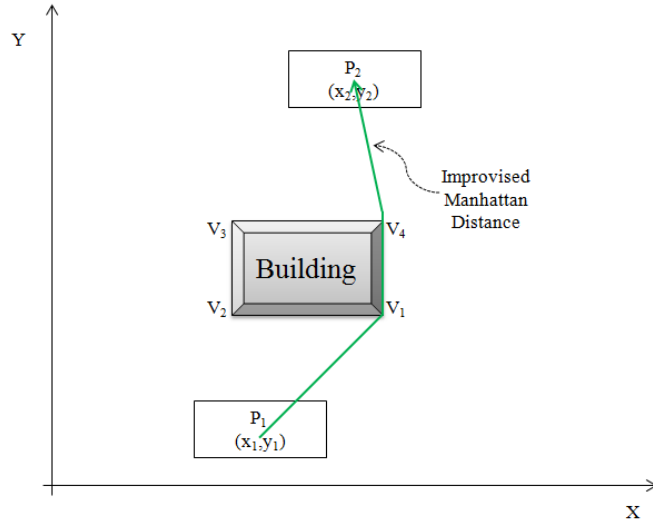


Figure 4.4 Improved Distance Measurement Approach

The algorithm first identifies the vertices (of any obstruction) that could be accessed directly from point p_1 without passing through any obstruction (say, accessible vertices). For this case, the accessible vertices would be V_1 and V_2 . Then, for each of the identified accessible vertices, it measures the distance to access point p_2 from that vertex through other vertices in both clockwise and anticlockwise directions and identifies the minimum distance. For example, for the case under discussion, to measure the distance between the points p_1 and p_2 , distance will be measured for all the below listed feasible ways and the minimum distance ($P_1 \rightarrow V_1 \rightarrow V_4 \rightarrow P_2$) will be identified. Thus, the distance measurement models the likely movement and measures the associated travel distance and thus improves the optimality of the layouts.

- i) $P_1 \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow P_2$ (clockwise from V_1)
- ii) $P_1 \rightarrow V_1 \rightarrow V_4 \rightarrow P_2$ (anticlockwise from V_1) – **Optimal route**
- iii) $P_1 \rightarrow V_2 \rightarrow V_3 \rightarrow P_2$ (clockwise from V_2)
- iv) $P_1 \rightarrow V_2 \rightarrow V_1 \rightarrow V_4 \rightarrow P_2$ (anticlockwise from V_2)

Thus, the distance measurement algorithm measures the actual travel distance and aids to accomplish true optimal layouts.

In the developed optimization model, the site area is broken down into smaller grids of size provided by the user (through user interface) and the available grids are identified first. The distance measurement algorithm is then invoked to measure the distance between the centroid of each of the available grid. For example, consider the Figure 4.5. The site area is first divided into smaller grids of size 25'x25', where 25' is the preferred grid size provided by the user. The available grids (white colored grids) are identified and then the distance measurement algorithm is invoked which measures the distance among the centroid of each of the available grid.

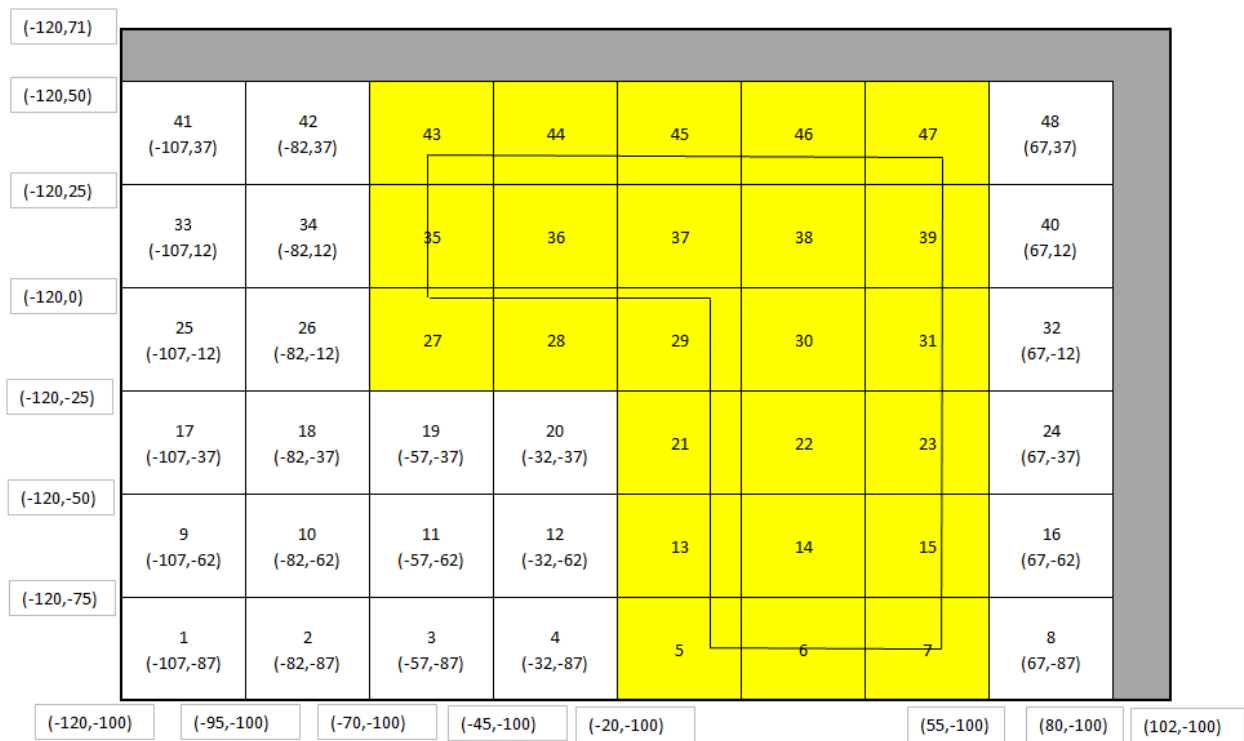


Figure 4.5 Distance Measurement Strategy

Pseudo code for distance measurement algorithm is provided in the following section.

Distance Measurement Pseudo Algorithm

Store the cells occupied by the corners of the building in an array variable 'BCC(n)' (Building Corner Cells)
Store the x coordinate of point1 in 'x1' and x coordinate of point2 in 'x2' (p1, p2 - points between which the distance need to be measured)
Store the y coordinate of point1 in 'y1' and x coordinate of point2 in 'y2'

Function Distance_Measurement ([x1,y1], [x2,y2])

```
Define a variable MinDistance
Call the function Has_Block([x1,y1], [x2,y2])
If Has_Block = 'False' Then
    MinDistance = sqrt[(x2-x1)^2 + (y2-y1)^2]
Else
    For i = 0 to n-1 (n = number of vertices)
        Call the function Has_Block([x1,y1], [BCC(i)])
        If Has_Block = 'True' Then
            i = i+1
        Else
            Call the function clockwise ([x1,y1], [BCC(i)], i)
            Call the function anticlockwise ([x1,y1], [BCC(i)], i)
            MeasuredDistance = Minimum(TotalDistanceC, TotalDistanceAC)
            If MinDistance = 0 then
                MinDistance = MeasuredDistance
            Else
                If MeasuredDistance < MinDistance then MinDistance = MeasuredDistance Else End
            End
        End
        i = i+1
    Loop
End function
```

Function Has_Block ([x1,y1], [x2,y2])

```
Store the cells occupied by the building in an array variable 'BC()' (Building Cells)
Define an array variable Path(p)
Set x = x1
For x = x1 to x2
    y = [(x-x1) * (y1-y2)] + y1(x1-x2) / (x1-x2)
    y = round(y,0)
    Path(p) = (x,y)
    p = p+1
    x = x+1
Loop
For i = 0 to p-1
    If path(i) in BC() Then
        Has_Block = 'True'
        Exit Loop
    Else
        Has_Block = 'False'
        i = i+1
    End
Loop
Empty Path(p)
End function
```

Function clockwise ([x1,y1], [BCC(i)], i)

```
Define a variable PC (Present Cell) to store the x,y coordinate of the current vertex
Define an array Distance(v)
Define a variable newi
Define a variable TotalDistanceC
newi = i
Distance(v) = sqrt[(BCCnewix-x1)^2 + (BCCnewiy-y1)^2]
PC = BCC(newi)
v = v+1
```

```

Do until Has_Block[PC, (x2,y2)] = 'False'
  If new_i = 0 then new_i = (n-1) else new_i = (new_i-1) End
  Distance(v) = sqrt[(BCCnewix-PCx)^2 + (BCCnewiy-PCy)^2]
  PC = BCC(new_i)
  v = v+1
Loop

Distance(v) = sqrt[(PCx-x2)^2 + (PCy-y2)^2]
TotalDistanceC = sum[distance(v)]
End Function

Function anticlockwise ([x1,y1], [BCC(i)], i)
Define a variable PC (Present Cell) to store the x,y coordinate of the current vertex
Define an array Distance(v)
Define a variable new_i
Define a variable TotalDistanceAC
new_i = i
Distance(v) = sqrt[(BCCnewix-x1)^2 + (BCCnewiy-y1)^2]
PC = BCC(new_i)
v = v+1

Do until Has_Block[PC, (x2,y2)] = 'False'
  If new_i = n-1 then new_i = 0 else new_i = (new_i+1) End
  Distance(v) = sqrt[(BCCnewix-PCx)^2 + (BCCnewiy-PCy)^2]
  PC = BCC(new_i)
  v = v+1
Loop

Distance(v) = sqrt[(PCx-x2)^2 + (PCy-y2)^2]
TotalDistanceAC = sum[distance(v)]
End Function

```

4.2.2 Optimization Algorithms

As discussed in Chapter 2 and 3, the optimization of layouts of temporary facilities in construction projects is a complex, computationally intensive problem which can't be solved using analytical optimization methods on account of the vast size of the solution space. A number of global optimization algorithms inspired by natural phenomena (e.g. ant colony optimization, genetic algorithms) have been proposed as an effective alternate solution to exploring an extremely vast solution space to identify a good layout. This research study has adopted Genetic Algorithms (GAs) as an optimization tool because it has been used and proven effective in solving complex engineering optimization problems. Furthermore, GAs have the capability to explore the solution space without getting trapped in a local optimum. Genetic Algorithms essentially provide an intelligent bias to random exploration of the solution space to identify a good layout. The classic unconstrained GAs has limited applicability in modeling the real life, temporally dynamic requirements of construction sites. This section will provide the details of the essential elements of the algorithms used in the optimization tool. The optimization tool developed in this study is designed to model the constraints typically observed at construction sites. This tool consists of the following important algorithms:

- An algorithm to conceptualize the site and generate initial layouts
- An algorithm to evaluate the efficacy of each layout in meeting a predefined fitness objective
- An algorithm to perform genetic operations of crossover (to improve the quality of solutions) and mutation (to enable the algorithm to escape local optimum and search the solution space more effectively through diversification)

4.2.2.1 Initial Solutions

As discussed in chapter 3, the GAs starts with a population of randomly generated initial potential solutions (ie., chromosomes), which then undergo the optimization process through the genetic operators (e.g., Crossover and Mutation) for several generations and the algorithm eventually converges to a population dominated with a single solution. Several of the earlier literature studies formulate the layout planning problem as discrete one-to-one assignments, where facilities are assigned to previously defined locations. In this conceptualization, as the available positions are previously identified, it eases the process of generating the initial population and leaves the initial population generation algorithm as just randomly assigning the previously identified locations to the temporary facilities to be laid out without imposing any constraints. This definition (one-to-one assignment problem where ‘n’ facilities are to be located in ‘n’ predefined location) is limited in scope and it can only be used to effectively model a situation where space is limited and the number of locations available for locating the temporary facilities is predetermined. But in reality, the available locations are not discrete and are not predetermined as-well.

The site area can be conceptualized as a continuous domain, theoretically creating infinite layouts. While this conceptualization can potentially result in evaluation of more solutions, it has limited potential benefits when it comes to implementing the layout plan at the site. For example, if the optimization process yields a layout that includes a facility which should be located exactly 102 feet and 11.78 inches from the left corner of the site, it may not be practical or possible to layout the temporary facility with such precision. Hence, this study conceptualizes the site as a discrete domain divided into grids.

The algorithm starts with dividing the entire site space into smaller grids of size provided by the user. The vertices and the associated grids are numbered starting at the left bottom corner of the site. It then identifies the available grids (girds which are not occupied by any building/utilities and are available for temporary facilities allocation) and unavailable grids (girds which are occupied by the building/utilities and are not available for temporary facilities allocation), which are mutually exclusive. Once the available and unavailable grids are identified, the distance measurement algorithm is used to measure the actual travel distance among each of the identified available grids and is stored in an xml file.

This study limits the shape of the facilities to rectangular or square. The algorithm selects a random grid number from the available grids to position the first facility that needs to be stored onsite. Once a grid/node number is chosen, there are 8 possible different alignments (4 horizontal, 4 vertical) by which a rectangular facility can be placed, illustrated in Figure 4.6.a and 4.6.b.

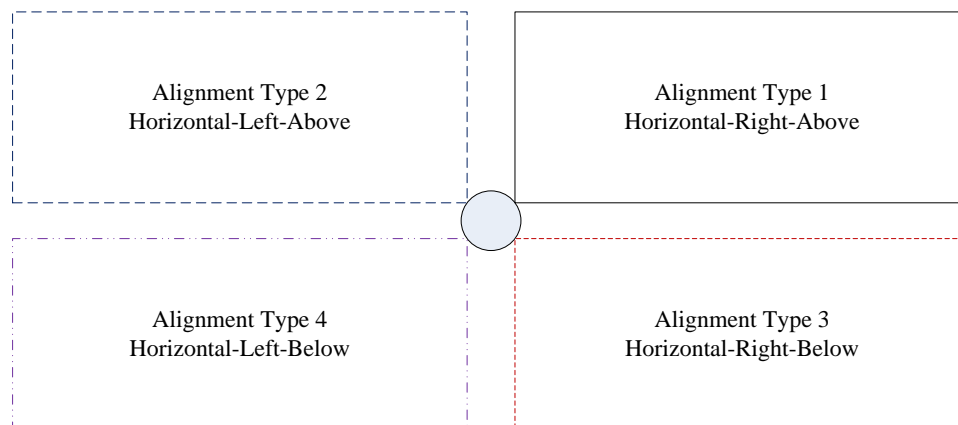


Figure 4.6.a – Horizontal alignment

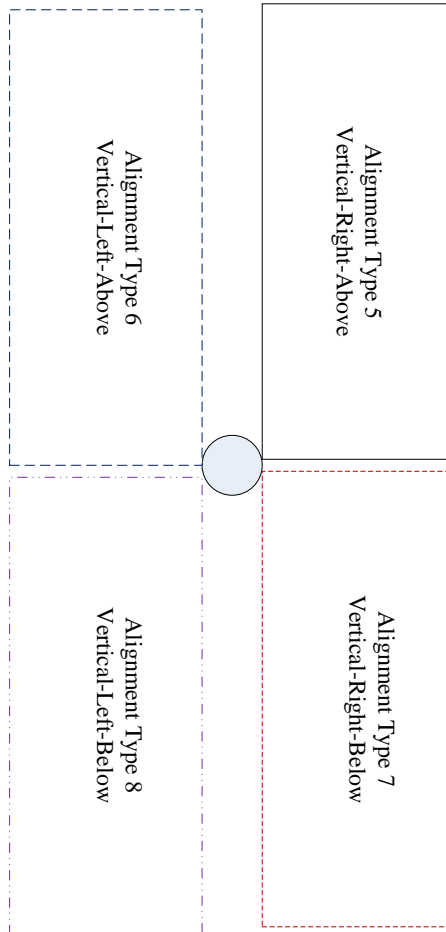


Figure 4.6.b – Vertical alignment

Of these eight possible alignment types, the algorithm randomly chooses an alignment type for the selected facility and identifies the grid numbers required for that facility. If the required grids for a facility are not available, the algorithm regenerates a different random grid number and identifies the required grids. If the algorithm is not able to find a location to position a facility for more than 100 iterations, it is supposed that there is not a feasible solution and the user is informed with a message. This process is repeated until the location for all the temporary facilities to be positioned onsite are identified. As the literature review on GAs inferred that an initial population of 100 (100 randomly generated solutions) can lead to better optimal solutions (Li and Love 1998), the algorithm identifies 100 different feasible solutions to position the

temporary facilities and assigns to the initial population pool, which will subsequently be optimized through the GAs process. An example of how the algorithm identifies the required grids for a facility is illustrated below.

20	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

Figure 4.7 – Illustrative example

Consider a site space with 25 continuously available grids (Figure 4.7) in which a rectangular facility of size 30x15 needs to be positioned. Assume that the user has provided the preferred grid size as 10x10. Hence, to position this facility it requires 3 grids (30/10) along the length side and 2 grids (round of 15/10) along the width side. Thus, the facility totally requires a sum of 6 grids (3x2). Assume that grid number 12 and alignment type 1 (horizontal-right-above) has been randomly chosen by the algorithm to position the facility. For this random grid number and alignment type, the algorithm will identify the grids 12, 13, 14, 17, 18, 19 as the required grids to position the facility.

Layout solutions/chromosomes generated from the initial population algorithms will have the onsite stored facility numbers associated with their corresponding starting grid index number and alignment type. For example, consider a case where six temporary construction facilities are used in a construction project of which four are stored onsite (facilities 1, 3, 4, 5) and two in offsite (facilities 2, 6). Assume that Table 4.1 represents one of the layout

solutions/chromosomes. The solution/chromosome reads as; facility 1 is located with starting grid as 25 and alignment type as 7, facility 3 is located with starting grid as 100 and alignment type as 6, facility 4 is located with starting grid as 5 and alignment type as 3, facility 5 is located with starting grid as 215 and alignment type as 4. Since facilities 2 and 6 are stored in offsite they are not represented in the solution chromosome.

Table 4.1: Representation of Solution Chromosome

Facility #	1	3	4	5
Alignment	7	6	3	4
Grid Position	25	100	5	215

Pseudo code for initial population algorithm is provided in the following section.

Initial Population Generation Pseudo Algorithm

Store the #'s of the grids occupied by the building (with safety boundary) and unusable areas in an array variable 'UG()' (Unusable Grids)

Store the #'s of the available grids in the site area in an array variable 'AG()' (Available Grids)

Store the # of rows available in the entire grid split-up in a variable 'Rows'

Store the # of columns available in the entire grid split-up in a variable 'Columns'

Store the grid size provided by the user (in UI) in a variable 'Grid_Size'

Function Initial Population ()

Define an array variable Solutions(i) // To store the different layout solutions

For i = 1 to 100

 Define a variable 'Temp_AG()' // To store the available grids locally for computations

 Define a variable 'Temp_UG()' // To store the unavailable grids locally for computations

 Temp_AG() = AG()

 Temp_UG() = UG()

 For j = 1 to n (n=number of facilities provided by the user)

 Define a variable 'Position'

Position = Pick a random # from Temp_AG()

 Define a variable 'Horz_Grid_Reqd' //To store the number of horizontal grids required

 Define a variable 'Vert_Grid_Reqd' //To store the number of vertical grids required

 Define a variable 'Reqd_Grids' //To store grids #'s occupied by a facility

 Horz_Grid_Reqd = int(Length(j)/Grid_Size)

 Vert_Grid_Reqd = int(Breadth(j)/Grid_Size)

 Define a variable Alignment

 Alignment = int(random[1, 8])

 Case When Alignment =1 // For the case Horizontal, Right, and Above

 Define a variable Vert_Position

 Reqd_Grids = Null //To empty the variable before starting the assignment for next facility

 For m=0 to Vert_Grid_Reqd-1

 Vert_Position = Position + (m * Columns)

 For n=1 to Horz_Grid_Reqd

 If Vert_Position is not available in the array variable Temp_AG() then

 Exit and Goto Step I (highlighted in Green+Bold) **without incrementing facility #**

 Else

 Reqd_Grids = Reqd_Grids + “,” + Vert_Position

 Vert_Position = Vert_Position +1

 End

 n=n+1

 Loop

 m=m+1

 Loop

 Grid #'s available in the variable 'Reqd_Grids' should be removed from the array variable Temp_AG() and added to the array variable Temp_UG()

 When Alignment =2 // For the case Horizontal, Left, and Above

 Define a variable Vert_Position

 Reqd_Grids = Null //To empty the variable before starting the assignment for next facility

 For m=0 to Vert_Grid_Reqd-1

 Vert_Position = Position + (m * Columns)

 For n=1 to Horz_Grid_Reqd

 If Vert_Position is not available in the array variable Temp_AG() then

 Exit and Goto Step I (highlighted in Green+Bold) **without incrementing facility #**

 Else

 Reqd_Grids = Reqd_Grids + “,” + Vert_Position

 Vert_Position = Vert_Position -1

 End

 n=n+1

 Loop

 m=m+1

 Loop

 Grid #'s available in the variable 'Reqd_Grids' should be removed from the array variable Temp_AG() and added to the array variable Temp_UG()

When Alignment =3 // *For the case Horizontal, Right, and Below*
 Define a variable Vert_Position
 Reqd_Grids = Null // *To empty the variable before starting the assignment for next facility*
 For m=0 to Vert_Grid_Reqd-1
 Vert_Position = Position - (m * Columns)
 For n=1 to Horz_Grid_Reqd
 If Vert_Position is not available in the array variable Temp_AG() then
 Exit and Goto Step I (highlighted in Green+Bold) **without incrementing facility #**
 Else
 Reqd_Grids = Reqd_Grids + “,” + Vert_Position
 Vert_Position = Vert_Position + 1
 End
 n=n+1
 Loop
 m=m+1
 Loop
 Grid #'s available in the variable ‘Reqd_Grids’ should be removed from the array variable Temp_AG()
 and added to the array variable Temp_UG()

When Alignment =4 // *For the case Horizontal, Left, and Below*
 Define a variable Vert_Position
 Reqd_Grids = Null // *To empty the variable before starting the assignment for next facility*
 For m=0 to Vert_Grid_Reqd-1
 Vert_Position = Position - (m * Columns)
 For n=1 to Horz_Grid_Reqd
 If Vert_Position is not available in the array variable Temp_AG() then
 Exit and Goto Step I (highlighted in Green+Bold) **without incrementing facility #**
 Else
 Reqd_Grids = Reqd_Grids + “,” + Vert_Position
 Vert_Position = Vert_Position - 1
 End
 n=n+1
 Loop
 m=m+1
 Loop
 Grid #'s available in the variable ‘Reqd_Grids’ should be removed from the array variable Temp_AG()
 and added to the array variable Temp_UG()

When Alignment =5 // *For the case Vertical, Right, and Above*
 Define a variable Vert_Position
 Reqd_Grids = Null // *To empty the variable before starting the assignment for next facility*
 For m=0 to Horz_Grid_Reqd-1
 Vert_Position = Position + (m * Columns)
 For n=1 to Vert_Grid_Reqd
 If Vert_Position is not available in the array variable Temp_AG() then
 Exit and Goto Step I (highlighted in Green+Bold) **without incrementing facility #**
 Else
 Reqd_Grids = Reqd_Grids + “,” + Vert_Position
 Vert_Position = Vert_Position + 1
 End
 n=n+1
 Loop
 m=m+1
 Loop
 Grid #'s available in the variable ‘Reqd_Grids’ should be removed from the array variable Temp_AG()
 and added to the array variable Temp_UG()

When Alignment =6 // *For the case Vertical, Left, and Above*
 Define a variable Vert_Position
 Reqd_Grids = Null // *To empty the variable before starting the assignment for next facility*

```

For m=0 to Horz_Grid_Reqd -1
  Vert_Position = Position + (m * Columns)
  For n=1 to Vert_Grid_Reqd
    If Vert_Position is not available in the array variable Temp_AG() then
      Exit and Goto Step I (highlighted in Green+Bold) without incrementing facility #
    Else
      Reqd_Grids = Reqd_Grids + “,” + Vert_Position
      Vert_Position = Vert_Position -1
    End
  n=n+1
  Loop
m=m+1
Loop

```

Grid #'s available in the variable 'Reqd_Grids' should be removed from the array variable Temp_AG() and added to the array variable Temp_UG()

When Alignment =7 // *For the case Vertical, Right, and Below*

```

Define a variable Vert_Position
Reqd_Grids = Null //To empty the variable before starting the assignment for next facility
For m=0 to Horz_Grid_Reqd -1
  Vert_Position = Position - (m * Columns)
  For n=1 to Vert_Grid_Reqd
    If Vert_Position is not available in the array variable Temp_AG() then
      Exit and Goto Step I (highlighted in Green+Bold) without incrementing facility #
    Else
      Reqd_Grids = Reqd_Grids + “,” + Vert_Position
      Vert_Position = Vert_Position +1
    End
  n=n+1
  Loop
m=m+1
Loop

```

Grid #'s available in the variable 'Reqd_Grids' should be removed from the array variable Temp_AG() and added to the array variable Temp_UG()

When Alignment =8 // *For the case Vertical, Left, and Below*

```

Define a variable Vert_Position
Reqd_Grids = Null //To empty the variable before starting the assignment for next facility
For m=0 to Horz_Grid_Reqd -1
  Vert_Position = Position - (m * Columns)
  For n=1 to Vert_Grid_Reqd
    If Vert_Position is not available in the array variable Temp_AG() then
      Exit and Goto Step I (highlighted in Green+Bold) without incrementing facility #
    Else
      Reqd_Grids = Reqd_Grids + “,” + Vert_Position
      Vert_Position = Vert_Position -1
    End
  n=n+1
  Loop
m=m+1
Loop

```

Grid #'s available in the variable 'Reqd_Grids' should be removed from the array variable Temp_AG() and added to the array variable Temp_UG()

End Case

Solutions(i) = Solutions(i) + (Position, Alignment)

```

j = j+1
Loop

```

```

i = i+1

```

```

Loop

```

End Function // Initial Population

4.2.2.2 Solutions Evaluation

In order to converge from a population dominated with diverse solutions to a single optimal solution, the initially generated random solutions have to be evaluated and their objective function has to be refined through successive iterations of genetic operations. In the majority of research papers reviewed in chapter 2, the objective function designed to evaluate the fitness of each solution is defined as the sum of the products of distances between the facilities, cost of transportation between the facilities, and preferred proximity weight between the facilities, to which a penalty value for constraint violations, if any, is added (Osman and Georgy 2005). While using actual transportation costs in the objective function has the clear objective of minimizing the total transportation costs among the temporary facilities, obtaining accurate values for the actual inter-facility transportation costs can become quite difficult, especially during project planning stages. This limitation promotes the use of proximity weights instead of transportation costs (Osman et al. 2003).

Osman et al. (2003) has presented a nice summary of the objective functions used in several earlier studies. Table 4.2 provides the summary studied by (Osman et al. 2003).

Table 4.2: Objective functions used in literature (Osman et al. 2003)

S. No.	Pseudo model of objective function	Study
1	To minimize the frequency of trips made by construction personnel	(Li and Love 1998)
2	To minimize the total transportation costs of resources between facilities	(C.M. Tam 2001; S.O. Cheung 2002)
3	To minimize the cost of facility construction and the interactive cost between the facilities	(Yeh 1995)
4	To minimize the total transportation costs of resources between facilities (<i>presented through a system of proximity weights associated with an exponential scale</i>)	(Hegazy and Elbeltagi 1999)
5	To minimize the total transportation costs of resources between facilities and the total relocation costs (<i>presented through a system of proximity weights and relocation weights</i>)	(Zouein and Tommelein 1999)

Objective function defined in this study (E.q. 4.1) has both cost and proximity parameters integrated in it. This facilitates the flexibility of utilizing the cost parameters in project cases where the data is available, and ignoring the cost aspect by defaulting to 1, where the data is not available.

$$\text{Objective Function} = \sum_{i=1}^{i=n-1} \sum_{j=i+1}^{j=n} d_{ij} * c_{ij} * w_{ij} + \text{Penalty value} \quad (\text{Eq. 4.1})$$

Where, n = number of temporary construction facilities to be laid out

d_{ij} = actual route distance between the facilities i and j

c_{ij} = cost of moving the materials between the facilities i and j

w_{ij} = preferred proximity weight between the facilities i and j

- Smaller the objective function value, better the chromosome fitness.

In order to handle the hard constraints (overlap) and soft constraints (tower crane accessibility) in the evaluation process, 'Penalty Approach' has been identified as an efficient approach (Mahachi 2001). Hence, in this optimization, if a solution violates any of the hard/soft constraints, an arbitrary high positive value (penalty) will be added to the objective function, which reduces the fitness of that solution, leading to the elimination of that layout and better optimal solutions in the successive iterations. Deb (2009) has suggested the following considerations for assigning the penalty values.

- i) Very small penalty values (compared to objective function value) have a small distortion on the objective function, and the optimum $F(x)$ may not be near the true constrained optimum.
- ii) Very large penalty values have a severe distortion on the objective function and may lead to locally optimal solutions.
- iii) Penalty value should be assigned in such a way that it is relative to the objective function value.

In evaluating the objective function, as recommended by Osman and Georgy (2005), this study uses the proximity weight to be defined as 0, 1, 3, 9, 37, or 81 (Table 4.3) and the penalty function value as 10,000 for each constraint violation. The values in Table 4.3 represent the user's preference of the required proximity among the temporary construction facilities. As illustrated in the table, if the user prefers two facilities to be located as close as possible for some project specific constraints, it shall be represented by a high proximity value (e.g., 81). Similarly, if project requirements prefer two facilities to be located as farthest as possible, it shall be represented by a low proximity value (e.g., 0).

Table 4.3: Preferred proximity relationship mapping (Osman and Georgy 2005)

Desired proximity relationship between the facilities	Proximity weight
Absolutely necessary (A)	81
Especially important (E)	37
Important (I)	9
Ordinary closeness (O)	3
Unimportant (U)	1
Undesirable (X)	0

The fitness evaluation algorithm developed in this study works as follows. For all the solutions in the solution pool, the objective function is calculated as a cumulative value of product of distance between each of the onsite stored temporary facilities (centroid to centroid) and their corresponding proximity weight and interaction cost values. While this value represents the unconstrained objective function value, a check for facilities overlap and violation of any user defined constraints is performed as-well. If there is any constraint violation, an appropriate penalty value is added to the unconstrained objective function value to reduce the fitness of that solution.

In this study, constraints have been handled implicitly and explicitly. Implicit handling of constraints includes ensuring not allocating temporary facilities within the boundaries of the permanent structure and utility/unusable site areas, if any. It is implemented in the beginning of the optimization model, where the non-available grids (grids occupied by permanent structure

including safety boundary and utility/unusable areas) are removed, and only the available grids are considered for layout identification in the initial population algorithm.

Two types of constraints are handled explicitly in this study. First constraint is the crane accessibility constraint which ensures that all the facilities that require access by the tower crane are within the reachable radius of the crane. This constraint violation is checked at the facility level, in which for each of the temporary facility in the solution that requires crane access, a check is done to identify if the facility is within the reachable radius of the crane provided by the user. If there is a constraint violation (not within the reachable radius), penalty value is added to the unconstrained objective function value of the corresponding solution. If not, the unconstrained objective function value is retained.

The second constraint handled in this study is the non-overlap constraint through which a check is done to ensure that none of the temporary facility overlaps with the position of other temporary facilities and all the onsite temporary facilities are positioned within the available site boundaries. The algorithm evaluates the non-overlap of temporary facilities by analyzing their position and alignment types that were identified during the solution generation process. If an overlap or out-of-boundary violation is identified for a solution, penalty value is added to the objective function value of the corresponding solution.

Offsite Facilities

Once the constrained objective function value is identified, the functional value is recalculated to account the optimality of the facilities stored offsite. In this, an objective function is calculated as a cumulative value of product of distance between the onsite facility and each of the offsite stored temporary facilities and their corresponding proximity weight, interaction cost

values, and is then added to corresponding solution function. The distance between the onsite facility and offsite facilities is measured as the sum of distance between the offsite location and site gate (provided by user), and distance between the site gate and the facility centroid located onsite.

Pseudo code for solutions evaluation algorithm is provided in the following section.

Solutions Evaluation Pseudo Algorithm

All the available solutions will be stored in the variable Solutions(i) // *Generated from the initial solutions algorithm*

Function Evaluate_Solutions ()

Define an array variable Objective_Function(s) // *To store the objective function value of the different solutions*
 Define an variable TC_Reachable_Radius // *To store the reachable radius of the tower crane*
 Define an variable TC_Coordinate_X // *To store the X coordinate of tower crane position*
 Define an variable TC_Coordinate_Y // *To store the Y coordinate of tower crane position*
 TC_Reachable_Radius = Value entered by the user in UI in the text box for reachable radius of tower crane
 TC_Coordinate_X = Value before the comma entered by the user in the text box for TC position
 TC_Coordinate_Y = Value after the comma entered by the user in the text box for TC position
 Define an variable Penalty_Value // *To store the penalty value that need to be added for each constraint violation*
 Penalty_Value = 10000 // *Can be changed later*
 Define a variable Offsite_Distance // *To store the distance b/w Offsite storage and site gate*
 Define a variable Gate_Position // *To store the coordinate of site gate*
 Define a variable Gate_Grid // *To store the grid # of the site gate (to be manipulated)*
 Offsite_Distance = Value provided by the user in the text box for Offsite Distance
 Gate_Position = Coordinate provided by the user in the text box for site gate position
 Gate_Grid = Grid # of Gate_Position (to be manipulated based on the Gate_Position Value)
 Store the #'s of the grids occupied by the building (with safety boundary) and unusable areas in an array variable 'Evtl_UG()' (Unusable Grids)
 Store the #'s of the available grids in the site area in an array variable 'Evtl_AG()' (Available Grids)
 Store the # of rows available in the entire grid split-up in a variable 'Rows'
 Store the # of columns available in the entire grid split-up in a variable 'Columns'

For k = 1 to i // *Where i = number of available solutions, which will always be 100*
 Objective_Function(k) = 0 // *To start with the objective function value as zero*
 Define a variable 'Evtl_Temp_AG()' // *To store the available grids locally for computations*
 Define a variable 'Evtl_Temp_UG()' // *To store the unavailable grids locally for computations*
 Evtl_Temp_AG() = Evtl_AG() // *or AG()*
 Evtl_Temp_UG() = Evtl_UG() // *or UG()*

For j = 1 to n // *Where n = number of facilities provided by the user*
 If Offsite_Storage(j) is Unchecked, then proceed further, else exit and increment j by 1
 Define a variable Source_Position // *To store the position of the jth facility of the current solution iteration*
 Source_Position = Grid # assigned to the jth facility in the current solution iteration // *To be read from Solutions(k)*
 For m = j+1 to n // *Where n = number of facilities provided by the user*
 If Offsite_Storage(m) is Unchecked, then proceed further, else exit and increment m by 1
 Define a variable Target_Position // *To store the position of the mth facility of the current solution iteration*
 Target_Position = Grid # assigned to the mth facility in the current solution iteration
 Objective_Function(k) = *Objective_Function(k) + (Distance b/w Source_Position & Target_Position * Interaction cost value of j,m * Proximity weight value of j,m)*
 m=m+1
 End Loop // *For m*

// *Start Penalty assignment*
 // *Tower crane accessibility of each facility entered in UI*
 If facility j accessed by tower crane = True // *To check if the jth facility is accessible by tower crane*
 Define a variable Eucl_Dist // *To store the Euclidean distance b/w tower crane and source position*

$$Eucl_Dist = \sqrt{(TC_Coordinate_X - X\ of\ Source_Position)^2 + (TC_Coordinate_Y - Y\ of\ Source_Position)^2}$$
 If Eucl_Dist <= TC_Reachable_Radius
 Objective_Function(k) = *Objective_Function(k) + 0 // Value not increased as it is reachable*
 Else
 Objective_Function(k) = *Objective_Function(k) + Penalty_Value // Value increased as it is not reachable*
 End if // *For reachable radius*
 Else
 Objective_Function(k) = *Objective_Function(k) + 0 // Value not increased as it is not accessed by TC*
 End if // *For TC access check*

```

// to check overlap
Define a variable 'Source_Alignment'
Define a variable Evtl_Horz_Grid_Reqd
Define a variable Evtl_Vert_Grid_Reqd
Source_Alignment = Alignment # assigned to the jth facility in the current solution iteration // To be read
from Solutions(k)
Evtl_Horz_Grid_Reqd = int(Length(j)/Grid_Size) // Length(j) = Length of current facility
Evtl_Vert_Grid_Reqd = int(Breadth(j)/Grid_Size) // Breadth(j) = Breadth of current facility

Case When Source_Alignment =1 // For the case Horizontal, Right, and Above
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Vert_Grid_Reqd-1
  Evtl_Vert_Position = Source_Position + (m * Columns)
  For n=1 to Evtl_Horz_Grid_Reqd
    If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position-1/columns)) then
      Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
overlap
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position +1
    Else
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position +1
  End
  n=n+1
Loop
m=m+1
Loop

When Source_Alignment =2 // For the case Horizontal, Left, and Above
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Vert_Grid_Reqd-1
  Evtl_Vert_Position = Source_Position + (m * Columns)
  For n=1 to Evtl_Horz_Grid_Reqd
    If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position+1/columns)) then
      Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
overlap
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position -1
    Else
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position -1
  End
  n=n+1
Loop
m=m+1
Loop

When Source_Alignment =3 // For the case Horizontal, Right, and Below
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Vert_Grid_Reqd-1
  Evtl_Vert_Position = Source_Position - (m * Columns)

```

```

For n=1to Evtl_Horz_Grid_Reqd
  If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
  int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position-1/columns)) then
    Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
    overlap
    Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
    and added to the array variable Evtl_Temp_UG()
    Evtl_Vert_Position = Evtl_Vert_Position +1
  Else
    Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
    and added to the array variable Evtl_Temp_UG()
    Evtl_Vert_Position = Evtl_Vert_Position +1
  End
  n=n+1
Loop
m=m+1
Loop

```

```

When Source_Alignment =4 // For the case Horizontal, Left, and Below
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Vert_Grid_Reqd-1
  Evtl_Vert_Position = Source_Position - (m * Columns)
  For n=1to Evtl_Horz_Grid_Reqd
    If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
    int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position+1/columns)) then
      Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
      overlap
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
      and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position -1
    Else
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
      and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position -1
    End
    n=n+1
  Loop
  m=m+1
Loop

```

```

When Source_Alignment =5 // For the case Vertical, Right, and Above
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Horz_Grid_Reqd -1
  Evtl_Vert_Position = Source_Position + (m * Columns)
  For n=1to Evtl_Vert_Grid_Reqd
    If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
    int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position-1/columns)) then
      Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
      overlap
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
      and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position +1
    Else
      Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
      and added to the array variable Evtl_Temp_UG()
      Evtl_Vert_Position = Evtl_Vert_Position +1
    End
    n=n+1
  Loop
  m=m+1
Loop

```



```

When Source_Alignment =6 // For the case Vertical, Left, and Above
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Horz_Grid_Reqd -1
    Evtl_Vert_Position = Source_Position + (m * Columns)
    For n=1to Evtl_Vert_Grid_Reqd
        If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position+1/columns)) then
            Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
overlap
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position -1
        Else
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position -1
        End
    n=n+1
    Loop
m=m+1
Loop

```

```

When Source_Alignment =7 // For the case Vertical, Right, and Below
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Horz_Grid_Reqd -1
    Evtl_Vert_Position = Source_Position - (m * Columns)
    For n=1to Evtl_Vert_Grid_Reqd
        If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position-1/columns)) then
            Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
overlap
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position +1
        Else
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position +1
        End
    n=n+1
    Loop
m=m+1
Loop

```

```

When Source_Alignment =8 // For the case Vertical, Left, and Below
Define a variable Evtl_Vert_Position
For m=0 to Evtl_Horz_Grid_Reqd -1
    Evtl_Vert_Position = Source_Position - (m * Columns)
    For n=1to Evtl_Vert_Grid_Reqd
        If Evtl_Vert_Position is not in Evtl_Temp_AG() or (if n>0 and
int(Evtl_Vert_Position/columns) <> int(Evtl_Vert_Position+1/columns)) then
            Objective_Function(k) = Objective_Function(k) + Penalty_Value // Facility has a
overlap
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position -1
        Else
            Evtl_Vert_Position should be removed from the array variable Evtl_Temp_AG()
            and added to the array variable Evtl_Temp_UG()
            Evtl_Vert_Position = Evtl_Vert_Position -1
        End
    End

```

```

        n=n+1
        Loop
        m=m+1
        Loop
    End Case
    //End Penalty assignment

    // Offsite
    For z = 1 to n // Where n=number of facilities provided by the user
    If Offsite_Stored(z) is Checked, then proceed further, else exit and increment z by 1
        Objective_Function(k) = Objective_Function(k) + ((Offsite_Distance + Distance b/w Gate_Grid &
        Source_Position) * Interaction cost value of j,z * Proximity weight value of j,z)

        z=z+1
    End Loop // For z

    j=j+1 // Step I
    End Loop // For j

k=k+1
End Loop // For k
End Function Evaluate_Solutions

```

4.2.2.3 Genetic Operations Algorithms: Crossover and Mutation

This algorithm improves the solutions available in the population set by eliminating the worst solutions and creating new solutions from the retained best solutions by utilizing the genetic operators, crossover (to generate new offspring from parents), and mutation (to introduce random changes to the solution set to ensure diversity). The algorithm works as follows.

Once the value of the objective function of each solution is determined, the solutions are first sorted in ascending order (lowest to highest objective function value). In this sorted solution set, half of the solutions with worst objective function value (highest objective value) are then eliminated (Haupt and Haupt 2004). Thus, the solutions set is now left with half number of solutions of that was initially started, which have the best objective function value.

Crossover

First two solutions from the reduced solution set are selected and a random crossover point is chosen to swap the solution genes among the selected solutions and generate new offspring. The crossover point is identified as a random number between 0 and number of facilities stored onsite. This operation of generating new offspring using the crossover operator is repeated for the next set (solutions 3 and 4) available in the solution set and is repeated until all the best retained solutions are subjected to the crossover operation. Thus, at the end of the crossover process, the number of solutions available in the solutions set will again be equal to number of solutions that was initially started with.

For example, consider the example 4.2 discussed earlier where six temporary construction facilities are used in a construction project of which four are stored onsite (facilities 1, 3, 4, 5) and two in offsite (facilities 2, 6). Assume there are two parent chromosomes (Tables

4.4.a, 4.4.b) over which the crossover operation is performed to generate new offspring/solutions. Assume that the random crossover point is identified as 3. Hence, the algorithms perform the crossover operation of the facilities represented after facility# 3 in the parent chromosomes and generate the two offspring (represented in Tables 4.5.a, 4.5.b).

Table 4.4.a: Parent Chromosome I

Facility #	1	3	4	5
Alignment	7	6	3	4
Grid Position	25	100	5	215

Table 4.4.b: Parent Chromosome II

Facility #	1	3	4	5
Alignment	2	5	8	6
Grid Position	52	99	189	268

Table 4.5.a: Offspring Chromosome I

Facility #	1	3	4	5
Alignment	7	6	8	6
Grid Position	25	100	189	268

Table 4.5.b: Offspring Chromosome II

Facility #	1	3	4	5
Alignment	2	5	3	4
Grid Position	52	99	5	215

Mutation

Once the new offspring are generated using the crossover operation, the solution set is subject to a random, yet directed change to ensure diversity among the generated solutions and avoid premature sub-optimal convergence. This change is introduced through the Mutation operator, in which a random solution and a gene location is identified from the solution set and the alignment type of the facility located in the corresponding gene location is substituted with a new possible random alignment types. The random chromosome and random gene location is identified using the equation 4.2 provided in the following section.

$$\begin{aligned}
rC &= \text{int}((\mu * \text{numberOfSolutions}) + \text{random}(1, \text{numberOfSolutions} - 2)) \\
rGL &= \text{int}(\text{random}(\mu, 1) * (\text{numberOfOnsitefacilities} - 1)) \\
\text{where, } \mu &= \text{mutation probability/rate} \\
rC &= \text{random chromosome, } rGL = \text{random gene location}
\end{aligned}
\tag{Eq. 4.2}$$

Based on the review of literature, this study uses a mutation probability/rate (μ) of 1% (Deb 2009).

Once new offspring and random changes are introduced to the population set, the solutions are again evaluated using the evaluation algorithm and this process of solutions evaluation and solutions generation is iterated for several times until the solution set is dominated with one single optimal solution. (Li and Love 1998) has suggested that number of solution iterations of 100 leads to better solutions with reasonable computational time. This study also defines the number of solution iterations as 100. If the iteration is run for the last time (100th time), the algorithm will not introduce the random changes to the solution set as it would have an adverse effect on the objective value of the solution set (Haupt and Haupt 2004).

Pseudo code for solutions generation algorithm is provided in the following section.

All the available solutions with their corresponding rank (objective value) will be stored in the variable Solutions(i) // *Generated from the evaluate solutions algorithm*

Function Generate_Solutions ()

Define a variable Solution_Iteration // *To store the # of iterations to be done*

Solution_Iteration = 100

Define a variable MutationProbability // *To store the probability of mutation*

MutationProbability = 0.01

Define a variable MutationCycle // *To store for how many chromosomes mutation need to be done*

MutationCycle = int(MutationProbability * numberOfSolutions)

Define a variable OnsiteStoredFacilities

Define a variable numberOfOnsiteStoredFacilities

numberOfOnsiteStoredFacilities = count of facilities for which 'Is Stored Offsite ' is not checked

OnsiteStoredFacilities = Facility #s of the facilities for which 'Is Stored Offsite ' is not checked

For s = 1 to Solution_Iteration

Sort the solutions available in Solutions(i) based on their rank (objective value) in ascending order

Discard the last (numberOfSolutions/2) from Solutions(i)

If s >= Solution_Iteration-1 then Exit function // *To check if last iteration*

Else proceed

// Crossover

For t = 1 to numberOfSolutions/2

Define a variable 'Chromosome1'

Define a variable 'Chromosome2'

Chromosome1 = Solutions(t)

Chromosome2 = Solutions(t+1)

Define a variable 'CrossoverPoint' // *To store the random crossover point*

CrossoverPoint = Random (from OnsiteStoredFacilities)

IntermediateSolutions(t) = Solution from Chromosome1 for until CrossoverPoint + Solution from Chromosome2 after CrossoverPoint

IntermediateSolutions(t+1) = Solution from Chromosome2 for until CrossoverPoint + Solution from Chromosome1 after CrossoverPoint

Add the solutions available in IntermediateSolutions (t) and IntermediateSolutions (t+1) to Solutions(i)

t = t+2

Loop // *End Crossover*

// Mutation (flip2edge)

For v = 1 to MutationCycle

Define a variable 'MutateChromosome' // *To store the Chromosome # to be mutated*

Define a variable 'MutateGene' // *To store the gene/facility # to be mutated*

Define a variable 'MutateValue' // *To store the alignment type to be flipped*

*MutateChromosome = int[(MutationProbability * numberOfSolutions) + Random(1, numberOfSolutions - 2)]*

*MutateGene = int(Random(MutationProbability, 1) * numberOfOnsiteStoredFacilities - 1)*

MutateValue = int(Random (1,8))

For the Solution(MutateChromosome), and the facility of 'MutateGene' the alignment type should be updated with the 'MutateValue'

v = v+1

Loop // *End Mutation (flip2edge)*

If s < Solution_Iteration then

Call Evaluate_Solutions()

Else

Exit Loop

End if

s = s+1

Loop // *End Generate Solutions function*

4.3 Summary

CSUP problem that deals with accommodating the site-level temporary facilities that are used to support the project is a vital resource, and greatly contributes in achieving the project objectives. Solving the construction site layout planning problem is also inherently complex in nature that presses the need for a computationally robust global optimization technique. As researchers have identified that GAs has the potential to solve several engineering and construction management problems without getting into to local optima in which other algorithms often get trapped (Elbeltagi et al. 2004; Li and Love 1998), the optimization model developed in this study is based on GAs. Also, to overcome the limitations of earlier studies reviewed in the literature, four new algorithms were developed in this study that includes,

- i) Improved distance measurement algorithm
- ii) Initial solutions generation algorithm
- iii) Solutions evaluation algorithm
- iv) Solutions generation algorithm

The following chapter discusses in detail about the user interface and architecture of the optimization tool developed in this research study.

Chapter 5: CONSITEPLAN – Architecture Review

5.1 Introduction

While a detailed discussion of the algorithms developed in this study was presented in Chapter 4, a review of the architecture of the optimization tool (CONSITEPLAN) is discussed in this chapter. CONSITEPLAN is a multi-objective optimization tool developed to solve the site utilization planning problem encountered in construction projects. CONSITEPLAN has been developed using the Microsoft Visual Studio platform. This chapter provides details of the architecture of the tool.

5.2 CONSITEPLAN – Process flow

A graphical illustration of flow of components of the developed optimization tool is provided in Figure 5.1.

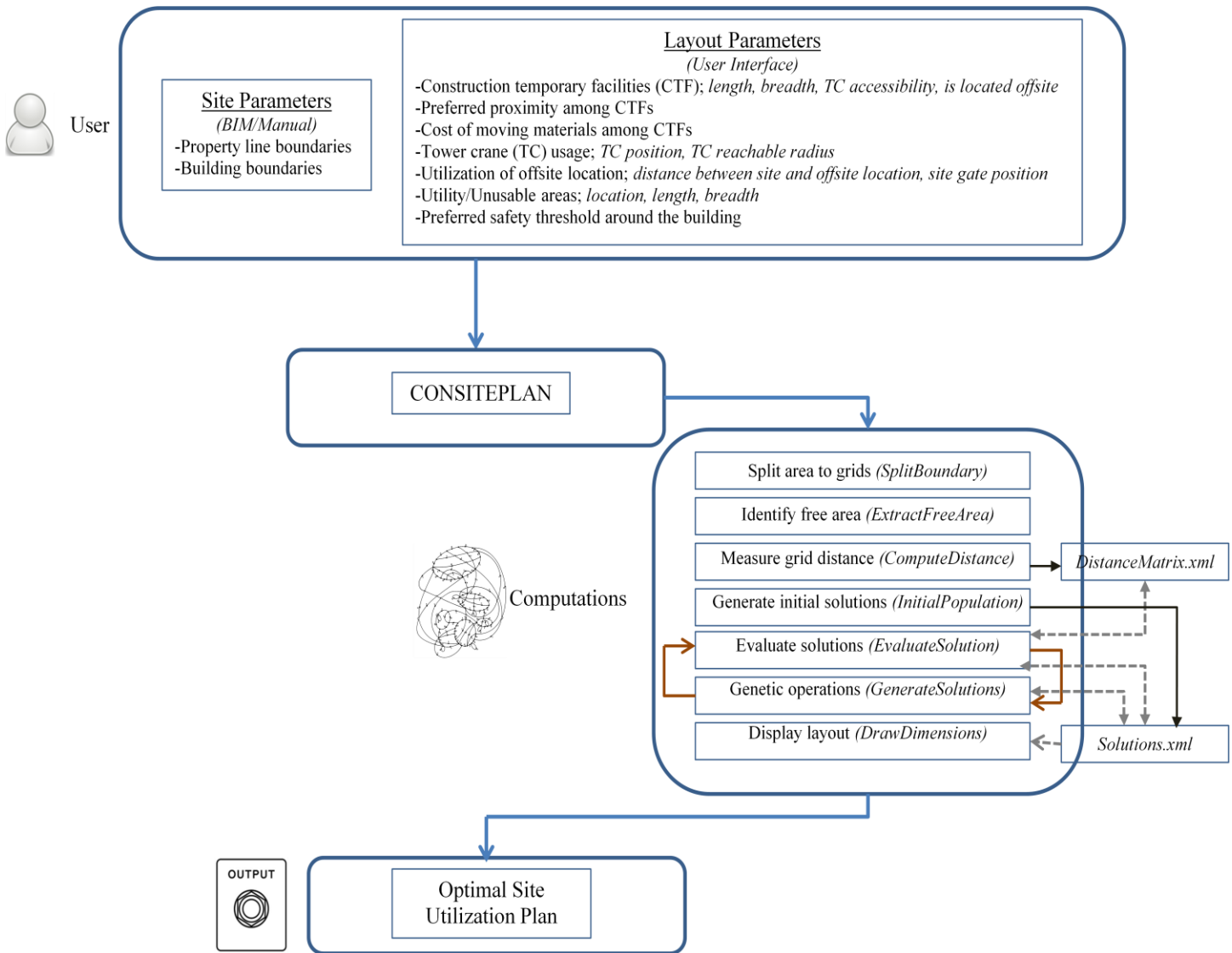


Figure 5.1: CONSITEPLAN Architecture Flow

While Figure 5.1 illustrates the architecture of the tool graphically, and Figure 5.2 illustrates the GAs optimization process flow of the tool, the following discussion provides detailed information about each of the components listed in the diagram.

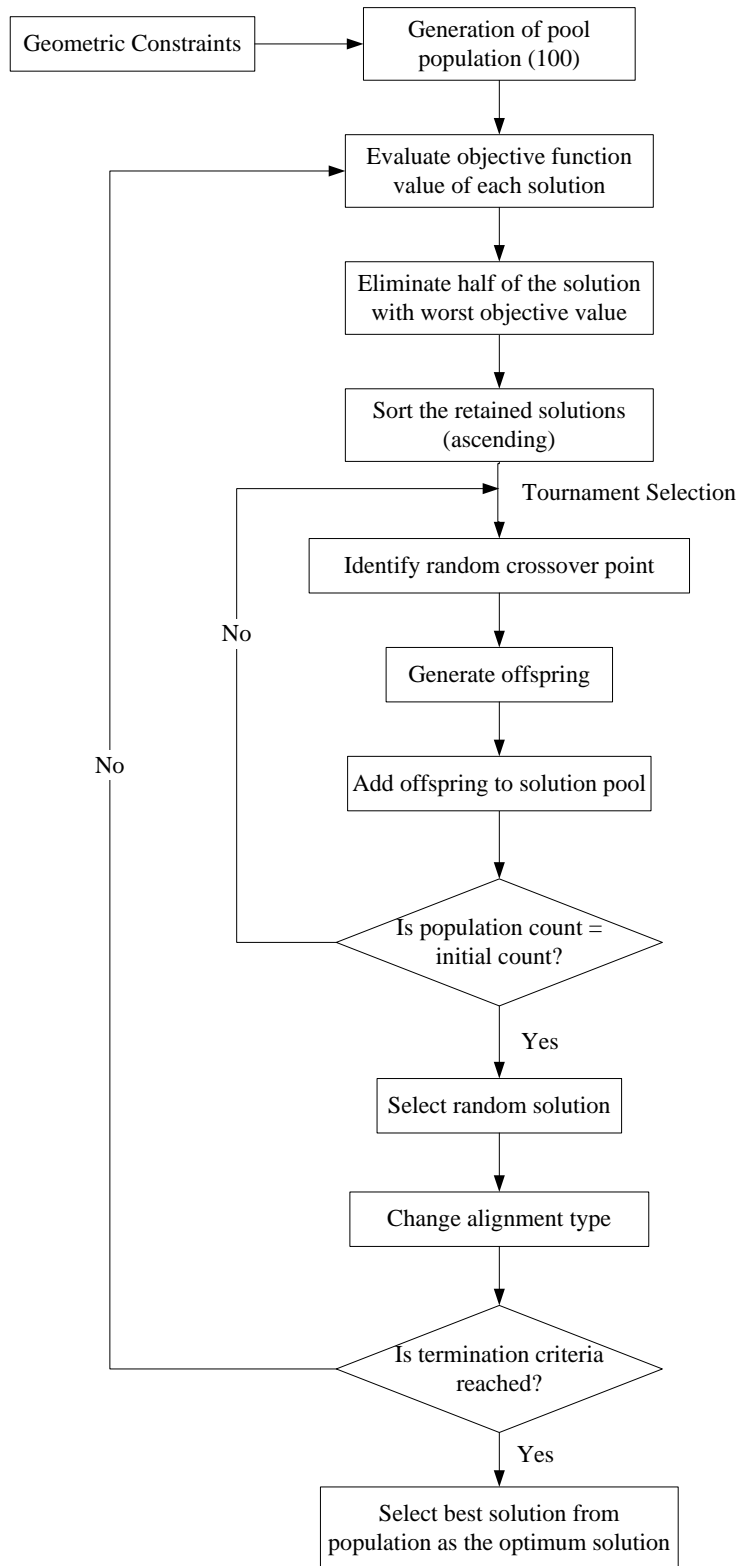


Figure 5.2: CONSITEPLAN GAs Optimization Process Flowchart

5.3 CONSITEPLAN – Input

The data required for identifying a good site layout plan includes geometric information about the site, the final structure to be completed, the size of various temporary facilities, the project manager’s preferences regarding the location of these facilities, areas of the site where temporary facilities can’t be laid out and the location of tower crane if it is used in the project. The data requirements of the tool are discussed in detail in the following sections.

5.3.1 Site Parameters

The primary input to the site utilization planning tool includes the site boundaries within which the layout needs to be planned, and the layout of the building to be constructed. The information can be obtained in two ways. If a BIM model using REVIT software is being used, this tool provides some basic interfaces to extract information from the BIM model file. A REVIT application programming interface has been integrated with CONSITEPLAN which can extract geography of the property lines (site boundaries) and the building layout from a REVIT model. In project cases where a REVIT model is not available, the user can always manually provide the appropriate parameters in an Extensible Markup Language (Xml) file (Sitedetails.xml), which extends the usability of the tool to a wide range of projects. Xml files are a textual data format file which aids to define a set of rules for encoding documents in formats that can be read by both humans and machine. As the xml files are simple to read, highly generic, and are currently used in programming interfaces, CONSITEPLAN uses xml files for data storage.

Following section provides a sample of how the site boundaries / property lines are stored in the Sitedetails.xml file. As shown, the file starts with a main tag ‘SiteDetails’ beneath which

the layout parameters (site boundaries, building layout) are stored under individual tags. The provided example illustrates the data storage format for site boundaries, where all the site boundary vertices are stored in clockwise direction under the sub-tag 'PropertyPoints'.

Solutions Evaluation Pseudo Algorithm

```
<?xml version="1.0" encoding="utf-8"?>
<SiteDetails xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <PropertyPoints>
    <XYZPoint>
      <X>-120</X>
      <Y>-100</Y>
      <Z>0</Z>
    </XYZPoint>
    <XYZPoint>
      <X>-120</X>
      <Y>71</Y>
      <Z>0</Z>
    </XYZPoint>
    <XYZPoint>
      <X>102</X>
      <Y>71</Y>
      <Z>0</Z>
    </XYZPoint>
    <XYZPoint>
      <X>102</X>
      <Y>-100</Y>
      <Z>0</Z>
    </XYZPoint>
  </PropertyPoints>
</SiteDetails>
```

5.3.2 *Layout Parameters*

To enhance the ease of using the tool, CONSITEPLAN is designed with a User Interface (UI) screen through which details about the layout parameters can be provided by the user. The UI is developed with one primary screen and four secondary screens which are discussed in detail in the following section.

Primary Screen:

The primary screen is designed to collect the following information:

i) Phase of the project:

In reviewing the published literature, it was found that there is an agreement amongst researchers that the construction site layout planning problem may be a dynamic problem, i.e., different facilities may be required during different phases of time during the life cycle of the project based on the work being executed at the site. While one layout may be sufficient in some projects, it is quite possible that various facilities get used during different time spans of a project. This tool broadly breaks the project down in two phases, sub-structure construction phase and super-structure construction phase. The architecture of the tool is modular and extensible to allow for definition of additional user defined phases in the future revision of the tool.

ii) Safety boundary around the permanent structure

OSHA recommends not to store materials within the free fall zone of the permanent structure and to maintain a recommended buffer distance around the structure to improve construction safety in site. For example, a buffer distance of 6 feet can be maintained around the

permanent structure which eliminates material storage in those spaces and increases site safety. This buffer distance need to be decided based on the construction activities going on-site and any project specific constraints. For example, the safety buffer may be significantly wider than the building's footprint during the substructure phase on account of the need to construct basements/foundations at a significant depth. Hence, CONSITEPLAN has been designed with the option to request the user's preference on the preferred safety boundary around the building structure, and not utilizing that space for identifying optimal layouts.

iii) Preferred grid size

As discussed in the literature, layout planning on construction can be conceptualized as, i) making discrete one to one assignments where facilities are assigned to previously defined locations ii) continuous domain (implemented in CONSITEPLAN). The second conceptualization theoretically creates infinite layouts and is computationally intensive as it increases the potential locations for locating the facilities exponentially. To overcome this complexity, CONSITEPLAN divides the site space into smaller grids and uses the available grids as discrete options for the layout. For example, consider a case where the size of the site is 100 x 100 ft and the preferred grid size is provided as 10. In this case, the site space will be divided into 100 smaller grids as illustrated in Figure 5.3. The size of the grids (the size of the search space) has a direct correlation with the number of possible layout options as well as the computational effort required.

$$Pu, n = \frac{n!}{(n-u)!} \quad (\text{Cheng et.al, 2002})$$

where,

u = number of facilities

n = number of locations

For example, the 100' x 100' site, divided into 10' x 10' grids has 100 vertices at which a facility could be located. If, for example, we need to lay out 4 facilities on this site, we would have a search space of 3,921,225. Thus, the smaller the grid size, the larger the search space and the computational effort. For example, with a 1' X 1' grid size, the size of search space increases exponentially to 416,416,712,497,500 (416.41 trillion).

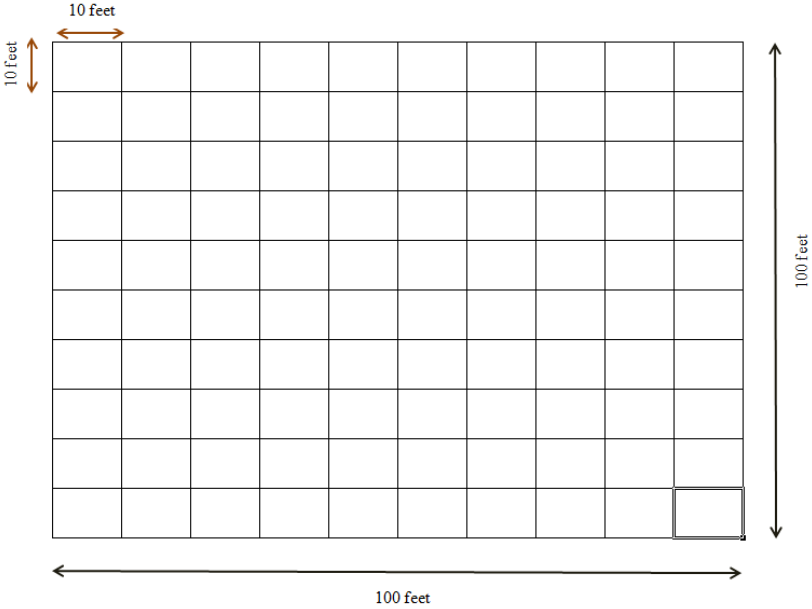


Figure 5.3: Site Space Conceptualization

iv) Tower crane

In many building projects where space is confined, tower cranes are an effective mechanism of transporting materials and equipment. If a tower crane is used in a project, it may be efficient to enable it to access certain temporary facilities (such as rebar fabrication area). This requirement / optimization constraint has been implemented in this tool.

v) Temporary facilities, Offsite storage

In addition, the user provides the number of temporary facilities that need to be laid out on the site. Construction projects in urban areas tend to have limited working space at the site and tend to utilize the available offsite spaces, if any. Hence, if one or more facilities need to be located offsite, the tool facilitates the user to provide information regarding the location of the site gate, distance between the offsite location and the site, and identify the facilities to be located offsite. Figure 5.4 shows a screen capture of the primary screen.

CONSITEPLAN

About Contact us

Phase Details

Project phase : Sub-Structure

Safety boundary : 6

Preferred grid size : 10

Number of facilities : 5

Tower crane usage : Yes

Tower crane reachable radius : 100

Tower crane position : 50,85

Offsite storage use : Yes

Distance between Offsite and site gate : 525

Site gate position : -100,85

Facilities details Other occupied areas detail

Facilities-Phase Constraints/Requirements

Cost of interaction between the facilities Proximity weight between the facilities

Site Layout Generation

Generate site layout

Figure 5.4: CONSITEPLAN – Primary Screen

Secondary Screen – *Facilities Details*:

Facilities details screen facilitates the user to provide information about the temporary facilities to be laid out for the selected project phase. These details of the facilities include, facility description, length (measured in feet), breadth (measured in feet), tower crane access, and offsite location of the facility. Figure 5.5 shows the screen shot of the data entry page.

Site Layout Planner: Enter phase specific details

Facilities details:
Please enter the facilities details required for the selected project phase

Facility No	Facility Description	Length	Breadth	Tower Crane Accessed	Is Stored Offsite
1	Storage	25	25	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Fabrication	50	30	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Rest Area	8	4	<input type="checkbox"/>	<input type="checkbox"/>
4	Laydown	75	50	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Trailers	45	20	<input type="checkbox"/>	<input type="checkbox"/>

Save Reset Clear Close

Figure 5.5: CONSITEPLAN – Facilities Detail


Secondary Screen – *Unusable Areas*:

There may be some areas of the site where the project manager may not want to locate a temporary facility. The unusable areas screen facilitates the user to define the unusable areas in

the project site. The unusable area can be a utility area (for example, piping, sump), hazardous area, or project manager’s preference of not utilizing specific onsite area(s) for laying out the temporary facilities. In this screen, the user can provide details about the unusable areas at the project including, unusable area description, geographical location, length (measured in feet), and breadth (measured in feet). Figure 5.6 shows the image of the utility/unusable areas screen.

Site Layout Planner: Enter phase specific details

Unusable area details:
Please enter the unusable area details for the selected project phase

	Description	Centroidal Position x,y	Length	Breadth
	Utility	67,12	25	25
	Hazard Zone	-107,-85	15	10
*				

Save Reset Clear Close

Figure 5.6: CONSITEPLAN – Unusable Areas

Secondary Screen – *Cost Interaction*:

The cost interaction screen is designed to facilitate the user in providing the cost of moving the materials among the temporary construction facilities provided in the facilities detail screen. Cost details provided in this screen reflects the cost of moving the materials per unit distance from one temporary facility to another temporary facility. As the cost of moving a material from location1 to location2 will be the same as moving from location2 to location1, the cost matrix is designed as a symmetric matrix in which the lower matrix will be disabled for user input. Figure 5.7 shows the image of the cost interaction screen.

Site Layout Planner: Enter phase specific details

Cost Interaction Matrix:
Please enter the cost of interaction among the facilities for the selected project phase.

Facility	Storage	Fabrication	Rest Area	Laydown	Trailers
Storage		25	8	6	10
Fabrication			12	7	5
Rest Area				5	9
Laydown					2
Trailers					

Save Reset Clear Close

Figure 5.7: CONSITEPLAN – Cost of Interaction among Facilities

Secondary Screen – Preferred Proximity:

The preferred proximity screen is designed to facilitate the user in providing his preference on how close the facilities should be located based on the project requirements. As discussed in chapter 4, the user shall provide the preferred proximity as 81, 37, 9, 3, 1 or 0 based on the project preferences/requirements (Osman and Georgy 2005). The preferred proximity screen is designed as symmetric matrix. Figure 5.8 shows the image of the preferred proximity screen.

Site Layout Planner: Enter phase specific details

Preferred Proximity Weight Matrix:
Please enter the preferred proximity weight among the facilities for the selected project phase.
Proximity weight shall be signified as below.
81 (Absolutely necessary), 37 (Especially important), 9 (Important), 3 (Ordinary closeness), 1 (Unimportant), 0 (Undesirable)

Facility	Storage	Fabrication	Rest Area	Laydown	Trailers
Storage		81	1	37	0
Fabrication			1	37	0
Rest Area				1	9
Laydown					0
Trailers					

Save Reset Clear Close

Figure 5.8: CONSITEPLAN – Preferred Proximity among Facilities

5.4 CONSITEPLAN – Optimization Components

This section discusses about the various optimization components/computational functions of the developed tool.

5.4.1 SplitBoundary

Once all the input parameters are provided and the user chooses the option to generate an optimal site utilization plan, the function ‘SplitBoundary’ is invoked by CONSITEPLAN. This function splits the entire site area into smaller grids of the preferred grid size provided by the user in the primary screen. Splitting the entire site area into grids of smaller size facilitates the idea of conceptualizing the site space as a continuous domain in solving the problem.

5.4.2 ExtractFreeArea

The function ‘ExtractFreeArea’ identifies the available grids (grids which are not occupied by building and utility areas) from the total grids in the project site. To accomplish this, the function first expands the layout out of the building by adding the preferred safety boundary provided by the user. Then, the set of grids which are not in the zones of the expanded building layout and the utility areas (unusable areas provided by the user) are identified as available grids. These grids are used in the subsequent functions in identifying an optimal layout plan for the temporary facilities.

5.4.3 ComputeDistance

Once the available grids are identified, the function ‘ComputeDistance’ calculates the potential travel distance (improvised distance measurement) between all the available grids

based on the algorithm discussed in chapter 4 (4.2.1). The distance measured between all the available grids are written in a dataset format to the output file, DistanceMatrix.xml, which will then be subsequently accessed in later part of the optimization algorithm.

5.4.4 InitialPopulation

This algorithm generates 100 random initial solutions to position the onsite temporary construction facilities in the available grids in any of the possible 8 alignment types. The section 4.2.2 of this thesis document provides a detailed discussion about the algorithm and pseudo code.

5.4.5 EvaluateSolution

In optimizing layouts using Genetic Algorithms, the generated layouts need to be evaluated for their fitness in meeting the objectives and constraints specified by the user. This process is carried out for every solution in every generation. Then, typically, the worst solutions (solutions with high objective function value) are eliminated and new solutions are created from the best retained solutions using the genetic operators such as Crossover and Mutation. The function 'EvaluateSolution' assesses each of the solution in the solution pool based on the provided user preferences and constraints, and calculates their corresponding objective function value. The detailed information about the objective function and evaluation processes is presented in section 4.2.3 of this document.

5.4.6 GenerateSolutions

As discussed in section 5.4.5, genetic operations are performed on the evaluated solutions. These include elimination of worst solutions and generation of new solutions from the remaining high quality solutions. The function 'GenerateSolutions' identifies and eliminates the

worst solutions (solutions with high objective function value) from the set of evaluated solutions and then generates new solutions from the retained solutions by performing the crossover and mutation operations. The functions ‘EvaluateSolution’ and ‘GenerateSolutions’ are iterated for several times (100 times in this research study, (Li and Love 1998)) after which the best solution from the solution pool (solution with minimum objective function value) is identified as the optimal layout. The detailed information about the crossover and mutation operations with the related pseudo-code can be found in section 4.2.4.

5.4.7 DrawDimensions

The function ‘DrawDimensions’ is used to display the identified optimal solution in a graphical format. It displays the building layout along with the utility and facilities areas supplemented with a tooltip listing their dimensions, position, and alignment type.

5.5 Summary

CONSITPLAN is a Microsoft Visual Studio based optimization tool and it comprises of several functional modules. The code is developed in a modular fashion to facilitate the changes for any future enhancements with minimal effort. It has a primary input screen that facilitates the user to enter the phase specific details and four secondary screens to provide detailed layout attributes (facilities, utility areas, cost of interaction, preferred proximity). It comprises of seven major functional modules (SplitBoundary, ExtractFreeArea, ComputeDistance, InitialPopulation, EvaluateSolution, GenerateSolutions, DrawDimensions) which are used in identifying an optimal layout and displaying the output in a graphical format.

Chapter 6: CONSITEPLAN- Illustrative Example

6.1 Introduction

Several optimization tools have been created by researchers in an attempt to address the site utilization planning problem. As each study has unique attributes, most of the researchers have developed their own examples for implementing their algorithm, and no single example is applied across several studies. In many cases, especially (Yeh 1995) and (Li and Love 1998) have formulated the problem as a one-to-one quadratic assignment problem where the locations of temporary facilities were pre-defined. This problem was further modified and used by Mawdesley and Al-Jibouri (2003) added an interactive cost criteria to it to evaluate the tool developed in their study. (Sanad et al. 2008) applied this tool in ‘Tanta University Educational Hospital’ project in Egypt. The project involved locating 18 temporary construction facilities based on the provided proximity preferences in a site of area 28,500 sq.m., and does not impose additional layout constraints.

(Harmanani et al. 2000) created their own example where the developed tool was used to identify an optimal site utilization plan for a construction site of size 30’ x 20’ in which 10 temporary construction facilities of different size need to be laid out without considering any additional layout constraints. Evosite, an excel based model developed by (Hegazy and Elbeltagi 1999) was implemented on a sample project in which 17 temporary construction facilities (each 800 sq.ft. average) were to be positioned in an available site space of 25500 sq. ft.. The tool

identifies an optimal layout based on the proximity preference, and the example does not account for any additional parameters.

(Elbeltagi and Hegazy 2001; Elbeltagi et al. 2004; Ning et al. 2010a; Ning et al. 2010b; Osman et al. 2003) also evaluated the developed optimization tool by implementing against a manipulated example. As the tools developed in these studies didn't provide any geometric data about the construction site or provide the option of accounting the usage of tower cranes, excluding unusable areas, and utilization of offsite space, these examples were not suitable for optimization using CONSITEPLAN.

Since most of the earlier studies which address the site utilization planning problem have not considered several of the practical layout parameters (tower crane accessibility, offsite storage, unusable areas), the hypothetical examples formulated to implement those models do not provide any relevant information. Since CONSITEPLAN has the sophistication of defining those additional parameters to identify an optimal layout, it becomes necessary to define those parameters in the input for effective evaluations.

The new kinesiology building project, currently under construction at Auburn University on Heisman drive was also evaluated for implementation of the tool. Unfortunately, project execution strategy calls for extensive use of off-site area located at Hemlock Ave. for most temporary facilities. The temporary facilities envisioned on the site include parking lot, site trailers and a small rest area. This project would not have been sufficient to demonstrate the abilities of CONSITEPLAN. Hence, a hypothetical example is used to demonstrate the capabilities of the tool.

6.2 Illustrative Example

This example involves locating 12 temporary construction facilities (11 onsite, 1 offsite) in a site of size 600' x 400', in which the permanent structure is of size 200' x 150' (Figure 6.1).

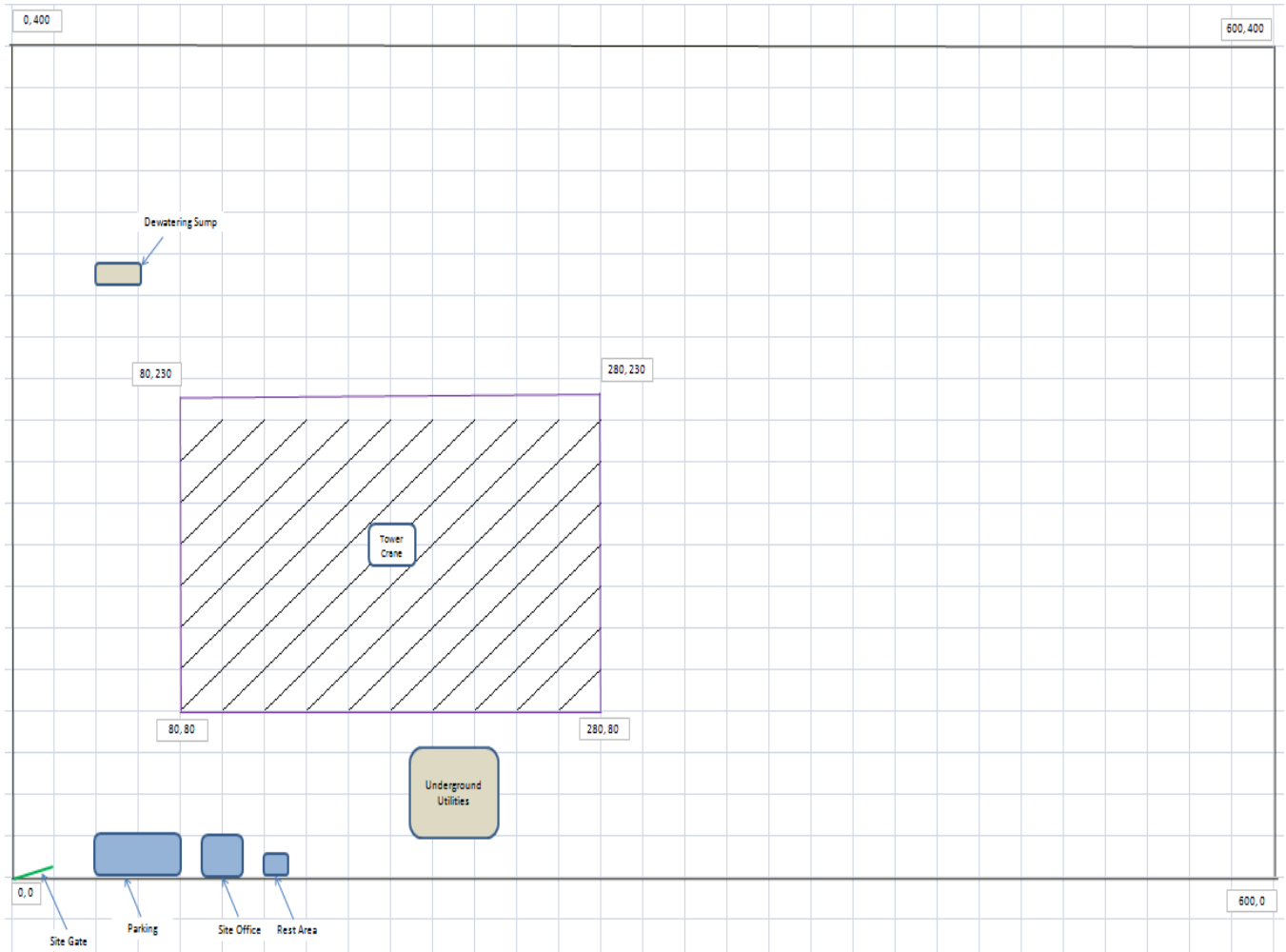


Figure 6.1: Case Study Example Layout

The example utilizes an offsite location that is assumed to be at a distance of 0.25 miles (2640') from the site gate location. The project site is assumed to have two unusable areas which cannot be used for positioning the temporary facilities. It include, i) underground utilities (40' x 40') ii) dewatering sump (20' x 10'). Typically, locating the parking space, site office, and rest areas are not integrated with the layout process of temporary facilities, and are determined by the

project manager. Hence, their locations are predetermined and are defined as fixed facilities (by defining them as unusable areas) in this example. As they are generally located to the site gate (to avoid unnecessary site congestion, enabling easy removal of waste from port-a-potties), the example assumes them to be close to the site gate. Tower crane is assumed to be used in the project (boom length of 150') and is located within the permanent structure. A preferred safety boundary of 10' is assumed around the building footprint.

Table 6.1 provides the list of facilities with their corresponding layout attributes, which needs to be located for this case study example. Typically, as the reinforcement fabrication yard, staging area for tower crane lifting, formwork contractor's laydown area, and cladding contractors laydown are accessed by tower crane, the tower crane accessibility option is enabled for those facilities. Concrete batching plant of the example is assumed to be located offsite.

Table 6.1: Case Study Example Temporary Facilities

<i>Facility #</i>	<i>Facility Description</i>	<i>Length</i>	<i>Breadth</i>	<i>TC access</i>	<i>Offsite</i>
1	Reinforcement Fabrication yard	30	30	Yes	No
2	Reinforcement Contractor's Tool Shed	10	10	No	No
3	Warehouse	40	30	No	No
4	Area for plumbing contractor	60	60	No	No
5	Staging Area for Tower Crane lifting	20	20	Yes	No
6	Formwork Contractor's laydown area	50	50	Yes	No
7	HVAC Contractor's work area	100	60	No	No
8	Cladding Contractor's laydown area	40	40	Yes	No
9	Formwork Contractor's tool trailer	15	10	No	No
10	Cladding Contractor's tool trailer	15	10	No	No
11	Electrical Contractor's Storage Shed	30	20	No	No
12	Concrete Batching Plant	200	200	No	Yes

Table 6.2 provides the preferred proximity among the listed temporary facilities, in which a lower value signifies a least preference and vice versa.

Table 6.2: Case Study Example Preferred Proximity

Facility #	1	2	3	4	5	6	7	8	9	10	11	12
1		5	2	1	1	2	2	1	1	1	1	2
2			2	1	1	1	1	2	2	2	2	2
3				3	3	1	4	2	2	2	3	2
4					3	2	2	2	2	2	2	2
5						1	4	1	1	1	3	5
6							2	2	4	2	2	2
7								2	2	2	2	2
8									1	5	3	2
9										2	2	2
10											2	2
11												2
12												

Preference Rating	1	2	3	4	5
Preference Description	Undesirable	Ordinary	Important	Especially Important	Absolutely Necessary

CONSITPLAN was run several times for this example with the optimization parameters in Table 6.3. Li and Love (1998) have recommended an initial solution pool of 100 with 90 iterations to get satisfactory results using Genetic Algorithms. Most of the past research studies have used a 1% probability of mutation. The same is used for optimization here.

Table 6.3: Case Study Example Optimization Parameters

Optimization Parameter	Case Study Value
Number of Iterations	90
Number of Solutions	100
Percentage of Probability of Mutation	1%

The optimal layouts suggested by CONSITEPLAN during the several runs are shown in the Figures 6.2, 6.3, 6.4, and 6.5.

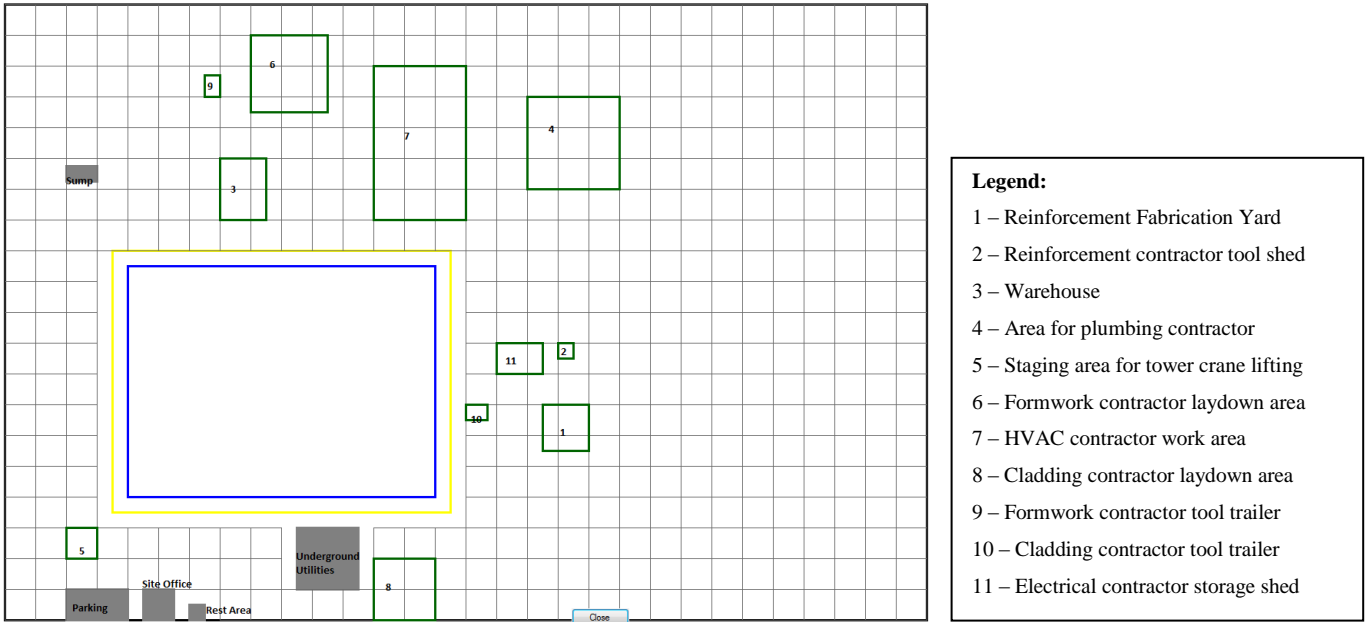


Figure 6.2: CONSITEPLAN – Site Utilization Plan 1

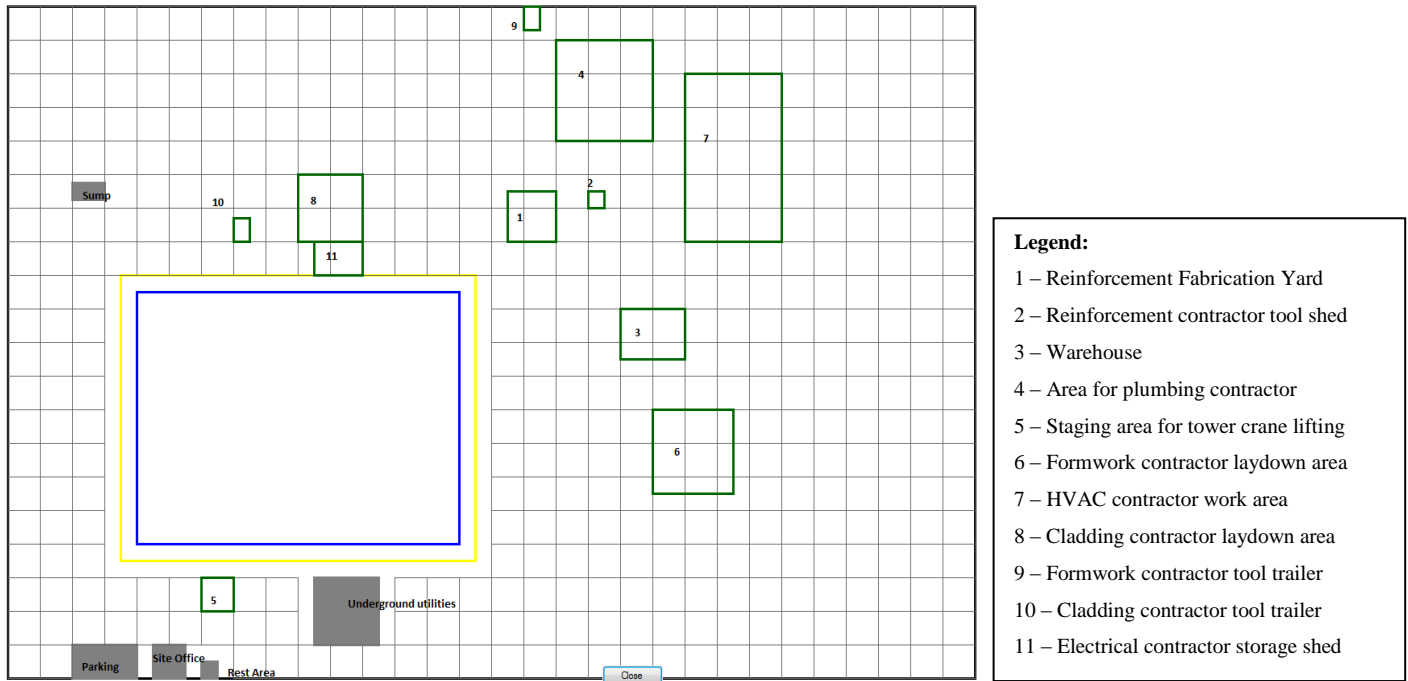


Figure 6.3: CONSITEPLAN – Site Utilization Plan 2

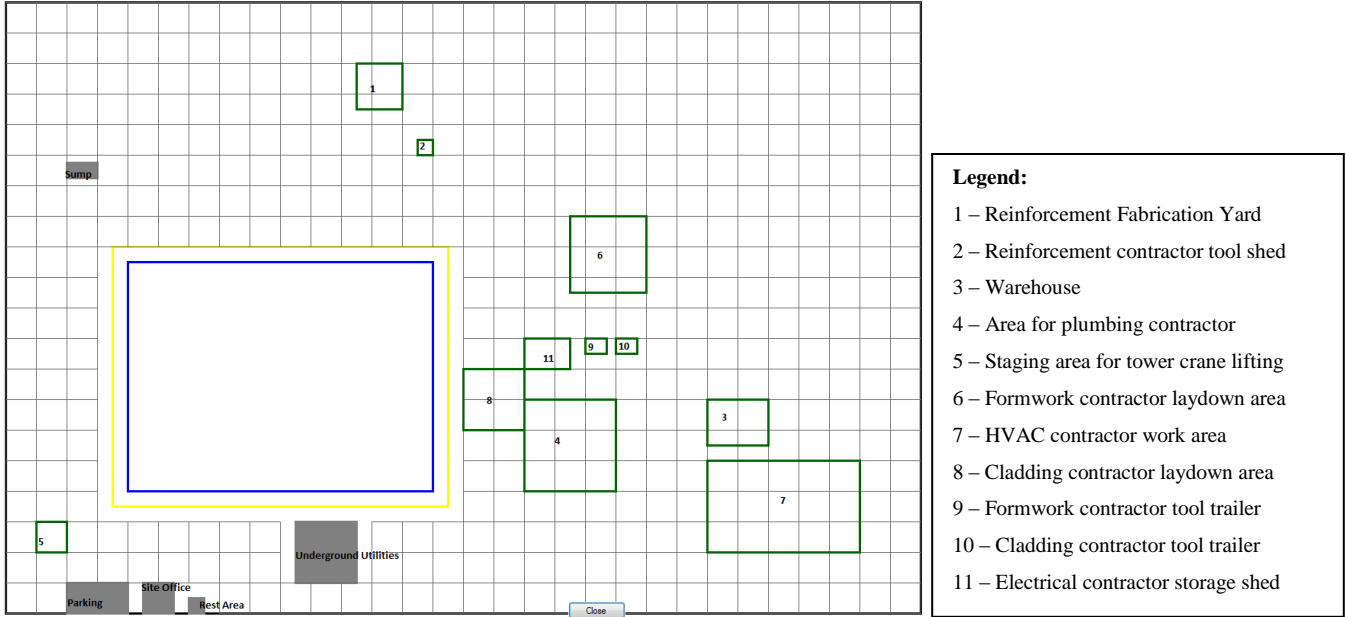


Figure 6.4: CONSITEPLAN – Site Utilization Plan 3

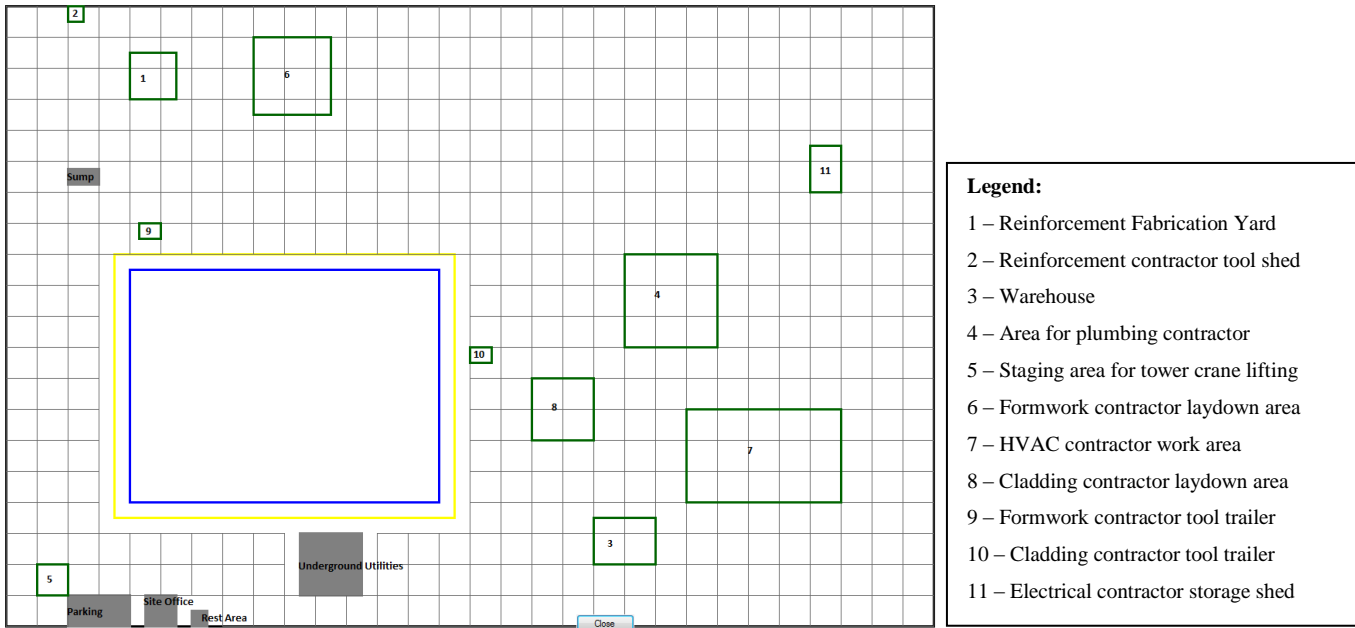


Figure 6.5: CONSITEPLAN – Site Utilization Plan 4

On analyzing the layouts created by the tool, one can notice that the location of the formwork contractor's trailer is not close to their laydown area. Upon examining the proximity preference, it can be seen that the user stated that their proximity is "Especially Important". One of the reasons for the anomaly in layouts 2 and 4 could be the effect of the proximity relationships between the tool trailer and rest of the facilities. Thus, it becomes clear that careful evaluation of the required proximity is especially important in achieving a useful utilization plan.

6.3 Analysis of the impact of Optimization Parameters

To analyze the impact of optimization parameters on the objective function value and computation time, CONSITEPLAN was run for several iterations for the above example with differing optimization parameters. The following section discusses about the results.

6.3.1 Impact of the Number of Iterations on Objective Function Value

To study the impact of number of iterations on the objective function value, CONSITEPLAN is run for several times with different solution iteration values. Figure 6.6 illustrates the variation in objective function value based on the number of solution iterations. As plotted, as the number of solutions iteration increases, the objective function value decreases (improved solution quality). But if the number of iterations is increased there seems to be disruption in the objective function value. Literature review suggested that a number of solution iterations between 90 to 100 could yield better optimization results. In this study, as a solution iteration of 90 yields the best objective function value, suggestion from the literature holds good.

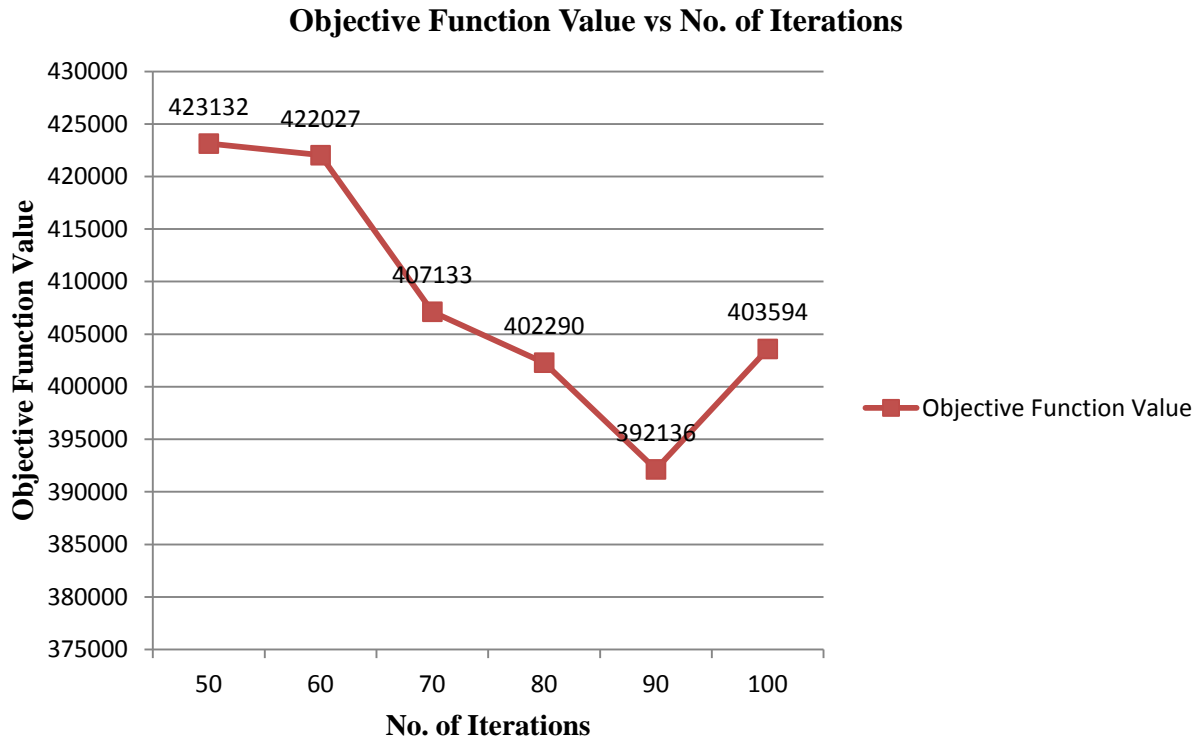


Figure 6.6: Objective function value vs. No. of iterations

6.3.2 Impact of the Percentage of Probability of Mutation on Objective Function Value

The mutation component of the genetic algorithms is designed to enable the escape of a solution from a local optimum. As discussed earlier, the typical value of mutation is 1%, i.e. 1% of the solutions are randomly changed. In order to impact of this on the quality of the resulting solutions, CONSITEPLAN was run for several times with different percentage of probability of mutation values, and Figure 6.7 illustrates the results. It can be seen that up to 5% of mutation probability, there is a decrease in objective function value with an increase in percentage of probability of mutation. However, with higher percentage of probability of mutation (i.e., greater than 5%), the objective function value does not seem to have a continuous trend, as illustrated in Figure 6.7.

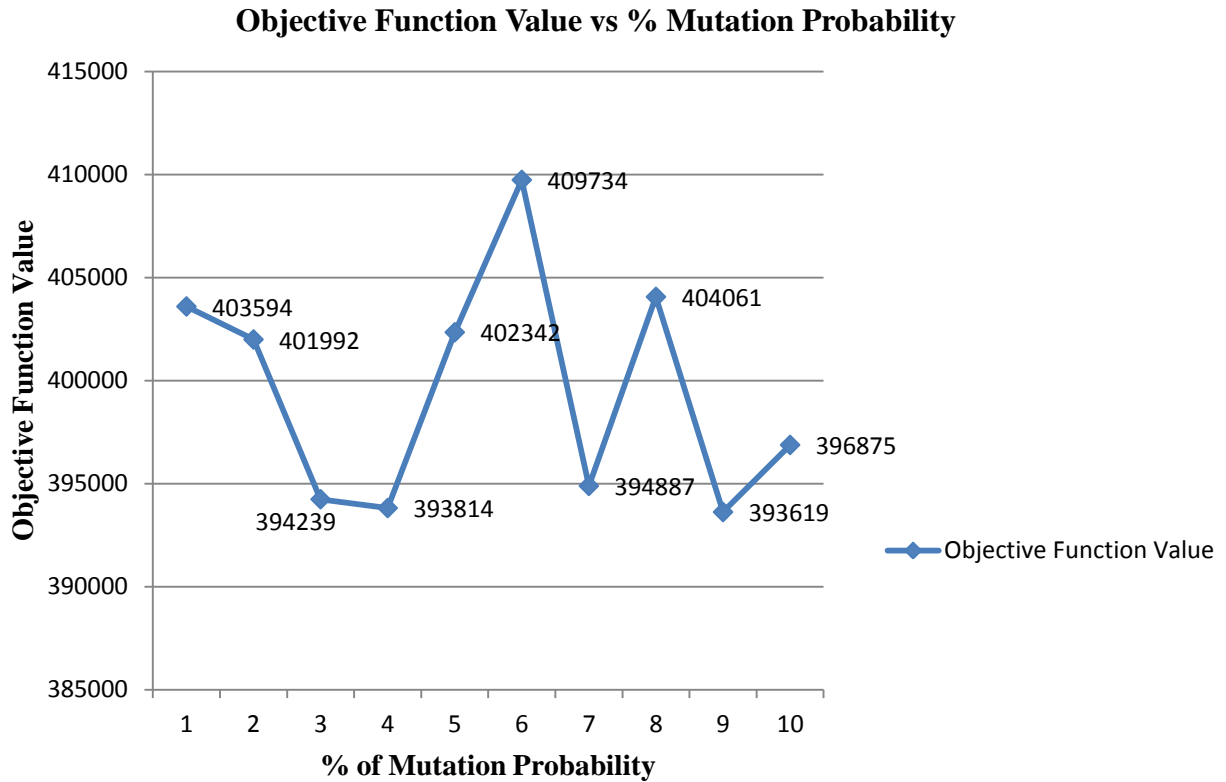


Figure 6.7: Objective function value vs. Percentage of probability of mutation

6.3.3 Impact of the selected Grid Size on the Objective Function Value

It was hypothesized that smaller size of grids will improve the quality of the solution while impacting the computation time negatively. In order to evaluate this, CONSITEPLAN was run for several times with different grid size values, and the results are plotted in Figure 6.8. As it can be seen in Figure 6.8, an increase in grid size value results increases the objective function value (less optimal solutions). This is because of the fact that as the grid size increases, the number of potential available locations decreases. But, a very low grid size value (as compared to the site space) seems to have disruptive impact in the objective function value, as with the grid size 10.

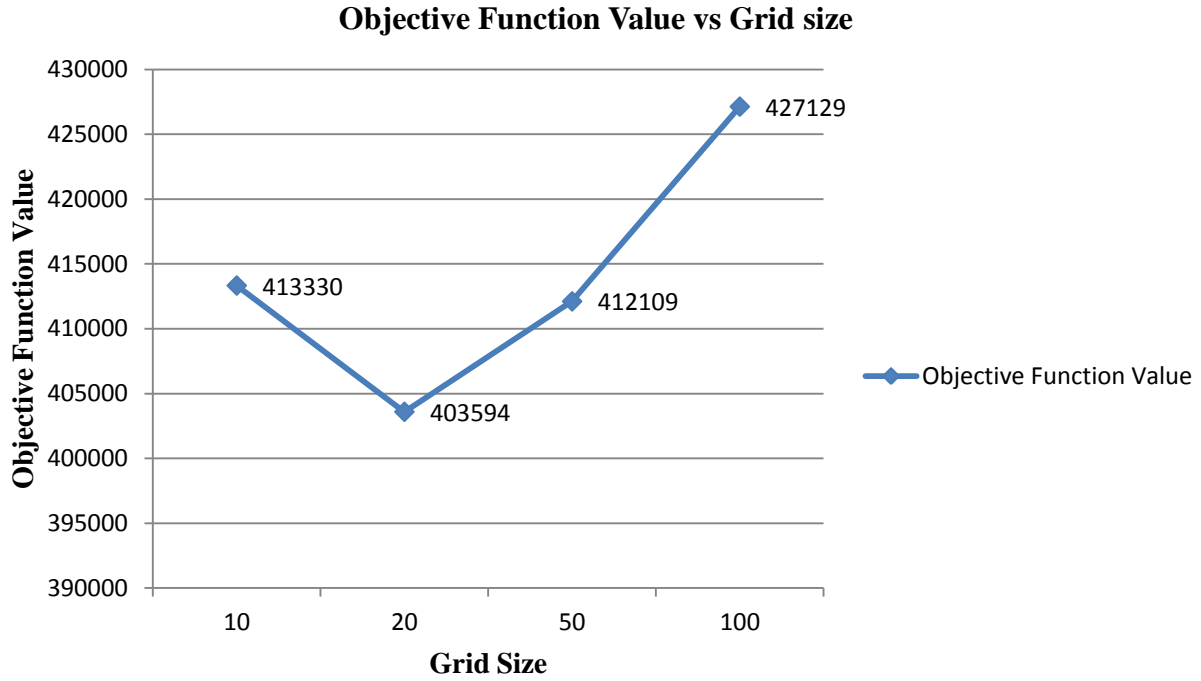


Figure 6.8: Objective function value vs. Grid size

6.3.4 Number of Iterations, Grid Size vs Computation Time

Figures 6.9 and 6.10 depict the impact of number of solution iterations and grid size on the computation time respectively. The computation time increases with an increase in number of solution iterations and grid size. The optimization was carried out using a Windows 7 PC powered by an Intel Core2Quad Q9400 @ 2.66 GHz with 4GB memory.

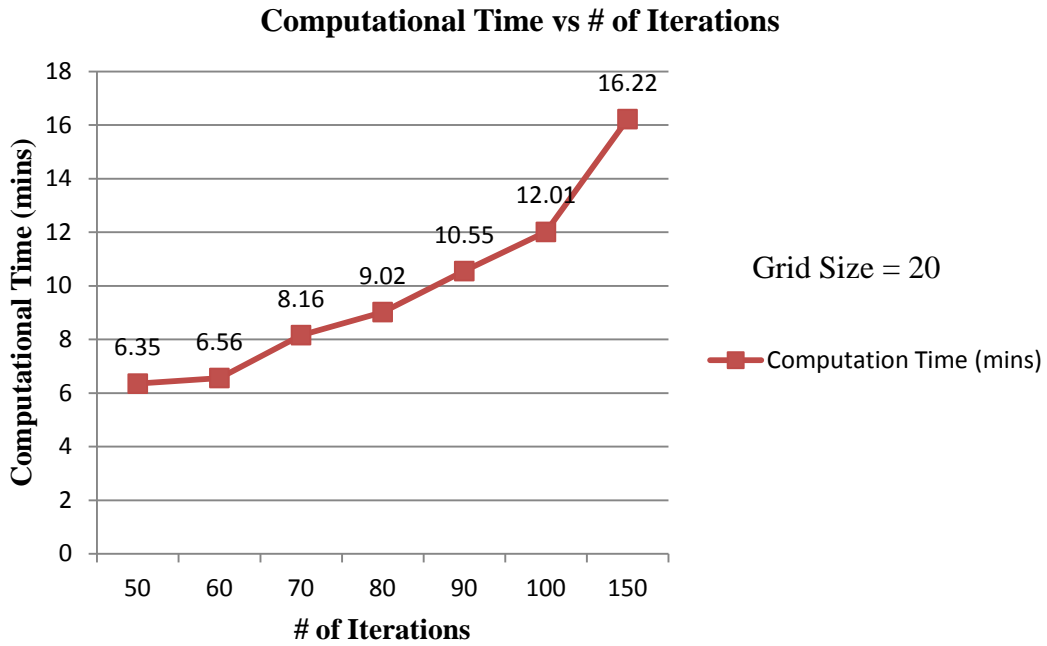


Figure 6.9: Number of solution iterations vs. Computation time

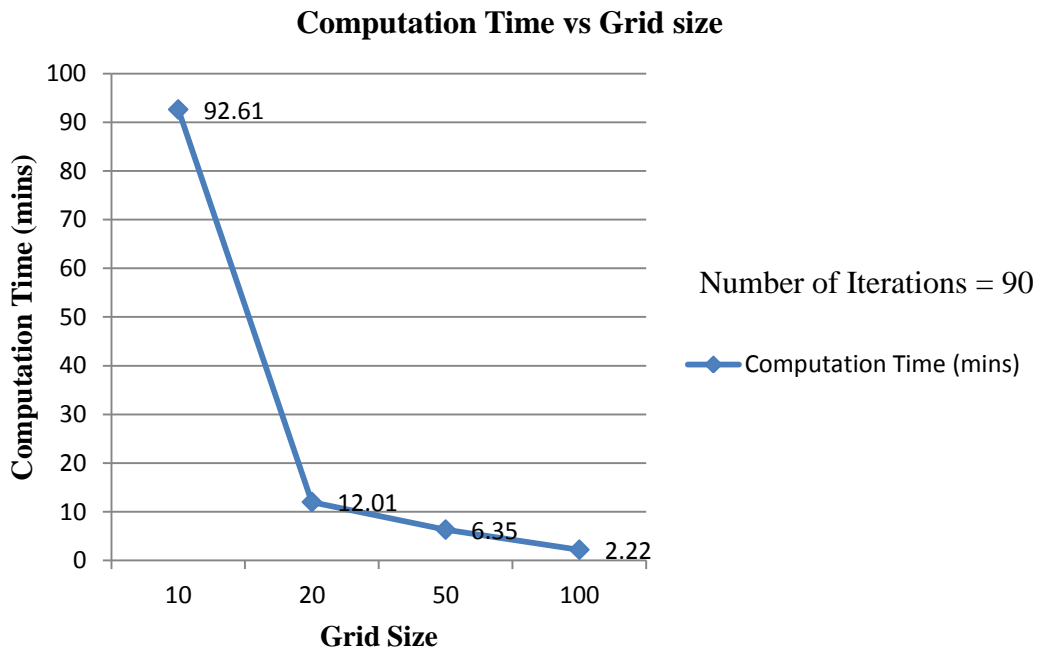


Figure 6.10: Grid size vs. Computation time

6.4 Summary

The illustrative examples presented in published literature don't specify many of the parameters used in CONSITEPLAN. The dimensions of site, permanent structure are notably absent in all the studies reviewed in literature. None of the studies provide the option of considering offsite areas, tower crane accessibility for facilities or unusable areas on the site. Hence, the algorithm couldn't have realistically been implemented on these examples. Hence, CONSITEPLAN was applied to a hypothetical project. The tool was also run for several iterations for the above example with differing optimization parameters to analyze the impact of optimization parameters on the objective function value and computation time. The results suggest that the number of solution iterations of 90, with 1% mutation probability, grid size as 20, and 100 initial solutions yield better solutions. Also, the analysis results generically suggest the following.

- Optimality of solutions increase with increase in number of iterations
- Optimality of solutions decrease with increase in grid size
- Up to 5% of mutation probability, optimality of the solutions increases with increase in percentage of mutation probability. However, this trend is not followed for higher mutation probability values (i.e., greater than 5%).
- Computation time increases with increase in number of iterations
- Computation time decreases with increase in the grid size

Chapter 7: Summary and Scope for Future Research

7.1 Introduction

Construction Site Utilization Planning, which primarily deals with accommodating the site-level temporary construction facilities that are used to support the construction process, is a vital aspect of construction operation planning which greatly contributes in achieving the project objectives. Typical temporary facilities in a construction project include material laydown areas, fabrication yards, concrete batching plants, equipment storage areas, and temporary site offices used by various contractors. A poor planning of laying out the temporary facilities could potentially lead to work delays, temporary material storage only to move it subsequently, multiple handling of materials resulting in reduced labor productivity, schedule delays, loss of time and money, and unsafe working conditions. For the construction process to flow smoothly and efficiently, utilization of site space need to be determined with great care.

Many construction projects carried out in urban areas have very limited working space at the site and don't have sufficient space for positioning all the temporary facilities required for construction. Therefore, planning is essentially aimed at effectively utilizing the available smaller spaces within the site boundaries to execute the construction activities efficiently. In construction projects on large sites, the project managers might tend to not focus on site utilization planning at the beginning of construction phase due to abundant availability of space, and the layout of temporary facilities is done on an ad-hoc basis with no concern for optimality or layout efficiency. This results in increased costs of transportation of materials and inefficient

labor movement. For example, in extreme cases, decision on where to store the materials might only be made after the delivery of the materials to the project site. Evidently, the careful planning of the layout of temporary facilities has a bearing on the success of construction projects whether the site is confined with limited space, or it is a very large site where travel between various facilities could be time consuming.

The layout planning problem in construction can be conceptualized as a multi-objective problem where an optimal layout to position the temporary facilities need to be identified, which incorporates the project requirements and constraints while simultaneously minimizing the cost of resource flow and improving construction safety. The layout of temporary facilities is unique in every construction project due to the changing process requirements on account of different site configurations and project specific conditions. Moreover, the layout planning problem in construction also has a temporal aspect to it, in which a particular layout may need to change over the course of the project to meet the unique requirements of the construction process. In addition, several key, unique considerations such as crane access to certain facilities also need to be accounted for in solving this problem. This further compounds the complexity of the problem.

In attempt to address this complex, multi-objective problem, a significant amount of research effort has been directed world-wide in the last two decades at developing algorithms for identifying optimal construction site layouts. Due to the combinatorial and computationally intensive nature of the problem, researchers have increasingly used meta-heuristics solutions inspired by evolution and other natural phenomena. These include population based strategies such as Genetic Algorithms (GAs), Ant Colony Optimization, Particle Swarm Optimization, and trajectory based strategies such as Simulated Annealing, Tabu lists etc. GAs is the most commonly used optimization strategy in recent research in this area. GAs is favored by most

researchers because of its capacity to move randomly in the fitness landscape from one feasible solution to another without getting trapped into local optima.

Layout planning in construction can be conceptualized as, i) making discrete one to one assignments where facilities are assigned to previously defined locations ii) assigning facilities to a continuous space. The first definition, where n facilities are to be located in n predefined locations, is limited in scope, and it can be effectively used only in situations where space is limited and the number of available locations is predetermined, which may not be applicable in many projects.

CONSITPLAN

While many tools and algorithms can be found in the review literature, the practical applicability of these algorithms is fairly limited due to quadratic assignment (predetermining locations and then simply assigning facilities to those locations), distance measurement (using Euclidean or Manhattan distance measurement formulas which can have severe limitations), lack of consideration of the availability of offsite spaces for locating temporary facilities, lack of a mechanism to implement a variable transportation constraint for certain facilities (location, radius of a crane, access to certain temporary facilities), lack of a mechanism to explicitly ensure that facilities are not located at certain locations within the site, and lack of flexibility in specifying a certain size and shape (rectangular/square) for the proposed temporary facility.

In this research, CONSITPLAN, a windows based tool was developed to overcome these limitations of the current layout planning tools. The main objective behind developing CONSITPLAN was to create a tool which addresses the practical requirements of layout

planning without sacrificing the mathematical robustness of the underlying optimization algorithm. The main features of the tool are as follows:

1. Space conceptualization

Unlike requiring predefined locations for temporary facilities, CONSITEPLAN conceptualizes the site as continuous domain, which it divides into smaller grids and uses the available grids as discrete options for the layout. Thus, the entire available area is considered for layout purposes. The tool gives the user flexibility in choosing the size of the grid that they would like to use for layout purposes. Thus, it is flexible enough to handle very small or very large construction sites.

2. Robust Optimization Engine

CONSITEPLAN uses genetic algorithms as an optimization engine to search the fitness landscape for a good layout based on the project requirements, site geometry, permanent facility location and dimensions, and as well as other constraints that the user might want to impose. Genetic Algorithms allow for a very transparent implementation of the constraints to locating the temporary facilities.

3. Improved Efficiency of the Genetic Optimization Algorithm

Typically, the optimization algorithms found in literature use the entire site space (including the final structure) to locate temporary facilities and impose a penalty on any solution that uses the occupied areas for locating temporary facilities. CONSITEPLAN improves the efficiency of the underlying genetic algorithm by initially creating a list of locations that are not available for laying out temporary facilities and then ensuring that the random process of solution generation doesn't consider these locations as options. This could

potentially reduce number of infeasible solutions in a great manner, which would have to be eliminated in the optimization process.

4. Distance Measurement

The distance measurement algorithms conventionally used in research literature, i.e. Euclidean (diagonal distance between the facilities using Pythagorean formula) and Manhattan (sum of absolute differences of the coordinates of the points under consideration), have significant limitations on account of the obstruction placed by the excavated area or the superstructure, which may realistically not allow for such movement (direct or rectilinear). As the objective function in identifying an optimal construction site layout involves minimizing the distance travelled, the method used to measure distance between facilities plays a critical role in giving validity to the results. To overcome these limitations, an improvised distance measurement algorithm is developed in this study. This algorithm accounts for the obstruction caused by the permanent facility or the excavated areas for foundation and measures the likely travel distance among the construction facilities.

5. Modeling offsite temporary facilities

Another unique feature of CONSITEPLAN is that it allows the user to specify the use of temporary facilities at offsite locations. The distance between these offsite facilities and the construction site (provided by the user) and the facility's relationship with other temporary facilities is used as a factor in the optimization process.

6. Modeling Unusable Areas

On construction sites, some areas could potentially be unusable on account of underground utility work, storm drainage connections etc. The size, geographic, and temporal location of these areas may change from project to project. Ideally, the project

manager may not want to locate a temporary facility on such a location because it may require removing and reinstalling that temporary facility at a later date. CONSITEPLAN allows for the definition and location of unusable areas at the site. A constraint is added to the optimization algorithm to ensure that the temporary facilities are not located in these unusable areas.

7. Modeling Crane Access

At a construction site, a tower crane may be used or a large crane may be permanently stationed to perform the erection / lifting activities. The project manager may want to ensure that certain temporary facilities, e.g. reinforcement steel fabrication area are accessible by the crane to eliminate the inefficiencies caused by double handling of materials. CONSITEPLAN allows the user to define the crane location, crane radius and the facilities that will be accessed by the crane. These factors form an additional constraint to the optimization process.

8. Safety Considerations

Safety is one of the important factors in construction projects and is widely accepted that considering safety factors in developing a site utilization plan has a notable impact in improving the site safety. For example, OSHA recommends not to store any materials within the free fall zone of the permanent structure, and to maintain recommended buffer distance around the permanent structure, which is a safety factor that could be considered while creating a site utilization plan. CONSITEPLAN has been designed with the option to request the user's preference on the preferred safety boundary around the building structure, and not utilizing that space for identifying optimal layouts.

9. Dynamic Planning

Construction site utilization planning problem can be defined as a dynamic problem, where different facilities may be required during different phases of time during the life cycle of the project. Hence, it is quite possible that different site utilization plans are being used during different time spans of a project. This tool broadly breaks the project down in two phases, sub-structure construction phase and super-structure construction phase, and provides the flexibility of generating multiple site utilization plans based on the project requirements.

10. Ability to seamlessly interact with REVIT BIM files

CONSITPLAN provides initial interfaces to extract essential geometric data about the site and permanent structure using the application programming interface (API) provided by Autodesk REVIT® products. This lays a foundation for exciting future developments in the development of site utilization plans.

7.2 Conclusion

The site utilization planning problem has been subject to a significant body of work by well-respected researchers over the last two decades. This study addresses the layout planning problem in construction in a very novel and sophisticated manner by developing the tool ‘CONSITPLAN’. CONSITPLAN has been developed in a modular fashion on a purpose to allow for improvements and additions in the future. This research lays a foundation for future development of a site utilization planning tool which could be useful to the construction community. The efficiencies envisaged through the use of this tool could help the owner and the contractor to save money, while simultaneously improving safety at the constructions site. There is ample scope for further improving this tool. This tool can be improved in the following ways.

Currently, the tool limits the project phase to two broader phases (sub-structure and super-structure). Future research shall target to allow for definition of additional user defined phases. In construction projects, it is quite possible to use the additional covered areas (that are part of the permanent structure) for short spells for temporary fabrication and material storage, which tend to become available for contractors with the progress of construction. This aspect has not been addressed in the current research and can be extended for future research.

In construction projects that use tower cranes, the lifting capacity of the tower cranes can vary based on the distance accessed. Currently, this attribute is not included in the tool, and the tool assumes a constant lifting capacity for any distance accessed within the reachable radius of the crane. The tool can be enhanced to facilitate this attribute which enables the option of providing varying reachable distance based on the weight of the accessed temporary facility. The tool can also be provided with an interactive layout plan, which allows the user to modify the layout provided by the tool as per his/her preference, based on which the proximity rating among the facilities can be updated and the optimality of the modified layout can be evaluated. The tool has some basic capabilities to interact with the Autodesk REVIT®. This capability can be enhanced even further in the future as more robust programming APIs become available. Ideally, CONSITEPLAN could become independent of BIM products used i.e., it could allow for seamless interaction with TEKLA, VICO BIM models as well. The seamless cost, estimating, scheduling, and geometric data integration into a centralized BIM model could provide even more opportunities of making CONSITEPLAN user friendly while staying mathematically robust.

References

- Anumba, C., and Bishop, G. (1997). "Importance of safety considerations in site layout and organization." *Canadian Journal of Civil Engineering*, 24, 229-236.
- Arikaran, P., Jayabalan, D. V., and Senthilkumar, R. (2010). "Analysis of Unequal Areas Facility Layout Problems." *International Journal of Engineering*, 4(1), 44-51.
- Balakrishnana, J., Chengb, C.-H., and Wongb, K.-F. (2003). "FACOPT: A user friendly FACility layout OPTimization system." *Computers & Operations Research*, 30(11), 1625-1641.
- Buffa, E. S., Armour, G. C., and Vollmann, T. E. (1964). "Allocating Facilities with CRAFT." *Harvard Business Rev.*, 42, 136-159.
- Cheng, M. Y., and O'Connor, J. T. (1997). "Arcsite - Enhanced GIS for construction site layout." *Journal of Construction Engineering and Management*, 122(4), 329-336.
- Deb, K. (2009). *Multi-objective optimization using evolutionary algorithms*, Wiley.
- Dorigo, M., and Gambardella, L. M. (1997). "Ant colony system: a cooperative learning approach to the traveling salesman problem." *IEEE Computational Intelligence Society*, 1(1), 53-66.
- El-Rayes, K., and Said, H. (2009). "Dynamic Site Layout Planning Using Approximate Dynamic Programming." *Journal of Computing in Civil Engineering*, 23(2), 119-127.
- Elbeltagi, E., Hegazy, T., and Eldosouky, A. (2004). "Dynamic layout of construction temporary facilities considering safety." *Journal of Construction Engineering and Management*, 130(4), 534-541.

- Gholizadeh, R., Amiri, G. G., and Mohebic, B. (2010). "An alternative approach to a harmony search algorithm for a construction site layout problem." *Canadian Journal of Civil Engineering*, 37(12), 1560-1571.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley.
- Hansen, P., and Mladenovic, N. (1997). "Variable neighborhood search for the p-median." *Location Science*, 5(4), 207-226.
- Harmanani, H., Zouein, P., and Hajar, A. "An Evolutionary Algorithm for Solving the Geometrically Constrained Site Layout Problem." *Computing in Civil and Building Engineering*, 1442-1449.
- Hegazy, T., and Elbeltagi, E. (1999). "EvoSite - Evolution-Based Model for Site Layout Planning." *Journal of Computing in Civil Engineering*, 13(3), 198-206.
- Heller, W. R., Sorkin, G., and Maling, K. (1982). "The Planar Package Planner for System Designers." *IEEE*, 20(6), 253-260.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*, Ann Arbor: University of Michigan Press.
- Holland, J. H. (1992). *Adaptation in natural and artificial systems*, MIT Press Cambridge.
- Jo, J. H., and Gero, J. S. (1998). "Space Layout Planning Using an Evolutionary Approach." *Artificial Intelligence in Engineering*, 12(3), 149-162.
- Kennedy, J., and Eberhart, R. C. (2001). *Swarm Intelligence*.
- Kirkpatrick, S., C.D. Gelatt Jr., and Cecchi, M. P. (1983). "Optimization by simulated annealing." *Science*, 220(4598), 671-680.

- Kozminski, K., and Kinnen, E. "An algorithm for finding a rectangular dual for a planar graph for use in area planning for VLSI integrated circuits." *Proceedings of the 21st Design Automation Conference*, Albuquerque, New Mexico, United States, 655 - 656.
- Lam, K.-C., Ning, X., and Lam, M. C.-K. (2009). "Conjoining MMAS to GA to Solve Construction Site Layout Planning Problem." *Journal of Construction Engineering and Management*, 135(10), 1049-1057.
- Li, H., and Love, P. E. D. (1998). "Site level facilities layout using genetic algorithm." *Journal of Computing in Civil Engineering*, 12(4), 227-231.
- Li, H., and Love, P. E. D. (2000). "Genetic search for solving construction site-level unequal-area facility layout problems." *Automation in Construction*, 9(2), 217-226.
- Liang, L. Y., and Chao, W. C. (2008). "The strategies of tabu search technique for facility layout optimization." *Automation in Construction*, 17(6), 657-669.
- Liggett, R. S. (2000). "Automated facilities layout: past, present and future." *Automation in construction*, 9(2), 197-215.
- Mahachi, J. (2001). "An integrated approach to critical time-space scheduling." *Construction Informatics Digital Library*.
- Mawdesley, M. J., and Al-Jibouri, S. H. (2003). "Proposed genetic algorithms for construction site layout." *Engineering Applications of Artificial Intelligence*, 16(5-6), 501-509.
- Mawdesley, M. J., Al-jibouri, S. H., and Yang, H. (2002). "Genetic algorithms for construction site layout in project planning." *Journal of Construction Engineering and Management*, 128(5), 418-426.
- Mincks, W. R., and Johnston, H. (2010). *Construction jobsite management*, Delmar Cengage Learning.

- Montreuil, B., Ratliff, H. D., and Goetschalckx, M. (1987). "Matching based interactive facility layout." *IIE Trans*, 19(3), 271-279.
- Nelder, J. a., and R.Mead. (1965). "A simplex method for function minimization." *Comput. J*, 7(1), 308-313.
- Ning, X., Lam, K.-C., and Lam, M. C.-K. (2010a). "A decision-making system for construction site layout planning." *Automation in Construction*, in press.
- Ning, X., Lam, K.-C., and Lam, M. C.-K. (2010b). "Dynamic construction site layout planning using max-min ant system." *Automation in Construction*, 19(1), 55-65.
- Osman, H. M., and Georgy, M. E. "Layout planning of construction sites considering multiple objectives: A goal-programming approach." *Construction Research Congress*.
- Osman, H. M., Georgy, M. E., and Ibrahim, M. E. (2003). "A hybrid CAD-based construction site layout planning system using genetic algorithms." *Automation in Construction*, 12(3), 749-764.
- Palumbo, A. (2010). "Safety in Design: Enhancing Construction Safety by Implementing Safety in the Design Phase." 1-32.
- Parsopoulos, K. E., and Vrahatis, M. N. (2002). "Recent approaches to global optimization problems through Particle Swarm Optimization." *Natural Computing*, 1(2-3), 235-306.
- Pitsoulis, L. S., and Resend, M. G. C. (2002). "Greedy randomized adaptive search procedure." In *Handbook of Applied Optimization*, Pardalos and R. M. Eds, eds., Oxford University Press, 168–183.
- Rajasekharan, M., Peters, B. A., and Yang, T. (1998). "A Genetic Algorithm For Facility Layout Design In Flexible Manufacturing Systems." *International Journal of Production Research*, 36(1), 95–110.

- Haupt, L. R., and Haupt, S.E. (2004). *Practical Genetic Algorithms*, Wiley.
- Rosenblatt, M. J. (1986). "The Dynamics of Plant Layout." *Management Science*, 32(1), 76-86.
- Sanad, H. M., Ammar, M. A., and Ibrahim, M. E. (2008). "Optimal construction site layout considering safety and environmental aspects." *Journal of Construction Engineering and Management*, 134(7), 536-544.
- Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*, Wiley.
- Singh, S. P. (2006). "A review of different approaches to the facility layout problems." *International journal of advanced manufacturing technology*, 30(5-6), 425-433.
- Stützle, T. (1999). "Local Search Algorithms for Combinatorial Problems---Analysis, Algorithms and New Applications."
- Szymberski, R. T. (1997). "Construction project safety planning." *Engineering and papermarks conference*, 8(11), 69-74.
- Tam, C. M., Tong, T. K. L., Leung, A. W. T., and Chiu, G. W. C. (2002). "Site Layout Planning Using Nonstructural Fuzzy Decision Support System." *Journal of Construction Engineering and Management*, 128(3), 220-231.
- Tate, D. M., and Smith, A. E. (1995). "A genetic approach to the quadratic assignment problem." *Computers & Operations Research*, 22(1), 73-83.
- Tommelein, I. D., and Zouein, P. P. (1993). "Interactive dynamic layout planning." *Journal of Construction Engineering and Management*, 119(2), 266-287.
- Voudouris, C., and Tsang, E. (1999). "Guided local search." *European Journal of Operations Research*, 113(2), 469-499.

- Wong, C. K., Fung², I. W. H., and Tam³, C. M. (2010). "Comparison of Using Mixed-Integer Programming and Genetic Algorithms for Construction Site Facility Layout Planning." *Journal of Construction Engineering and Management*, 136(10), 1116-1128.
- Yeh, I.-C. (1995). "Construction site-layout using annealed neural network." *Journal of Computing in Civil Engineering*, 9(3), 201-208.
- Zhang, H., and Wang, J. Y. (2008). "Particle Swarm Optimization for Construction Site Unequal-Area Layout." *Journal of Construction Engineering and Management*, 134(9), 739-748.
- Zouein, P. P., Harmanani, H., and Hajar, A. (2002). "Genetic algorithm for solving site layout problem with unequal size and constrained facilities." *Journal of Computing in Civil Engineering*, 16(2), 143-151.
- Zouein, P. P., and Tommelein, I. D. (1999). "Dynamic layout planning using a hybrid incremental solution method." *Journal of Construction Engineering and Management*, 125(6), 400-408.