CAD-BASED DYNAMIC LAYOUT PLANNING OF CONSTRUCTION SITES USING GENETIC ALGORITHMS

by

HESHAM MAGED OSMAN

A Thesis Submitted to the Faculty of Engineering at Cairo University in Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

in

CIVIL ENGINEERING (STRUCTURES)

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT

NOVEMBER 2002

CAD-BASED DYNAMIC LAYOUT PLANNING OF CONSTRUCTION SITES USING GENETIC ALGORITHMS

by

HESHAM MAGED OSMAN

A Thesis Submitted to the Faculty of Engineering at Cairo University in Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

in

CIVIL ENGINEERING (STRUCTURES)

Under the Supervision of

Dr. Moheeb El-Saeed Ibrahim
Professor of Construction Engineering & Management
Cairo University

Dr. Maged Ezzat Georgy
Assistant Professor,
Structural Engineering Department
Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT

NOVEMBER 2002

CAD-BASED DYNAMIC LAYOUT PLANNING OF CONSTRUCTION SITES USING GENETIC ALGORITHMS

by

HESHAM MAGED OSMAN

A Thesis Submitted to the Faculty of Engineering at Cairo University in Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

in

CIVIL ENGINEERING (STRUCTURES)

Approved by the Examining Committee

Prof. Dr. Moheeb El-Saeed Ibrahim, Thesis Main Advisor

Dr. Mahmoud Abdel-Salam Taha

Dr. Mohamed Abdel-Lateef Bakry

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT

NOVEMBER 2002

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Professor Moheeb El-Saeed my main advisor for his continuous encouragement, valuable ideas and lengthy discussions. My deepest thanks and extreme gratitude are due to Dr. Maged Georgy, my advisor. It goes without saying that this thesis would have not reached this level if it were not for the immense efforts and time spent throughout the course of this research.

Much thanks are due to Eng. Mohammed Saad, for his valuable time and assistance in providing data and information necessary for this research's case study.

Last but not least, my greatest thanks go to my beloved family. I cannot help but feel immense gratitude towards my parents for encouraging and supporting me in all means possible throughout the course of my life.

ABSTRACT

One of the important project resources that has been overlooked during the planning phases of most construction projects is site space. In some projects on-site space can be as crucial a resource as the traditional construction resources (time, capital, labor, equipment and material). In highly congested sites, space becomes a very scarce resource that needs to be carefully planned and efficiently utilized. On the other hand, in large sites having abundant space availability, the proper positioning of site facilities with respect to each other will greatly influence material handling and travel costs.

In a broad sense, layout planning is concerned with the placement of temporary facilities (e.g. Storage areas, fabrication yards, caravans, etc…) within the boundaries of the construction site with the goal of attaining one or several layout objectives. Two layout planning approaches are commonly found in the literature; namely static layout planning and dynamic layout planning. A layout is termed "static" if effort put in planning yields only one site layout that will span the entire project duration. Creating layouts that change over time as construction progresses is termed dynamic layout planning.

This research presents a fully automated computer system for performing the site layout planning task at the dynamic level. The system integrates the powerful graphical capabilities of CAD systems with the intricate search and optimization abilities of genetic algorithms for the purpose of solving the site layout problem. Modeling the continuously developing construction site is made possible via dynamic site layout planning, thus creating several layouts that change over time as construction progresses. Two approaches are proposed to deal with the dynamic layout problem, namely the critical phase approach and the mini-min approach. These approaches aim to overcome the shortcomings found in the traditional dynamic layout techniques.

The automated system is implemented via 6.0. The programmable features of AutoCAD™ are utilized to capture the geometrical details of the site layout and to represent the final solution graphically. A 4-phase project is used to demonstrate the system's capabilities. A 24,000 m^2 swimming pool complex under construction is chosen as a case study to validate the system's performance. A comparison is performed between the existing layout and that produced by the system. The system-generated layout achieved savings of nearly 25% in total layout costs compared to the existing site layout.

1 INTRODUCTION

1.1 BACKGROUND

The effective and efficient management of construction resources is the essence of success for any construction project. Traditionally, researchers and industry professionals identify the five main construction resources to include, time, capital, labor, equipment and material. Recently, attention has been given to information as being a vital resource in construction operations.

One of the most important construction resources that has been for long overlooked is *space*. Site space can – in some projects – be as crucial a resource as the traditional construction resources mentioned. In highly congested sites, space becomes a very scarce resource that needs to be carefully planned and efficiently utilized. On the other hand, in large sites having abundant space availability, the positioning of site facilities with respect to each other will greatly influence the efficiency of work flow.

Regardless of site dimensions and level of congestion, the need for careful planning of construction site layouts is evident. In practice, the task of site layout is usually performed through common sense and the adoption of past layouts to present projects. A survey conducted in the U.K. in the late 90's indicated that only 13% of contractors use computer methods or expert systems to assist in the site layout planning task (Markhomihelakis ,1997).

1.2 PROBLEM STATEMENT

The task of site layout consists of identifying the facilities needed to support construction operations, determining their size and shape and positioning them within the boundaries of the available on-site areas. Examples of these facilities include offices and tool trailers, parking lots, warehouses, batch plants, maintenance areas, fabrication yards or buildings, staging areas, and lay-down areas (Tommelien et al, 1992a).

Space is considered secondary to time and money because it is more difficult to model and the payoffs from modeling are not readily apparent. First of all it is difficult to represent and reason about space. Secondly it is difficult to precisely determine the space required for conducting construction operations that are needed by resources such

as material and equipment. Third, it is difficult to value the adequacy of a space to accommodate a resource. Last, spatial variables are three dimensional, in contrast with time and money which are scalar. All too often the costs associated with handling these complexities are prohibitive, so model simplifications or abstractions must be introduced at the expense of loosing interesting detail (Tommelien & Zouein 1993).

In the construction industry, the cost of site planning is typically charged to project overhead and is not treated as a direct cost or reimbursable item. Also the competitive bidding structure of construction and the bidding firm's need to keep overhead down during bidding process often gets in the way of providing project managers adequate staff and time to plan site layouts early (Cheng & O'Connor 1996).

Neglecting site layout planning during the early planning stages can lead to unsuitable layouts that need correction. Correcting a mistake costs much more than preventing it in the first place. The emergence of an unsuitable layout must not always occur during the early phases of a project, it is during the late phases (due to the vast changes that have occurred) that a layout might seem incompetent of achieving the site requirements at that time.

Considering the dynamic nature of construction projects and its direct reflection on site requirements in general further complicate the layout process. Site facilities constrain one another in the process of determining their final position. Gasoline or natural gas containers must be kept a minimum distance away from buildings or oxygen tanks. Physical resources (e.g., trailers) cannot occupy space that is occupied by other physical resources, that is they cannot overlap. Constraints themselves vary over time. A lay down area may provide space to store precast members while a structure is being erected, and space to accommodate machinery later. Interactions between resources also determine the quality of positions, which too can vary over time. A loader and filling material interact during backfilling, so they must be positioned as close as possible to each other to minimize travel time. Upon completion of this activity, the required interaction stops. The loader will probably be assigned to a different activity and thus relocated to function better (Zouein & Tommelien, 1999). This ever-changing nature of construction sites has led to the emergence of what is known as dynamic layout planning. This approach creates several layouts spanning the project duration so that each unique layout will strive to achieve the site requirements set forth during the layout's life span.

1.3 RESEARCH OBJECTIVES

The efforts put into this research aim to develop an automated system for dynamic layout planning of construction sites at the 2D level. To achieve this, the following sub-objectives are considered:

1- Investigate the potential for using genetic algorithms in solving the site layout planning problem. The use of genetic algorithms as complex function optimizers has been for long acknowledged. Recently some research in the site layout planning domain has been focused on using these evolutionary techniques to solve the site layout planning problem.

2- Formulate a CAD-based GA approach to perform the task of site layout planning. Due to the geometric nature of the problem at hand, the research incorporates a CAD input/output media to facilitate the use of the system. The approach used aims to integrate the powerful graphical capabilities of CAD systems with the intricate search and optimization abilities of genetic algorithms for the purpose of solving the site layout problem.

3- Expand this approach to the dynamic site layout problem. The dynamic aspects of site layout planning have not been thoroughly covered in the literature. The CAD-based GA approach is extended to include the changes that take place in the construction site throughout the project lifespan.

4- Develop an automated system for dynamic site layout planning. The research aims to develop an integrated software package that implements the developed dynamic CAD-based GA model. The CAD platform chosen is AutoCAD™ due to its widespread use in the construction industry in Egypt.

5- Test and validate the system. A carefully selected case study is chosen to test and validate the systems' performance.

1.4 METHODOLOGY

The proposed system primarily consists of an optimization engine and a geometric input / output interface. The link between the optimization engine and the geometrical data contained in AutoCAD™ drawings has been made possible through AutoCAD™ VisualBasic™ Applications. The programmable features of AutoCAD™ enable all site related geometrical data to be detected as an orthogonal 2-D grid. The

optimization engine utilizes this grid in its execution.

To perform the optimization task, the proposed system utilizes genetic algorithms as the main optimization engine. Due to the non-linearity of the objective function to be optimized, the various constraints to be considered and dynamic nature of the optimization problem, traditional operations research techniques were unsuitable. Due to the special nature of the problem at hand, commercial GA software were not used. Instead, a problem dependant genetic algorithm is developed to perform the dynamic site layout planning task.

1.5 THESIS ORGANIZATION

This thesis is organized into 7 chapters. Chapter 2 provides a thorough review of previous studies related to construction site layout planning and the applications of genetic algorithms in construction. The research approach adopted and theoretical background for this study is presented in chapter 3.

Chapter 4 presents functional details of the various components of the site layout planning system as well as details concerning their integration to perform the required task. Details of the physical structure of the program are provided in chapter 5 using an illustrated example. In chapter 6, validation of the system is performed using a 20,000 m^2 construction site as case study. Finally in chapter 7, conclusions and recommendations for future enhancements are presented.

2 LITERATURE REVIEW

2.1 DEFINITION

Many researchers have attempted to define the site layout planning process. One of the more crisp and generic definitions was given by Tommelien et al (1992a).

> *"The task of site layout consists of identifying the facilities needed to support construction operations, determining their size and shape and positioning them within the boundaries of the available on-site areas. Examples of these facilities include offices and tool trailers, parking lots, warehouses, batch plants, maintenance areas, fabrication yards or buildings, staging areas, and lay-down areas."*

2.2 APPLICATIONS IN OTHER DOMAINS

The construction site layout problem may be considered as a sub-domain of the greater "Facility Layout Problem". This problem is common in other specialization areas besides its use in construction engineering. Facility layout has been utilized extensively in the domains of industrial and electrical engineering.

2.2.1 Industrial Engineering

In Industrial Engineering, the facility layout problem is usually termed the plant layout problem. It is concerned with the arrangement of physical facilities (departments, machines) within a predetermined plant area. This arrangement is performed so as to minimize the total material handling costs between departments within the plant (Rosenblant, 1986).

2.2.2 Electrical Engineering

In electrical Engineering the facility layout problem is utilized in the physical design of VLSI (very large scale integrated) microchips in a task named "macro-cell layout generation". In this task the circuit is partitioned and the components are grouped into functional units, the macro cells. These cells can be described as rectangular blocks with terminals (pins) along their borders. These terminals have to be connected by signal nets, along which power or signals (e.g. clock ticks) are transmitted between the

various units of the chip. A net can connect two or more terminals, and some nets must be routed to pads at the outer border of the layout, since they are involved in the I/O of the chip. The goal to attain in the layout process is the minimization of the total chip area which is greatly influenced by the area between the cells occupied by the signal net wiring (Schnecke & Vornberger, 1997).

2.3 PROBLEM APPROACHES

In the field of construction engineering, researchers have chosen various approaches to deal with the layout planning problem. These approaches differ from one another in the level of detail they provide and the extent to which they yield a well round solution to the rather sophisticated problem of layout planning. The three approaches are briefly described in the following section.

2.3.1 Static Layout Planning

Most construction resources require space on site. This is the case for materials and equipment, support facilities (e.g., trailers or parking lots), and demarcated areas (e.g., law down areas, roads, or work space), but also for obstacles (trees or existing buildings). Allocating site space to resources so that they can be accessible and functional during construction is a problem known as layout planning Zouein & Tommelien (1999). A layout is termed "static" if effort put in planning yields only one site layout that will span the entire project duration.

In almost all static layout planning models, assignment of temporary facilities takes place such that:

1- Temporary facilities are assigned within the site boundaries.
2- Temporary facilities are assigned in positions that attain a certain or several objectives
3- Temporary Facilities are assigned in such a manner that specific geometrical constraints are attained.

Usually this layout will become obsolete after any significant progress in the project, as the needs of construction sites change considerably from time to another throughout different phases of construction.

Early research in the area of site layout planning was limited to static layout problems. Performing the more sophisticated "dynamic layout planning" was hindered

by the high computational capabilities that was unavailable in the late 80's and early 90's. Researchers acknowledged their models' limitations and recommended future research to consider the more generic "dynamic layout problem" (Tommelien et al. 1992b, Cheng & O'Connor 1996, Li & Love 1998, Elbeltagi,1999).

2.3.2 Dynamic Layout Planning

Creating layouts that change over time as construction progresses is termed dynamic layout planning. The needs of construction sites change considerably from time to another throughout different phases of construction.

1- As the project grows, more area is occupied by permanent facilities leaving less space to position supporting facilities.

2- The types and quantities of material being delivered to the site change considerably throughout the project. Thus, the areas needed for their storage change accordingly. (Zouein & Tommelien, 1999)

3- In most projects, the demand for heavy equipment and on-site administrative support facilities changes as construction progresses. This causes significant changes from time to another in both the required site space to support these facilities and the presumptive position of each relative to the others.

4- Access roads that are available during one stage of construction may not necessarily be available during another stage.

The need for a dynamic model to represent site requirements is clearly understood. Applications in construction engineering nonetheless lagged behind their counterparts in other fields such as industrial engineering. For instance, Rosenblant (1986) presented a mathematical model for dynamic plant layout where an optimization technique named "dynamic programming" was utilized to mimic the dynamism of the layout process. Tommelien & Zouein (1993) presented one of the earliest dynamic models for construction site layout planning. A number of studies have followed since then (Zouein & Tommelien 1999, Elbeltagi et al 2001).

Zouein & Tommelien (1999) presented a very comprehensive model for dynamic layout planning for construction sites. They laid the basis for researches to come. Some of the fundamental concepts introduced that constitute the basis of dynamic modeling of construction sites are:

i) *Primary Time Frames (PTF's):* They are the smallest time intervals demarcated by the arrival or departure of site facilities. (i.e. during a PTF

no facility arrives or leaves the site). By definition, PTF's include only those facilities that coexist on site in that time frame.

ii) *Dynamic layout objective function*: In addition to the tradition minimum travel distance / cost objective, the relative cost of relocating a facility is included in the objective function.

iii) Constraints: Were grouped into *Hard* and *Soft* constraints. Hard constraints include non-overlap, minimum / maximum, orientation, parallel / perpendicular and in-zone constraints.

The model they formulated utilized simple linear programming in solving the layout problem.

Researchers in the area of dynamic layout of construction sites have acknowledged the incompetence of dynamic modeling in solving all problems. Dynamic layout can only assign a set of facilities that occupy the site during a certain time period on a predetermined site area. If during any time frame there is no feasible solution, the planner must either alter his schedule or reconsider the area assigned to facilities. The more generic approach of "Space Scheduling" addresses problems of these kind.

2.3.3 Space Scheduling

The first to acknowledge the broader approach of space scheduling in construction were Tommelien & Zouein (1993). They defined space scheduling as "The bi-directional interaction between scheduling and layout construction or improvement." The research they conducted in 1993 admitted to the need of formulating a space scheduling model if comprehensive modeling was to be performed. Their research stated that "schedule changes are crucial for coming up with feasible layouts when insufficient space is available to accommodate all resources on site for any time interval."

It was not until 2001, when Zouein & Tommelien formulated an improvement algorithm for limited space scheduling. This work was a continuation of the dynamic layout planning solution method they presented in 1999. Their space scheduling model came into action when it was impossible to construct a feasible layout for any time interval. The model proposed two strategies to overcome this situation:

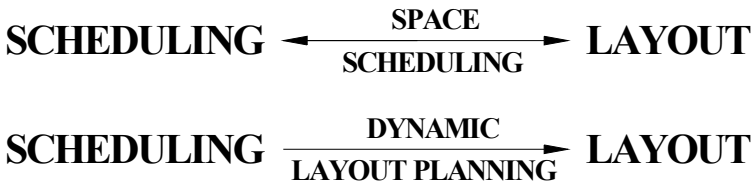


Figure 2-1 Difference between space scheduling and dynamic layout planning (Tommelien & Zouein, 1993)

Strategy A: Delays an activity in the problematic time interval. Activities are selected in order of decreasing total float.

Strategy B: Lowers resource level of activity and therefore lengthens its duration. Activities are selected in order of decreasing total float.

To conclude, space scheduling can be considered to be a form of constrained resource leveling, with the resource being site space.

2.4 METHOD OF FACILITY ASSIGNMENT

Another main difference between site layout planning models, is in the manner the facilities are assigned on site. Two distinct assignment methods can be characterized, namely *facility to location* assignment and *facility to site* assignment.

Facility to location, assigns a set of predefined facilities to a set of predefined locations such that #of locations ≥ # of facilities. On the other hand, facility to site assignment, assigns a set of predefined facilities to the entire space available on site. Facility to location assignment neglects one very important aspect, that of size. All locations are assumed to be able to fit all facilities. This assumption is weakened by the discrepancies found between the sizes of most construction site facilities. Facility to site assignment is considered more generic as it assumes that the planner has not yet settled on the feasible locations for facility placement. Also, during this type of assignment, many spatial constraints must be satisfied which poses an extra computational burden on the model. In brief, facility to site assignment is considered the more generic case of facility assignment.

2.5 PROBLEM SOLVING TECHNIQUES

Researchers have utilized many problem solving techniques in layout planning ranging from purely mathematical models to knowledge based systems. Artificial

intelligence and evolutionary algorithms have also been used in problem solving. Up till now, researchers have not acknowledged a specific problem solving technique to be more suitable than another. Techniques used can be broadly divided into two categories, namely heuristical techniques and mathematical techniques.

2.5.1 Heuristics & Expert Systems

Early research in the area of layout planning was focused more heavily on systems that provide guidelines or heuristics for assisting managers in layout planning. Systems of this type are more dependant on manual rather that automated design (Tommelien et al. 1992a).

One of the early innovative expert systems for construction site layout planning was the SightPlan model presented by Tommelien et al. (1992b). SightPlan was a model that mimics how people lay out construction sites and encodes the domain knowledge they apply in this process. The model was implemented using common lisp.

SightPlan's knowledge was modeled after two power plant project case studies. Input required to the system include:

1- Major permanent facilities on site with their dimensions.
2- Access roads with their dimensions and location.
3- Dimensions of temporary facilities.
4- Constraints on the location of temporary facilities relative to permanent facilities.
5- Zones that partition the site into smaller areas.

One of the interesting findings during knowledge acquisition was that the A/E and the construction manager each laid out part of the facilities on the case study project, with the A/E layout preceding the CM layout.

It is the investigator's opinion that the system is highly case specific due to the model's narrow scope of case studies used in excerpting knowledge. The model is mostly applicable for industrial projects having under-constrained layouts.

Cheng & O'Connor (1996) developed an automated site layout system for temporary facilities. Their system *ArcSite* integrated a database management system (DBMS) and geographic information system (GIS). The main objective of the system is to automate the planning tasks required for facility layout. This is performed through the identification of areas suitable for assigning facilities in order to minimize construction conflicts and improve project efficiency. The following four sub-

objectives were achieved by developing the system:

1- Obtain the knowledge and procedures that project managers use in laying out site facilities.
2- Model the experts' knowledge and experience of site planning and express it in a systematic form.
3- Define the dominant variables and develop an evaluation method to identify the suitable location for the facility.
4- Develop a GIS based site layout system to replace manual methods.

The ArcSite system is different in the fact that it performs the site layout planning process taking into account the possibility of incorporating temporary facilities (TF) inside constructed permanent facilities (PF). The procedures developed in the research to a acquire and represent site layout knowledge is classified into three phases:

1- Compiling the experts' knowledge and experience for site layout.
2- Interpreting the knowledge into the knowledge base.
3- Translating the knowledge base into the ArcSite implementation forms.

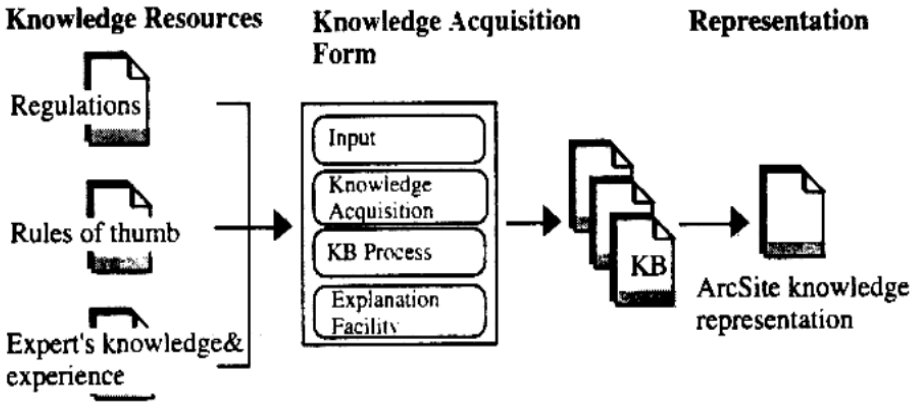


Figure 2-2 Procedure of knowledge acquisition and representation
Cheng & O'Connor (1996)

Knowledge representation is complied into five categories of constraints:

1- Spatial: Calculates the area required for the facility
2- Distance: Defines the facility's priority and proximity to the work sites.
3- Adjacency: Identifies if the facility is located next to the construction area, another facility, access roads or gates.
4- Position: Position of the facility *relative to* any other facilities.

5- Accessibility: Defines the accessibility of the area where the facility is located.

The system is implemented using 4 main software components:

1- CAD (MicroStation): To create site geometry
2- ARC tools: Conduct the facility layout analysis and design
3- Excel: Conducts the DBMS function such as adding tabular attributes, data consolidation, database query and user interface.
4- ARC prompt: ArcSite allows the usr to suspend the user interface and exit to the ARC prompt to use Arc/Info functions.

2.5.2 Mathematical Techniques

Most mathematical techniques involve the identification of one or more goals that the layout should strive to achieve. Any of these goals is interpreted to what mathematicians term "an objective function". This objective function is then optimized under problem specific constraints via any common optimization technique. Some of the techniques used by researchers in the site layout optimization problem are listed in Table 2-1.

Table 2-1 Summarization of optimization techniques used in solving site layout planning problem

Technique	Research
Linear Programming	Zouein & Tommelien (1999)
Genetic Algorithms	Li & Love (1998), Hegazy & Elbeltagi (1999)
Neural Networks	Yeh (1995)

2.6 OPTIMIZATION

Researchers that have used optimization techniques in site layout planning have formulated many equations to be used as their optimization goal or *objective function*. Table 2-2 summarizes the different objective functions formulated by researchers using optimization techniques.

In almost all optimization approaches, in site layout planning, the layout goal to be attained, takes the general form:

Min:[Inter-facility Transportation Costs]

This goal is translated to the objective function that takes the general form:

$$Min: \left(\sum_{i=1}^{P-1} \sum_{j=i+1}^{P} W_{i,j} * d_{i,j} \right) \quad \text{.....(2-1)}$$

Where:

P= Total number of fixed and temporary facilities present.

$d_{i,j}$ = Distance between facilities i and j

In the literature, the term $W_{i,j}$ represents one of the following:

1- Transportation cost data of some sort or

2- A relative proximity weight that reflects the required closeness between any two facilities.

Practically, in the absence of data, the project manager would try to assign a relative weight which represents his/her anticipated closeness required between any two facilities. But the question is what exactly should this weight depend on? Is it only a function of transportation costs or do other variables dictate its exact value? Even if these variables are known, how do they interact and what is the extent of the contribution of each to the value of the proximity weight?

Table 2-2 Objective functions formulated by researchers using optimization techniques

Objective function to minimize	**Research**
Frequency of trips made by construction personnel.	Li & Love (1998)
Total transportation costs of resources between facilities.	Tam et al. (2001) Cheung et al. (2002)
Cost of facility construction + Interactive cost between facilities	Yeh (1995)
Proximity weight on an exponential scale	Hegazy & Elbeltagi (1999)
Proximity weight + Relocation weight	Zouein & Tommelien (1999)

Considering a single objective in layout planning overlooks the intricate nature of construction sites. When planning a construction site any project manager wants to achieve various objectives while simultaneously abiding with various constraints.

Researchers who introduced the idea of the proximity weight were highly aware of this problem. They were also aware that other models that explicitly tried to minimize costs of any sort required enormous amounts of cost related data. Data of this sort may be unavailable or very hard to attain for the following two reasons:

1. Site planning is performed in a very early planning phase. The exact scope of the project may not be well defined.
2. Although organized construction corporations keep track of most relevant cost data, the cost data required for site optimization may not be readily available. For example, data on the cost of transportation of 1 unit of concrete per unit distance is not the type of cost data contractors usually keep a record of.

In an attempt to fully describe the proximity weight, (Elbeltagi & Hegazy 2001) proposed a fuzzy rule based system that quantifies this proximity weight based on three variables. There work was an extension to their previous EvoSite model (Hegazy & Elbeltagi 1999). The six level proximity weights (Table 2-3) were defined as fuzzy sets as shown in Figure 2-3

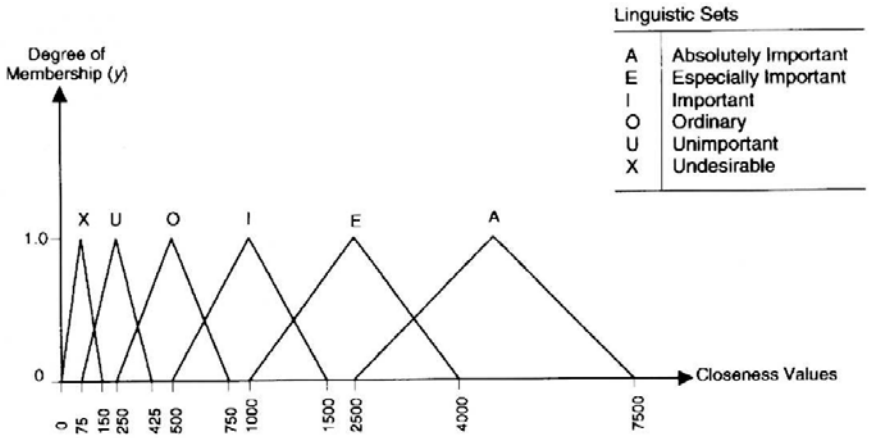


Figure 2-3 Fuzzy sets of the variable "proximity weights" (Elbeltagi & Hegazy, 2001)

Based on construction experts opinion, three input variables we formed to govern the proximity weight value. These input variables are:

1. Work flow (trips/day)
2. Safety / Environmental concerns (scale of 1 to 10)
3. User preference (scale of 1 to 10)

Figure 2-4 illustrates the fuzzy sets depicting each of these input variables.

Each of the three input variables was defined by three fuzzy sets, thus $3^3 = 27$ rules were formulated linking each of these input variables to the proximity weight. For

example, rule #3:

"If workflow is *low*, safety concerns are *medium* and user preference is *low*, then the required proximity is *unimportant*."

The process involved in the system is known as *fuzzy rule-based inferencing*.

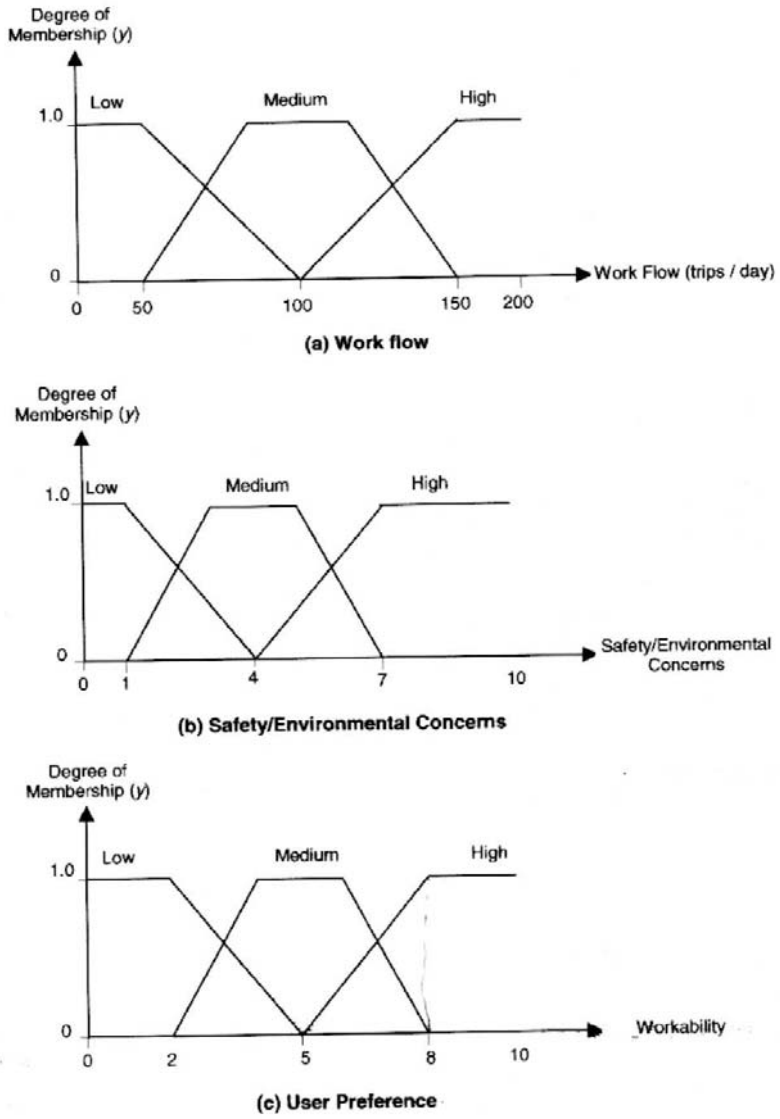


Figure 2-4 Fuzzy sets for the input variables (Elbeltagi & Hegazy, 2001)

2.6.1 Linear Programming

Linear programming is a class of mathematical programming models concerned with the efficient allocation of limited resources to known activities with the objective of meeting a desired goal (such as maximizing profit or minimizing cost). The distinct

characteristics of linear programming models is that the function representing the objective and the constraints are linear (Taha, 1971).

Despite its relative simplicity and prevalence in a vast range of engineering optimization applications from its initial conception in the 1950's, not many models have directly utilized linear programming in site layout planning. Zouein & Tommelien (1999) utilized a simple linear program model in their dynamic layout model. The typical objective function to be minimized each time a facility is to placed on site is simply:

$\Delta\ VFL$ = Inter-facility Transportation Cost + Facility Relocation Costs .....(2-2)

Which translates to the following mathematical equation:

$$\Delta VFL = T_i \times W_i \left(\left| X_{i-1} - X_i \right| + \left| Y_{i-1} - Y_i \right| \right) + R_i \left(\left| X_{i_{t-1}} - X_i \right| + \left| Y_{i_{t-1}} - Y_i \right| \right) \ldots (2\text{-}3)$$

Where:

ΔVFL: The change in the objective function due to the introduction facility *i*

T_i: The time resource *i* is present on site (Time of current time frame)

W_i: Proximity weight between resource i and resource i-1.

X_{i-1}: X coordinate of resource i-1.

Y_{i-1}: Y coordinate of resource i-1.

R_i: Relocation weight of resource i.

X_{ti-1}: X coordinate of resource i at previous time frame.

Y_{ti-1}: Y coordinate of resource i at previous time frame.

X_i: X coordinate or resource i at current time frame.

Y_i: y coordinate or resource i at current time frame.

The optimization problem is characterized by having various constraints each separated by an OR operator:

$$X_1 \le X \le X_2 \qquad X_3 \le X \le X_4 \qquad X_5 \le X \le X_6$$
$$Y_1 \le Y \le Y_2 \qquad Y_3 \le Y \le Y_4 \qquad Y_5 \le Y \le Y_6$$

The solution algorithm includes two main imbedded sub-modules; the CSPA (constraint satisfaction and propagation algorithm) and the PTFLCA (primary time frame layout construction algorithm).The CSPA module determines the sets of feasible positions for facilities in a given time frame. Upon assignment each facility is assigned a SPP (set of possible positions). The SPP of facilities not yet assigned is influenced by

the assignment of new facilities. On the other hand, the PTFLCA module performs the assignment process with time frames in chronological order. It starts from the SPP's output by the CSPA for a specific time frame, then singles out a position for each facility one at a time.

Until now the optimization seems quite simple. In fact this bare simplicity obscures some implementation difficulties, mainly in the formulation of the constraints. It is the investigators' opinion that the process of defining the SPP of a facility to prevent overlap or constraint violation via a purely mathematical approach ignores much of the capabilities of available graphically oriented softwares that can easily detect overlap between geometrical entities or position them based on set constraints.

2.6.2 Genetic Algorithms

Genetic or evolutionary algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance (Goldberg, 1989).

Li & Love (1998) presented a genetic algorithm for facility allocation. The algorithm presented was specific and limited because it optimally placed facilities in predefined positions. The user simply specified locations where facilities could be placed and the algorithm would assign facilities in their best locations so as to minimize the total travel distance between facilities. The algorithm only addressed the static layout problem. The solution generated was completely independent of the site layout geometry and individual facility size or shape.

The objective of site-level facility layout is to minimize the total travelling distance of site personnel between facilities. The total distance TD is defined as:

$$TD = \sum_{i=1}^{n} \sum_{x=1}^{n} \sum_{j=1}^{n} \delta_{xi} f_{xi} d_{ij} \quad \text{.....(2-4)}$$

Where:

N: Number of facilities

δ_{ij}: permutation matrix variable

f_{ij}: frequency of trips by personnel between facilities i and j ($f_{ij} = f_{ji}$)

d_{ij}: Distance between locations m and n

Selecting an appropriate representation for the solution is termed "coding" in the GA solution. It is considered one of the most important steps in formulating an accurate solution. The problem at hand is a *combinatorial* optimization problem. Many approaches, like the permutation, binary and ordinal representations have been used for these types of problems. In this study, the permutation type was used, the other approaches being unsuitable. The string layout representation for 8 facilities is as follows:

Facility	1	2	3	4	5	6	7	8
Location	5	3	1	7	8	2	4	6

Crossover is one of the main operators utilized in any GA to propagate a new population from an old one. Of the most efficient operators developed is the edge recombination operator (Li & Love, 1998). The edge recombination operator uses an edge table to construct an offspring that inherit as much genetic information as possible from the parent strings.

As we can see, the model developed by Li and Love is a purely mathematical model that utilizes facility to location assignment and has its limitations; thus; its use in the industry as a tool for site management is less likely.

Hegazy & Elbeltagi (1999) presented their EvoSite model for site layout planning. Their work was much more comprehensive and generic than that performed by Li and Love (1998). The model was novel in utilizing a simple but effective spreadsheet representation of site geometry. In the proposed model, a facility is represented as a group of unit areas that can take any user specified shape. The model accepts any user specified site shape and incorporates a flexible GA procedure for the optimum placement of facilities.

$$Min: \left(\sum_{i=1}^{P-1} \sum_{j=i+1}^{P} R_{i,j} * d_{i,j} \right) \quad \text{.....(2-6)}$$

Where:

P= Total number of fixed and temporary facilities present.

$d_{i,j}$ = Distance between facilities i and j

$R_{i,j}$ = A relative proximity weight that reflects the required closeness between

facilities i and j.

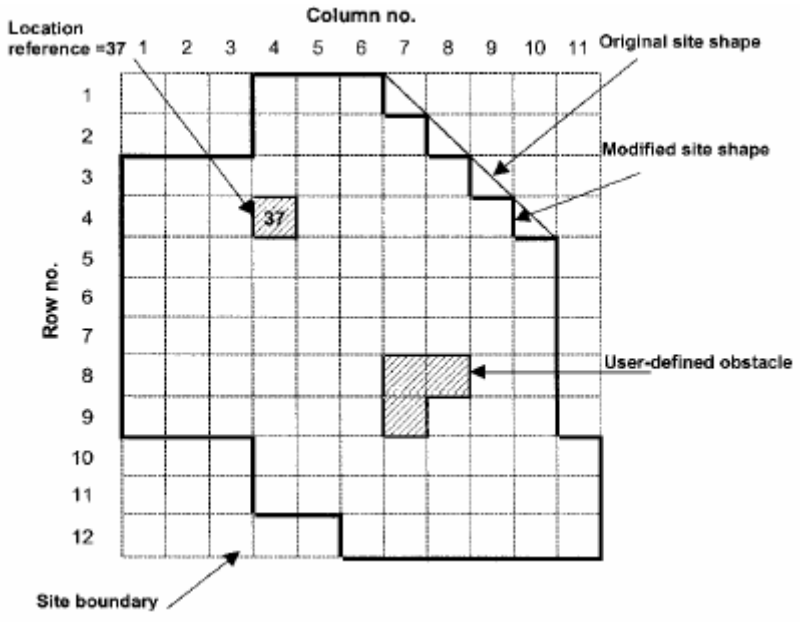


Figure 2-5 Site and Facility representation in the EvoSite model (Hegazy & Elbeltagi, 1999).

The EvoSite model is a static layout planning model. The facilities have relative proximity weights between each other, this represents the level of interaction between facilities or the preference in having the facilities close or apart from each other. The proximity weights used are *qualitative* proximity weights. Theses qualitative measures are then mapped to a *quantitative* weight that can be used in optimization as shown in Table 2-3. The six levels of desired closeness have been used by other researchers in the facility layout problem. It is to be noted that there is no theoretical background for the exact numerical values given to each closeness level.

Table 2-3 Closeness relationship values (Hegazy & Elbeltagi, 1999)

Desired relationship between facilities	Proximity weight
Absolutely necessary	$6^5 = 7{,}776$
Especially important	$6^4 = 1{,}296$
Important	$6^3 = 216$
Ordinary closeness	$6^2 = 36$
Unimportant	$6^1 = 6$
Undesirable	$6^0 = 1$

The genetic algorithm used generated random solutions (strings) as any GA would do. The solution was then evaluated for feasibility (for non-overlap between facilities)and the non-feasible solutions were disregarded. The system had a relatively high computation time. When the model was run with a population of 200 genes and 100 offspring, it took a 233 MHz processor nearly 150 minutes to solve.

Zouein et al (2002), formulated a genetic algorithm for solving the site layout problem with unequal-size and constrained facilities. The objective function utilized was similar to that described in Eq (2-5). Their genetic algorithm was highly problem oriented, incorporating modified mutataion and crossover operators that suit the gentic coding representing the problem at hand. Their research focused more on the investigation of the modified GA than on developing an integrated site layout planning system. The strengths and limitataions of their proposed GA was tested in the case of:

1. Loosely vs. tightly constrained layouts with equal levels of interactions between facilities.
2. Loosely vs. tightly packed layouts with variable levels of interactions between facilities.
3. Loosely vs. tightly constrained layouts.

2.6.3 Neural Networks

Yeh (1995) presented a novel research on the use of Annealed Neural Networks for construction site layout. Yeh formulated the problem as a discrete combinatorial optimization problem. This formulation is identical to that presented by Li & Love (1998) in Eq (2-4). Similarly, the model aimed at assigning a set of predetermined facilities on a set of predetermined locations while satisfying a set of layout constraints. Thus the system's method of assignment is classified as a facility to location assignment.

The annealed neural network proposed was a fusion between Hopfield neural networks and simulated annealing. Hopfield neural networks have been used to solve a wide variety of discrete combinatorial optimization problems. Its main drawback has been its limited inability to escape local minima. Simulated annealing was also proposed as a general technique to solve combinatorial optimization problems. Simulated annealing is considered as a probalistic hill-climbing search algorithm which

finds a global minimum by combining gradient decent with a random process. Although simulated annealing has provided quality solutions in many practical problems such as the traveling salesman problem, it requires unacceptably high computational times.

The annealed neural network proposed exhibits the rapid convergence of the neural network while preserving the solution quality afforded by simulating annealing. (Yeh, 1995).

2.7 SUMMARY OF RESEARCH

Studies in the field of site layout planning have commenced as early as the early 70's. Research has evolved from the development of heuristic models and expert systems to the formulation of analytical models having a precise optimization goal. Various optimization tools have been used in site layout planning. Traditional optimization tools like linear programming have given way to artificial intelligence techniques like neural networks and genetic algorithms.

Figure 2-6 summarizes the latest research pertaining to site layout planning based on the classification criteria discussed in this chapter. It is to be noted that, the model presented in this research is a mathematical model that uses facility to site assignment.

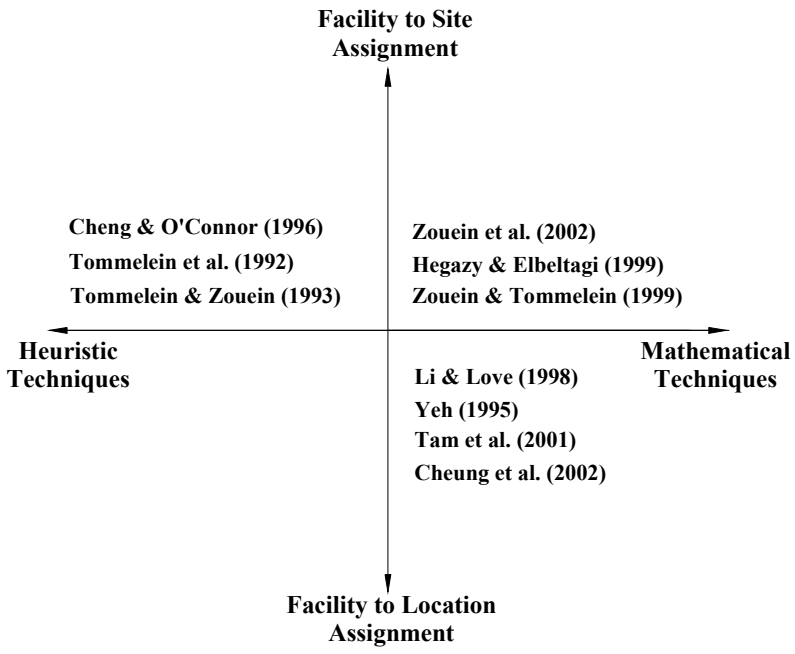


Figure 2-6 Classification of some of the construction site layout models

3 RESEARCH APPROACH & THEORETICAL BACKGROUND

In this chapter, three main topics are discussed. The first section discusses the suitability of using CAD platforms and genetic algorithms in an integrated system for site layout planning. The second section presents the theoretical background behind genetic algorithms. The final section compares between two representations for the objective function used in site layout planning.

3.1 SUITABILITY OF CAD PLATFORMS AND GENETIC ALGORITHMS

Computer Aided Design platforms experienced great advances during the late 80's. Their use in various engineering disciplines became inevitable. In the civil engineering branch, CAD software started off in use in the design stage as a drafting tool. Applications of CAD platforms in the construction stage lagged behind their counterparts in the design stage. In the field of construction engineering, Mahoney & Tatum (1994) reported the potential benefits of using CAD on construction site operations. Applications included planning survey control, planning construction sequence and method, analyzing concrete sequence and placements, designing formwork for concrete and coordinating subcontractors. The researchers suggested that CAD could be used to plan construction site layouts, as adoption of such systems allows easy and accurate visualization of the relationship between the permanent structures and temporary facilities on site.

The site layout planning problem is evidently graphical in nature. Site boundaries, existing buildings on site, obstacles, and temporary site facilities all occupy space and have distinct shape. Thus, the need to represent the relationship between all aforementioned entities in some sort of graphical format is apparent. Tommelien & Zouein (1993) were one of the earliest researchers that utilized CAD platforms. Their *MovePlan* model was implemented in object oriented common lisp (MCL 2.0). Recent research focused more on the use of artificial intelligence tools than on benefiting from the graphical capabilities of CAD platforms. This research tries to integrate the powerful graphical capabilities of CAD platforms with the evolutionary optimization technique known as genetic algorithms.

In this research, genetic algorithms are used as function optimizers, although the range of problems to which genetic algorithms have been applied is quite broad. (Chan

et al, 1996). The use of GA's as an optimization tool is dependant on the problem to be solved. In many optimization methods we move gingerly from a single point in the decision space to the next using some transition rule to determine the next point. This point to point method is dangerous because it is a perfect prescription for locating false peaks in multimodal search spaces. By contrast GA's work from a rich database of points simultaneously, climbing many peaks in parallel. GA's belong to the class of methods known as "weak methods" in the Artificial Intelligence community because it makes relatively few assumptions about the problem that is being solved. GA's do not utilize gradient information. Thus, they are highly applicable to problems having non-differentiable functions, as well as functions with multiple local optima. On the other hand, if there exists a specialized optimization method for a specific problem, then genetic algorithm may not be the best optimization tool for that application (Whitley, 1993).

Al-Tabtabi & Alex (1998) suggest that the use of GA in optimization is appropriate in the following circumstances:

1- Conventional statistical & mathematical methods are inadequate.

2- The problem is very complex, because the possible solution space is too large to analyze in finite time.

3- The additional information available to guide the search is absent or not sufficient, so conventional methods are not practical.

4- The solution to the problem can be encoded in the form of strings and characters.

5- The problem is large and poorly understood.

6- There is an urgent need for near-optimal solutions to use as starting points for conventional optimization methods.

Three of the aforementioned points make the utilization of GA in solving the site layout problem very suitable. Firstly, when modeling a large construction site the available solution space is immense. The larger the available areas for placement and the greater number of facilities needed to be assigned the larger the feasible solution space becomes. Secondly, the solution to the problem can be encoded in the form of strings. This will be highlighted in chapter 4. Thirdly, finding a comprehensive solution to the site layout problem is not as trivial as minimizing an objective function. Conditions on construction sites involve far more constraints, variables and uncertainties than those taken into consideration in most mathematical approaches for problem solving. Practically, the difference between optimum and near optimum

solution is not that important, as even the optimum solution may require slight enhancements dictated by unforeseen site conditions.

3.2 GENETIC ALGORITHMS

Genetic algorithms were first investigated by John Holland (1975) at the University of Michigan. Further studies were carried out by his students DeJong (1975) and Goldberg (1989).

In the very broad sense genetic algorithms or GA's, are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with seemingly randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance (Goldberg, 1989).

The parallelism between GA's and biological evolution and natural selection is evident. Goldberg (1989) summarized this parallel nature in Table 3-1.

Table 3-1 Comparison between natural and GA terminologies

Natural Terminology	**GA Terminology**
Chromosome	String
Gene	Feature, character, detector
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set
Epistatsis	Non-linearity

3.2.1 GA example

The best way to introduce the concepts of genetic algorithms is through an example of a GA in action. We will illustrate in this example the two principles of

coding and selection.

Suppose we are interested in finding the maximum value of the function $y = -0.005x^2 + 1.5x + 10$ over the range $0 \le x \le 255$ as illustrated in Figure 3-1. The first aspect to consider about a GA is its encoding, that is the transformation of the solution to a unique chromosome like structure. For simplicity we will use an 8 string binary coding to represent the solution space. Take for example the string 11001100, the binary representation for the decimal number 230.

The binary number is translated into its decimal equivalent as follows:

128	64	32	16	8	4	2	1
1	1	0	0	1	1	0	0

Coding:
= 128+64+8+4 = 230

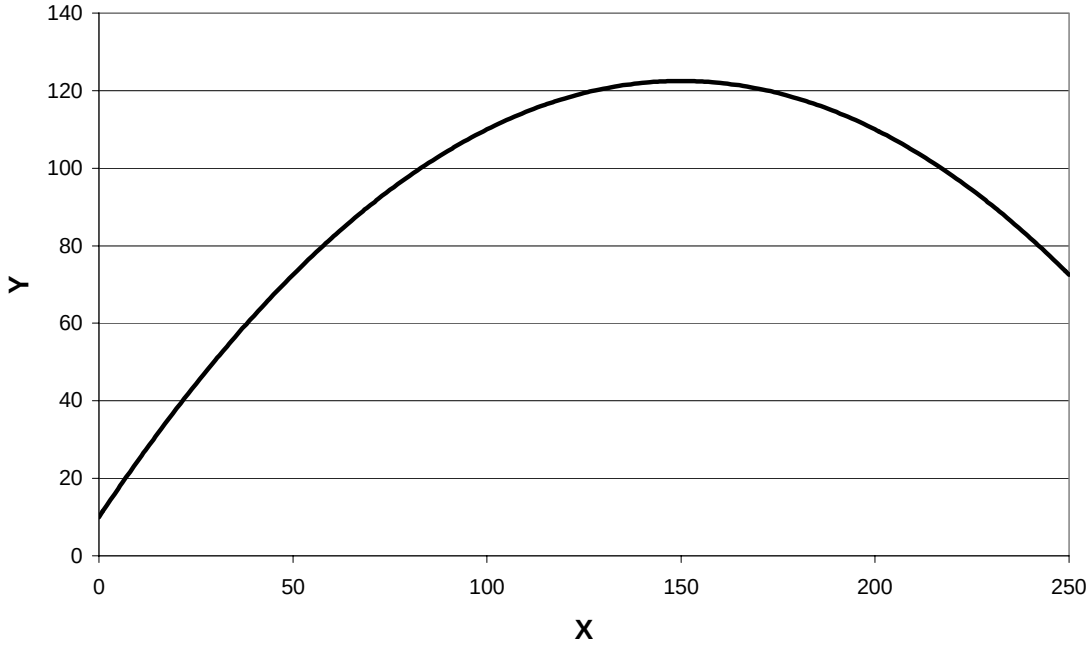


Figure 3-1 Graph of $y = -0.005x^2 + 1.5x + 10$

Any GA follows the shown simple algorithm (Chan et al. ,1996)

```
Begin
Generate a new population of solutions
While terminating condition is not met DO
   Evaluate the solutions
   Select the better solutions
   Recombine solutions using genetic operators
END
```

Initially a GA must start with a randomized initial population of solutions.

Suppose we will randomly choose 10 initial solutions. For each solution or *string* the GA evaluates its objective function. In column (5) we evaluate the probability of selection in the next population such that a solution's probability to be selected is directly proportional to the value of its objective function. P_{select} is sometimes referred to as the solution's *relative fitness*. In column (6) the expected count for the current solution i is calculated by multiplying p_{select} by the number of solutions N, while in column (7) the actual count is calculated by rounding off column (6). The selection is made such that fitter solutions have a higher probability of being reselected. On the other hand, bad solutions are eliminated from the population. It is in this manner that GA's mimic the process of natural selection and survival of the fittest.

Table 3-2 A genetic algorithm by hand

Solution # (1)	**String** (2)	**X** (3)	**y = f(x)** (4)	$\mathbf{p_{select} = f_i / \sum f}$ (5)	$\mathbf{Count_{exp} = p_{select} * N}$ (6)	$\mathbf{Count_{act}}$= (7)
1	00111001	57	79	0.11	1.1	1
2	00011110	31	52	0.073	0.73	1
3	11101111	223	95	0.132	1.3	1
4	00110001	49	71	0.099	0.99	1
5	00001001	9	23	0.032	0.32	0
6	00000110	6	19	0.02	0.2	0
7	11010101	213	103	0.144	1.44	1
8	10010110	150	122	0.17	1.7	2
9	00011010	26	45	0.06	0.6	1
10	01011111	95	107	0.15	1.5	2
Sum			717	1	10	10

As it can be seen, the objective function is evaluated several times during any one generation. Thus the evaluation process must be relatively fast. As the members of the population reproduce, their offspring must be evaluated. If it takes 1 hour to perform an evaluation, then it takes over 1 year to perform 10,000 evaluations. This is approximately 50 generations for a population of only 200 strings.

After the selection process the solutions are recombined using the common

genetic operators, namely *crossover* and *mutation*. They will be discussed in further detail later in this chapter.

3.2.2 Concept of hyperplane sampling

The question most people tend to ask at this point is, Why does any of this work? How would such a random search technique produce anything useful? John Holland, the founder of genetic algorithms developed several arguments to explain how GA's can result in complex and robust search by sampling hyperplane partitions of a search space.

Perhaps the best way to understand how a GA can sample hyperplane partitions is to consider a simple 3-dimensional space as illustrated in Figure 3-2. Assume we have a problem encoding with just 3 bits and each bit can take on either a 0 or 1 value. GA's that utilize binary encoding are known as *canonical genetic algorithms*. This can be represented as a simple cube with the string 000 at the origin. The 8 corners of the cube represent the 2^3 possible encodings or solutions. It can be noticed that the front plane of the cube contains all the points that begin with a 0. If an "*" symbol is used as a 'don't care' or wild card match symbol, then this plane can also be represented by the string 0**. Strings that contain * are referred to as *schemata*. Each schemata corresponds to a hyperplane in the search space. The *order* of the hyperplane refers to the number of actual bit values that appear in its schemata. Thus, 1** is order-1 while *011****1 is order-4.

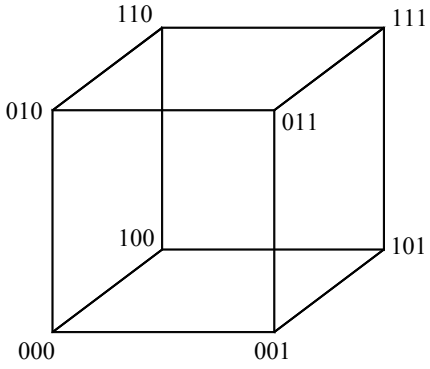


Figure 3-2 Cube representation of a 3 dimensional hyperplane

Let us inspect our cube example more closely. How many schema does a single solution belong to? Generally speaking for any string encoding of length L and coding alphabet n. Number of schema = n^L -1 . In our example any solution belongs to 2^3-1=8

schemata. Take the point 001, it belongs 3 order-1 schema (**1, 0**, *0*), 3 order-2 schema (0*1, *01, 00*) and 1 order-3 schemata 001 which is the point itself. It is to be noted that the order-1 schema represent three faces of the cube while the order-2 schema represent the three edges of the cube.

During the evaluation of any population of solutions in a GA far more hyperplanes are sampled than the number of strings contained in the population.

This mention of hyperplanes and schemata leads us to the concept of *implicit parallelism*. Implicit parallelism implies that many hyperplane competitions are simultaneously solved in parallel during the evaluation of a population. The theory suggests that through the process of reproduction and recombination the schemata of competing hyperplanes increase or decrease their representation in the population according to the relative fitness of the strings that lie in those hyperplane partitions. Thus even though a GA never explicitly evaluates any particular hyperplane partition, it should change the distribution of string copies as if it had. This idea is formalized through the following equation.

$$M(H,t') = M(H,t)\frac{f(H,t)}{\overline{f}} \quad \text{..... (3-1)}$$

Where:

M(*H,t*) = The number of strings sampling a hyperplane *H* at the current generation *t*

M(*H,t'*) = The number of strings sampling a hyperplane *H* at the current generation *t* after selection but before crossover.

f(H,t) = The average evaluation of the sample of strings in partition H in the current population

3.2.3 The Schema Theorem

A foundation has now been laid to develop the fundamental theorem of genetic algorithms. The schema theorem (Holland, 1975) provides a lower bound on the change in the sampling rate for a single hyperplane from generation *t* to generation *t+1*. From Eq.(3-1) we want to consider the effects of crossover as the next generation is created from the intermediate one. We consider that crossover is performed probabilistically with probability P_c. The portion that does not undergo crossover is unchanged. When crossover does occur we must account for the losses due to its disruptive effects.

$$M(H,t+1) = (1-P_c)M(H,t)\frac{f(H,t)}{\overline{f}} + P_c\left[M(H,t)\frac{f(H,t)}{\overline{f}}(1-losses) + gains \right] \quad \text{..... (3-2)}$$

The term "losses" refers to those disruptions in the schema representing the hyperplane H, produced by crossover . But crossover is not always disruptive to the schema. For example consider the schema 110*****. If crossover between the two strings 11010100 and 00010011 occurs after the second bit, their offspring will be 11010011 and 00010100. The schema 110***** is still preserved in the first offspring. A conservative approach is to ignore all gains, considering any crossover operation to be disruptive.

$$M(H,t+1) \geq (1-P_c)M(H,t)\frac{f(H,t)}{\overline{f}} + P_c\left[M(H,t)\frac{f(H,t)}{\overline{f}}(1-disruption) \right] \quad \text{..... (3-3)}$$

We might wish to consider one exception. If two strings that both sample H are recombined then no disruptions occur. Disruption is therefore given by:

$$\frac{\Delta(H)}{L-1}(1-P(H,t) \quad \text{..... (3-4)}$$

Where:

Δ(H) : The defining length associated with 1-point crossover

L : String length

P(H,t) = The proportional representation of H in the population *(=M(H,t) / N)*

Dividing both side of Eq. (3-2) by the population size "*N*" to transform the expression in terms of *P(H,t)* instead of *M(H,t)*.

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}\left[1 - P_c\frac{\Delta(H)}{L-1}(1-P(H,t)) \right] \quad \text{..... (3-5)}$$

Till now our version of the schema theorem does not account for mutation. Let o(H) be a function that defines the order of the hyperplane H. P_m is the probability of mutation, where the mutation operator always flips the string. The probability that mutation affects the schema H is $(1\text{-}P_m)^{o(H)}$. This leads to the general expression of the schema theorem.

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}\left[1 - P_c\frac{\Delta(H)}{L-1}(1-P(H,t)) \right](1-P_m)^{o(H)} \quad \text{..... (3-6)}$$

3.2.4 GA encoding

Usually there are only two main components of most genetic algorithms that are problem dependant, the problem encoding and the evaluation function. The encoding

scheme of any GA is the essence of its success. Goldberg (1989) presented two main principles for choosing GA encoding:

1) Principal of meaningful building blocks:

   "the user should select a coding so that short, low order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions"

2) Principal of meaningful alphabets:

   "The user should select the smallest alphabet that permits a natural expression of the problem"

Some GA encodings, that have been successfully used in practice are the binary, permutation and value encoding. Each type is discussed in detail below.

1. Binary Encoding: The first works of GA's used this type of encoding. Each chromosome assumes either the value of 1 or 0. Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Chromosome A	1	1	0	1	0	0	0	1	0	0
Chromosome B	0	1	1	1	1	0	1	0	1	1

Figure 3-3 Example of a chromosome with a 10-bit binary encoding

2. Permutation Encoding: Every chromosome is a string of numbers, which represents numbers in a sequence. This type of encoding is only useful for ordering problems.

Chromosome A	1	4	2	9	0	3	5	8	7	6
Chromosome B	8	7	9	2	6	4	5	0	1	3

Figure 3-4 Example of a chromosome with a 10-bit permutation encoding

3. Value Encoding: Direct value encoding can be used in problems where complicated values such as real numbers are used. Use of binary encoding for this type of problems would be very difficult. Values of the alleles can be anything related to the problem, whole numbers, real number, characters, or even objects.

Chromosome A	1.23	6.75	9.31	0.73	5.52	7.11
Chromosome B	A	C	B	C	B	D
Chromosome C	(up)	(down)	(left)	(up)	(right)	(down)

Figure 3-5 Example of a chromosome with a 6-bit value encoding

3.2.5 Selection of the fittest

Selection of the fittest is the cornerstone of the operation of the GA. From a broader point of view it can be considered as the process of creating an intermediate population. It is this intermediate population that undergoes the genetic operations of crossover and mutation. There are two distinct methods of performing the selection process, namely roulette wheel selection and rank selection.

1. Roulette wheel selection: The population is mapped onto a roulette wheel, where each individual is represented by a space that proportionally corresponds to its fitness. By repeatedly spinning the roulette wheel, individuals are chosen using "stochastic sampling with replacement".

A selection process that will more closely match the expected fitness value is "*remainder stochastic sampling*". For each string *i* where f_i / f_{avg} is greater than 1.0, the integer portion of this number indicates how many copies of that string will be copied to the intermediate population. All strings (including those with $f_i / f_{avg} < 1.0$) are then chosen with a probability corresponding to the fractional probability of f_i / f_{avg}. For example, a string with $f_i / f_{avg} = 2.3$ is chosen twice and then receives a 0.3 chance of placing a third copy.

Remainder stochastic sampling is most efficiently implemented using a method known as "*stochastic universal sampling*". Assume that the population is laid out in random order as in a pie graph, where each individual is assigned space on the pie graph in proportion to fitness. Next an outer roulette wheel is placed around the pie with N equally spaced pointers. A single spin of the roulette wheel will now simultaneously pick all X members of the intermediate population. The resulting selection is also unbiased.

2. Rank selection: Roulette wheel selection may cause the population to prematurely converge to a non-optimum solution. This situation occurs when the population consists of few chromosomes having a very high fitness value compared to the remainder of the population. These fit chromosomes will have a high probability of selection compared

to other unfit members. The reproduced population will inevitably be dominated by the fit members, causing potentially rich genetic information to be lost from the population.

An alternative selection process can be used. Instead of representing the chromosomes as spaces corresponding to their relative fitness, chromosomes are gives a ranked fitness. The worst will have a fitness of 1, second worst 2 and so on, the best chromosome will have a fitness of N (number of chromosomes in the population). Consider the 6 chromosomes having the fitness values shown in Table 3-3.

Table 3-3 Comparison between Raw selection and Rank selection

Chromosome	Raw Selection		Rank Selection	
	Fitness	Relative Fitness	Fitness	Relative Fitness
1	100	0.55	6	0.29
2	25	0.14	5	0.24
3	20	0.11	4	0.19
4	20	0.11	3	0.14
5	10	0.055	2	0.09
6	5	0.027	1	0.05

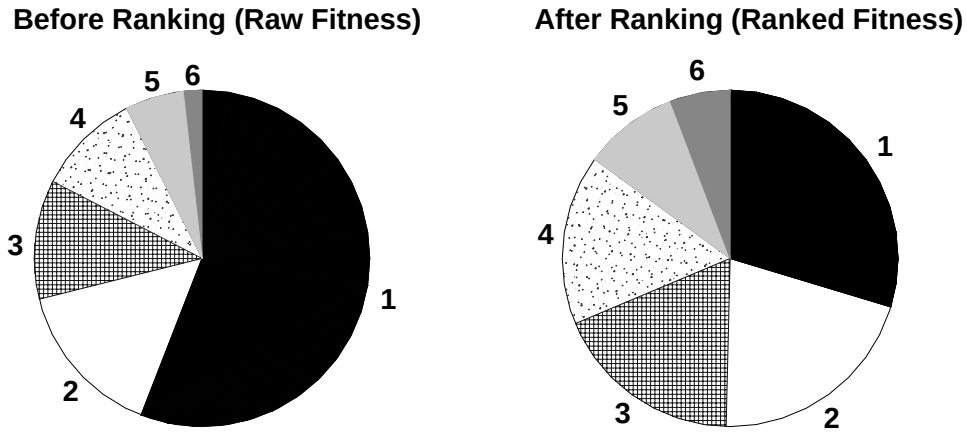


Figure 3-6 Comparison between Raw fitness and Ranked fitness after mapping on a roulette wheel

After selection has been carried out the construction of the intermediate population is complete and recombination (crossover – mutation) can occur.

3.2.6 Crossover

Crossover can very simply be defined as a process in which the newly reproduced strings are randomly coupled, and each couple of the string partially

exchanges information. There are various methods for performing crossover.

1. Single point crossover: One crossover point is randomly selected. String form the beginning of the chromosome to the crossover point is copied from one parent; the rest is copied from the second parent.

Parent A	**1**	**1**	**0**	**1**	*0*	*0*
Parent B	*0*	*1*	*1*	*1*	**1**	**0**

Crossover Point

Offspring 1	**1**	**1**	**0**	**1**	**1**	**0**
Offspring 2	*0*	*1*	*1*	*1*	*0*	*0*

Figure 3-7 Example of a single point crossover with a binary encoding

2. Two-point crossover: Two crossover points are randomly selected. This method copies string from the first parent, starting from the beginning of the chromosome to the first crossover point and from the second crossover point to the end of the chromosome. The remainder is copied form the second parent.

Parent A	**1**	*1*	*0*	**1**	**0**	**0**
Parent B	*0*	**1**	**1**	*1*	*1*	*0*

1st Crossover Point **2nd Crossover Point**

Offspring 1	**1**	**1**	**1**	**1**	**0**	**0**
Offspring 2	*0*	*1*	*0*	*1*	*1*	*0*

Figure 3-8 Example of a Two-point crossover with a binary encoding

3. Uniform crossover: Bits are randomly selected from both parents creating the offspring.

Parent A	**1**	*1*	**0**	*1*	**0**	**0**
Parent B	*0*	**1**	*1*	**1**	*1*	*0*

Offspring 1	**1**	**1**	**0**	**1**	**0**	**0**
Offspring 2	**0**	**1**	**1**	**1**	**1**	**0**

Figure 3-9 Example of a uniform crossover with binary encoding

3.2.7 Mutation

The main motivation for using mutation is to prevent the permanent loss of any particular allele. After several generations it is possible that selection will drive all alleles in some position to a single value. If this happens without the genetic algorithm converging to a satisfactory solution then the algorithm has prematurely converged. This may particularly be a problem if one is working with a small population. Without a

mutation operator, there is no possibility for reintroducing he missing bit value (Whitley, 1993).

Generally speaking, mutation involves the random alteration of a bit value in the chromosome. The probability of mutation occurrence is usually small compared to crossover. Mutation probabilities are usually in the range of (0.005 - 0.01), while crossover probabilities are usually in the range of (0.4 - 0.8). Two methods for performing mutation will be discussed below.

1- Bit inversion: Performed only with binary encoding. A randomly selected bit (or bits) is inverted from 0 to 1 or visa versa.

Figure 3-10 Example of 2-bit inversion with binary encoding

2- Order changing: Two alleles are randomly selected and exchanged.

Figure 3-11 Example of order changing with value encoding

3.3 THE OBJECTIVE FUNCTION: RELATIVE WEIGHTS VS. COST DATA

In almost all optimization approaches, when performing static layout, the objective function to be minimized, takes the general form:

$$Min: \left(\sum_{i=1}^{P-1} \sum_{j=i+1}^{P} W_{i,j} * d_{i,j} \right) \quad \text{..... (3-7)}$$

Where:

P= Total number of fixed and temporary facilities present.

$d_{i,j}^{t}$ = Distance between facilities i and j

In the literature, the term $W_{i,j}$ represents one of the following:

1- Transportation cost data of some sort or

2- A relative proximity weight that reflects the required closeness between any two facilities.

In all optimization problems the clear identification of a *goal* to attain is essential. Thus, the first representation has the clear objective of minimizing total transportation costs between site facilities. The objective is not as apparent when using

the relative weight representation. With the second representation one might argue, "What exactly are we trying to achieve?"

Several scales have been adopted in order to facilitate the verbal representation of the proximity weight. One of the common scales used in industrial facility layout planning is shown in figure-. In the previous chapter we outlined how these weights were quantified using a fuzzy rule based system (Elbeltagi & Hegazy 2001). The main advantage of using the relative weight representation is the great difficulty in obtaining accurate inter-facility transportation cost data. Using a relative proximity weight may be much easier for the site planner to provide. One of the common proximity weight representations used in industrial facility layout planning is shown in Table 3-4. (Askin & Standrige, 1993)

Table 3-4 The six value closeness relationship values used in industrial facility layout planning

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

On the other hand, when performing dynamic layout, the objective function takes the general form:

$$Min: \left(\sum_{i=1}^{P} \sum_{j=1}^{P} W_{i,j} * d_{i,j} \right) + \text{Relocation Cost of Facility}$$

Relocation costs of facilities are more easily quantifiable than inter-facility transportation costs. Formulating a relative relocation weight to be used with the proximity weight is quite complex. In this research, the objective function to be minimized takes the general form:

Layout Cost = Transportation Cost (T.C.) + Relocation Cost (R.C.)

4 SYSTEM DEVELOPMENT

This chapter provides details of the automated CAD-based GA system used for the dynamic layout planning of construction sites. The chapter begins with an overview of the system structure as a whole. Afterwards the various modules comprising the system are thoroughly explained. These include the space detection module, the constraint satisfaction module and the GA-based optimization procedure. The coding for all modules is provided in the appendices of this thesis.

4.1 SYSTEM STRUCTURE

The automated site layout planning system is comprised of three main components:

1- An input facility that incorporates various types of data needed for the layout planning task.
2- An optimization engine based on the concepts of genetic algorithms.
3- An output facility that utilizes the programmable features of the CAD platform of choice.

The automated system utilizes vast amounts of data. Data used in the system can be grouped into four major categories (Table 4-1), namely, schedule data, temporary facility data, site geometrical data and facility cost data.

Table 4-1 Description of the main data types required in the model

Data	**Description**
Schedule data	Main project phases. Phases are grouped based on temporary facility requirements.
Temporary facility data	Temporary facility requirements in each phase in addition to the expected sizes of these temporary facilities.
Site geometrical data	CAD drawings representing the layout of fixed facilities in each project phase.
Facility cost data	Inter-facility transportation costs for each phase in addition to the expected cost for relocating temporary facilities.

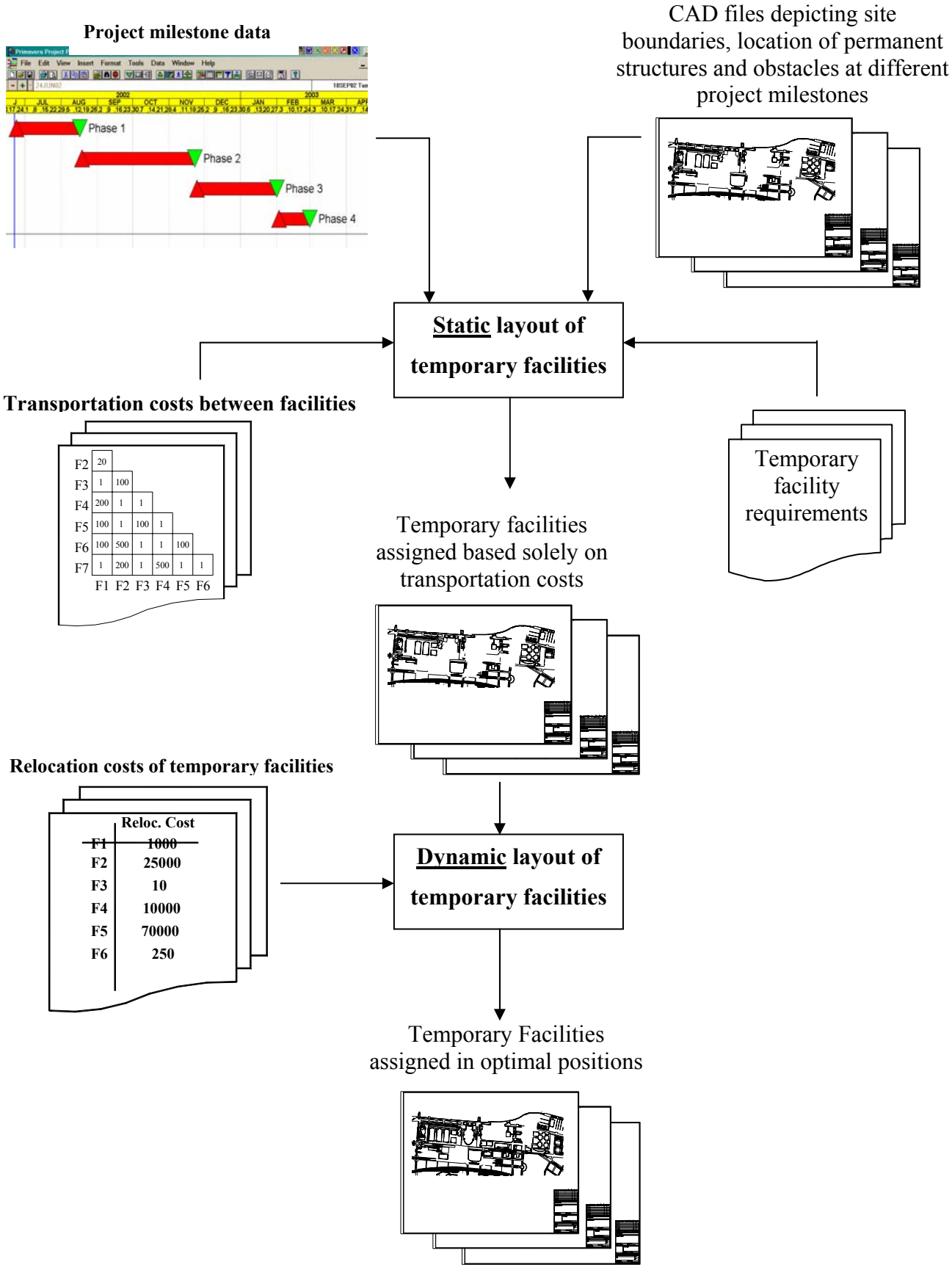


Figure 4-1 Detailed system architecture

Genetic algorithms are used to perform the optimization process, which proceeds in two stages. Firstly, static layout is performed and each phase is considered completely separate. Secondly, dynamic layout is performed taking layout continuity into consideration. Following the optimization process, the model delivers a series of CAD drawings each depicting a particular construction phase with all temporary facilities placed in their optimal positions. The detailed system architecture is illustrated in figure 4-2. The figure shows where the system process the different types of data described in table 4-1.

4.2 SPACE IDENTIFICATION

The functionality of the optimization engine largely depends on identifying the specifics of the CAD drawing (i.e., site boundaries, permanent facilities, and obstacles). Accurately identifying the available space on site for assigning the temporary facilities is essential in order to yield a feasible solution. Available space on site is detected through the algorithm explained hereafter. The main concept in space detection is that of space discretization, that is the division of space into an orthogonal two-dimensional grid. This grid is then coded, each grid cell having a unique (X,Y) coordinate as illustrated in Figure 4-1. Regardless of the site geometry, the system is able to capture all geometric data lying within the site boundaries, after which removal of space occupied by permanent facilities occurs.

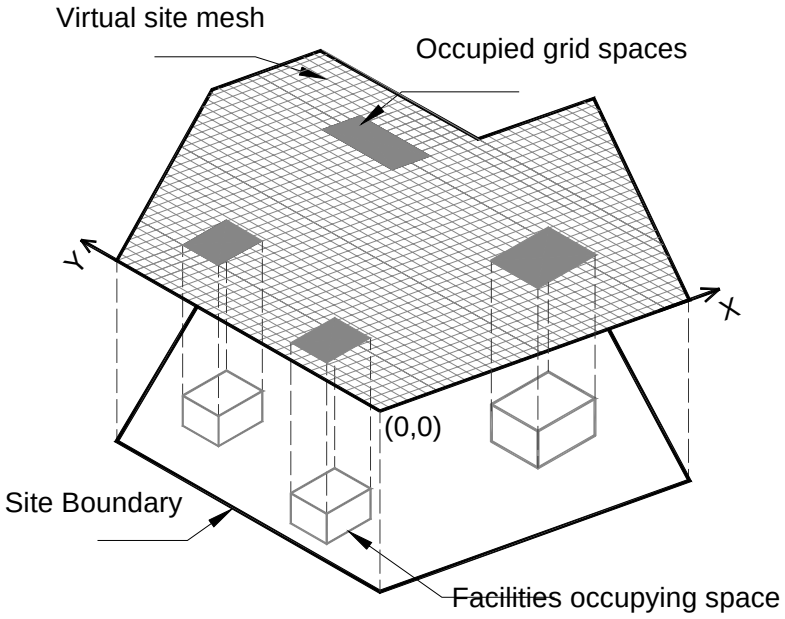


Figure 4-2 Details of space discretization

The available space on site is detected using the capabilities of the CAD platform of choice. In the widely used AutoCAD™, a macro can be employed to perform this function. Generally, two steps must be accomplished, first the site boundaries are identified, second fixed facilities and obstacles occupying space are identified.

4.2.1 Identification of enclosed site space:

In order to perform this task the user must identify the site boundaries (must be an AutoCAD™ *LWpolyline*) and any point lying inside the site boundaries. Performing this task proceeds as follows:

1. Using the rectilinear coordinates of the site boundary's vertices, the CAD macro identifies the edges of the site boundary as a set of equations:

   $y = a_1 x + b_1 \quad y = a_2 x + b_2 \quad y = a_3 x + b_3 ..... \quad y = a_n x + b_n$

2. Using a point inside the boundary, the CAD macro identifies the edges of the site boundary as a set of inequalities:

   $y \le a_1 x + b_1 \quad y \ge a_2 x + b_2 \quad y \ge a_3 x + b_3 ..... \quad y \ge a_n x + b_n$

3. By toggling through coordinates of all points inside the site boundary's "Bounding Box" (Figure 4-3), the macro chooses only those points that satisfy all linear inequalities simultaneously.
4. Feasible grid squares are stored in 2 arrays: AvailableX( ) and AvailableY ( ) such that any point "i" has coordinates (AvailableX(i), AvailableY(i))

4.2.2 Identification of fixed facilities and obstacles

Till this stage no account has been made for obstacles or fixed facilities. Obstacles are defined as site facilities with fixed positions having no closeness relationship with other facilities. Fixed facilities on the other hand have fixed positions but maintain closeness relationship with other facilities. (Hegazy & Elbeltagi 1999). In order to perform this task the user must identify all obstacles and fixed facilities present on site (must be an AutoCAD™ *LWpolyline*). Performing this task proceeds as follows:

1- Grid spaces occupied by these obstacles are removed from the arrays AvailableX and AvailableY.
2- Grid spaces occupied by these fixed facilities are removed from the arrays AvailableX and AvailableY. The coordinates of their centroid are stored for future use in the optimization procedure.

After these two sub-tasks are accomplished, available site space, obstacles and fixed facilities are translated into a set of X,Y coordinates. These coordinates will be utilized in the optimization procedure during the layout of the temporary facilities.

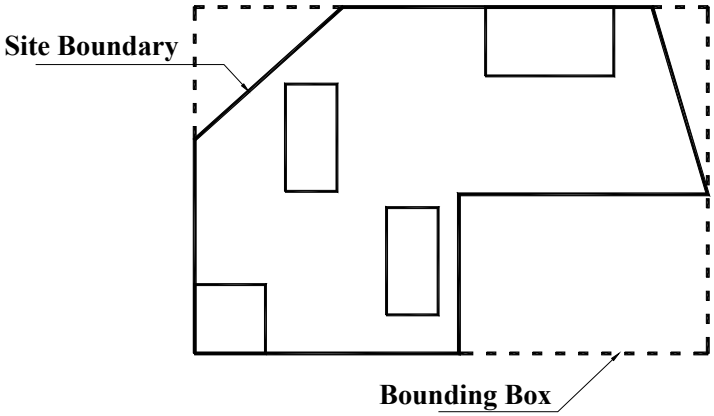


Figure 4-3 Bounding box of a polygon

4.3 GA STRING CODING

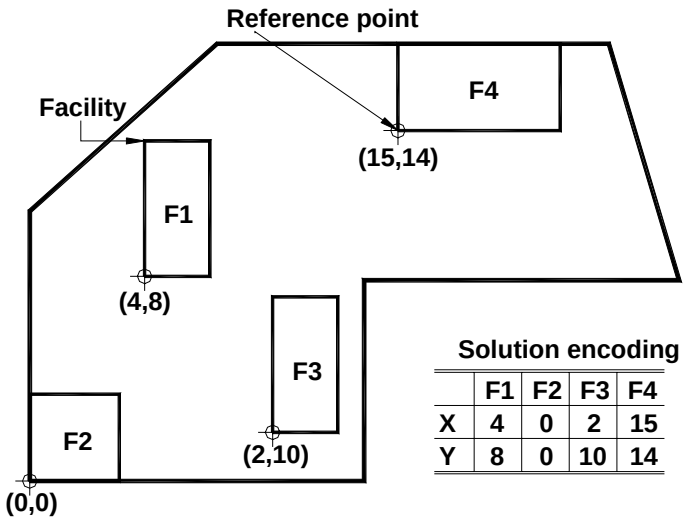
The optimization procedure itself depends primarily on GA optimization. As discussed in chapter 3, the coding of a solution to a string is one of the most important aspects of success for GA optimization. The coding scheme of any GA is the essence of its success. The coding scheme used in the presented model is a modified version of that adopted by Hegazy & Elbeltagi (1999). As their model depended largely on utilizing a set of grid cells to represent site space; each grid cell was uniquely coded based on the location of the row and column in which the cell is located. Site facilities were then referenced via these developed codes (figure 2-5). The model by Hegazy & Elbeltagi (1999) was implemented using MS-Excel.

In this research the case is quite different. The main dependency on geometrical data is by space detection through the CAD platform. It was seen previously that available space is represented by a set of X and Y coordinates. Therefore there is no need for the unique coding of grid spaces, as each grid space in uniquely identified by a set of two numbers (i.e., the X and Y coordinates). The coding of a certain solution representing the assignment of 4 temporary facilities is illustrated in Figure 4-4.

It is to be noted that the temporary facilities are arranged in descending order of size. That is: F1 > F2 > F3 > F4 > ….> Fn. The reason for this pre-sorting will be explained in the following section.

Although the position of fixed facilities affects the value of the objective

function, they are not represented in the string as their positions are fixed making their allele values in the chromosome constants.



	F1	F2	F3	F4
X	4	0	2	15
Y	8	0	10	14

Figure 4-4 GA string encoding of 2D space

4.4 CONSTRAINT SATISFACTION

Geometrical constraints are vital in the layout process. It is of utmost importance that temporary facilities be placed (1) Inside our site boundaries and (2) In such a manner that no overlap occurs between any two temporary facilities or between temporary facilities and fixed facilities.

To achieve the satisfaction of geometrical constraints, two main modules are utilized, namely *"CheckSite"* and *"CheckOverlap"*. These modules rely on 5 main variables in their operation as shown in Table 4-2.

The constraint satisfaction algorithm is applied on each facility in ascending string position (ie. F1 , F2, F3…).

```
For each Temporary Facility
CheckSite(Xmin, Ymin, FacilityX, FacilityY)
CheckOverlap(Xmin, Ymin, FacilityX, FacilityY)
Next Facility
```

Where:

Xmin and *Ymin* are gene values for current facility (coordinates of bottom left corner of facility) and *FacilityX* and *FacilityY* are the dimensions of the

temporary facility in the X and Y directions respectively.

Table 4-2 Main variables required in the constraint satisfaction procedure

Variable Name	Description
AvailableX( )	Available X-coordinates for assigning temporary facilities
AvailableY( )	Available Y-coordinates for assigning temporary facilities
OccupiedX( )	Space currently occupied by temporary facility
OccupiedY( )	Space currently occupied by temporary facility
ReservedPTS	Number of grid units currently occupied by temporary facilities

It is now clear why it is very important that facilities be sorted in descending order of size from F1 to Fn, as placing small facilities early on site may not leave sufficient room for other larger facilities to be placed later on. Other researchers have used different heuristics to guide their order of assignment. Zouein & Tommelien (1999) choose the facilities based on their relative interaction with other facilities as represented by the sum of their proximity weights. In tight congested sites, using this heuristic alone may lead to infeasible site layouts. Primarily because small facilities – having high proximity weights with other facilities – may be assigned to spacious areas that are more suited to place other larger facilities. Starting the assignment procedure in descending order or facility size ensures that this premature solution infeasibility does not occur.

4.4.1 CheckSite module:

This module is a built-in function that makes sure that any temporary facility (1) lies inside the site boundaries and (2) does not overlap with any fixed facility or site obstacle.

This function requires as input 4 variables; [Xmin, Ymin, FacilityX and FacilityY]. It provides a Boolean true/false output. In its operation it toggles through all grid coordinate occupied by the facility and compares them with the arrays AvailableX and AvailableY.

```
If any (X,Y) of facility ⊄ Available(X,Y) then CheckSite = False
Else CheckSite = True
```

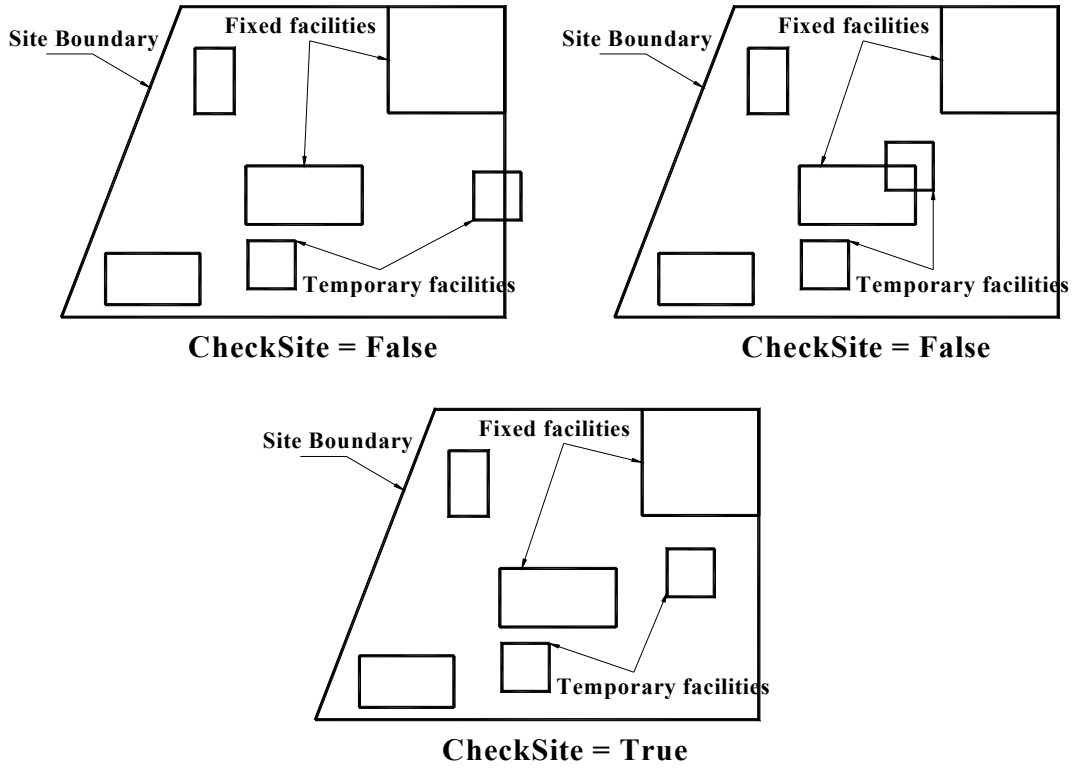


Figure 4-5 Functionality of the *Checksite* module

4.4.2 CheckOverlap module

This module is a built-in function that performs two sequential tasks.

a- Makes sure that the facility being checked does not occupy space already being reserved for another facility that has been assigned on site.

`If any (X,Y) of facility ⊂ Occupied(X,Y) then CheckOverlap = False`

`Else CheckOverlap = True`

b- If no overlap occurs space is reserved for the facility

`For ∀ (X,Y) of facility, Occupied(X,Y) = Facility(X,Y)`

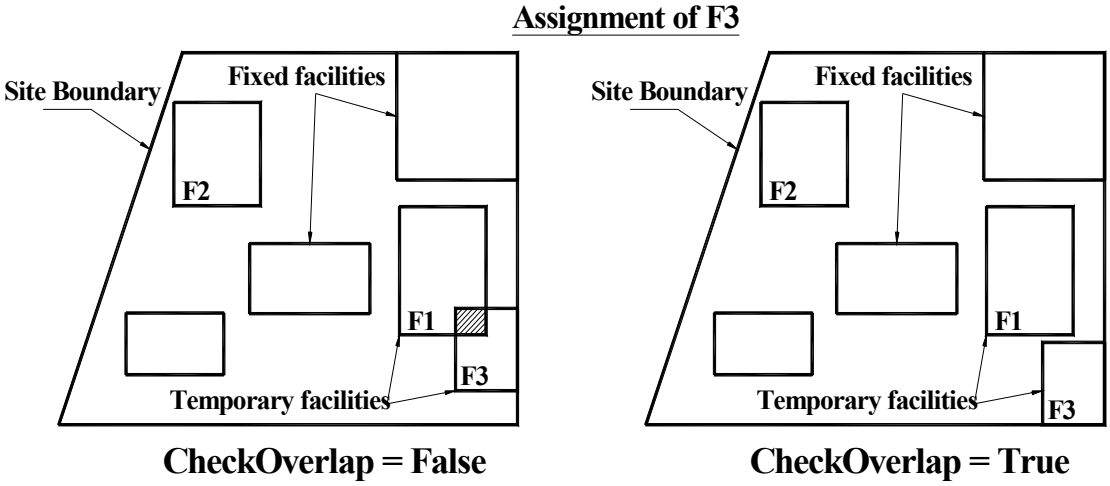


Figure 4-6 Functionality of the *CheckOverlap* module

4.5 INTER-FACILITY COST MATRIX

The inter-facility transportation cost matrix contains the most important optimization relevant data. Using these numbers, the optimization procedure will assign facilities close to or far from one another. The cost data is represented in a lower triangular matrix as shown in Table 4-3 for 6 temporary facilities and 3 fixed facilities. As shown in the table, no inter-facility costs data exists for fixed-to-fixed facilities.

Table 4-3 Inter-facility cost matrix for 6 temporary facilities and 3 fixed facilities

Temporary Facilities	F1								
	F2	W_{12}							
	F3	W_{13}	W_{23}						
	F4	W_{14}	W_{24}	W_{34}					
	F5	W_{15}	W_{25}	W_{35}	W_{45}				
	F6	W_{16}	W_{26}	W_{36}	W_{46}	W_{56}			
Fixed Facilities	F7	W_{17}	W_{27}	W_{37}	W_{47}	W_{57}	W_{67}		
	F8	W_{18}	W_{28}	W_{38}	W_{48}	W_{58}	W_{68}	N/A	
	F9	W_{19}	W_{29}	W_{39}	W_{49}	W_{59}	W_{69}	N/A	N/A
		F1	F2	F3	F4	F5	F6	F7	F8	F9

Temporary Facilities | Fixed Facilities

4.6 OPTIMIZATION PROCEDURE

The GA-based optimization procedure utilized is a steady state GA generation that utilizes single-point crossover and a modified mutation operator. Details of the objective function to be optimized and the GA procedure are fully described in the following section. The flowchart for the GA is illustrated in Figure 4-8.

4.6.1 Objective function

The objective function that is evaluated in the optimization process is a modified version of that adopted by Zouein & Tommelien (1999). The objective function to be minimized is:

Layout Cost = Transportation Cost (T.C.) + Relocation Cost (R.C.)

$$T.C. = \sum_{i=1}^{P-1} \sum_{j=i+1}^{P} W_{i,j}^{t} * d_{i,j}^{t} \quad \text{..... (4-1)}$$

$$R.C. = \sum_{i=1}^{P} R_{i}^{t} * d_{i}^{(t-1)t} \quad \text{..... (4-2)}$$

Where:

P = Total number of fixed and temporary facilities present.

$W_{i,j}^{t}$ = Transportation costs / unit distance between facility i and j during the current phase t

$d_{i,j}^{t}$ = Distance between facilities i and j during the current phase t

R_{i}^{t} = Cost of relocating facility i during the current phase t

$d_{i}^{(t-1)t}$ = Distance that facility i has moved from phase t-1 to phase t

In the newly developed model the term R.C. depicting the relocation cost has been slightly modified. Generally, relocation cost of a construction facility can be represented by the following equation:

R.C. = Fixed Cost + Variable Cost..... (4-3)

Where the fixed cost represents those costs spent on dismantling, re-installment, delay and providing an alternative facility. Variable costs are usually attributed to hauling, and are a direct function of hauling distance. In construction site planning, the cost of relocating a facility is not greatly dependant on the distance of relocation but on

whether relocation has taken place or not. When considering most construction related facilities, fixed costs are far larger than variable costs. Accordingly, the following representation for relocation cost can be used:

$$R.C. = \sum_{i=1}^{P} R_i * Oc_i^{(t-1)t} \quad \text{..... (4-4)}$$

Where Oc_i is a binary variable that takes only 0 or 1

If facility i has been relocated from phase (t-1) to phase t, then $Oc_i^{(t-1)t} = 1$

If no relocation has occurred $Oc_i^{(t-1)t} = 0$

4.6.2 Initialization of Population

Any GA procedure starts with an initial population of solutions. The number of initial solutions generated influences the GA. It is known that increasing the population size has the following effects on the GA:

1- Tremendously increases the time required for generating a new population.

2- Causes a very slow convergence rate.

3- Causes the GA to reach more optimum solutions.

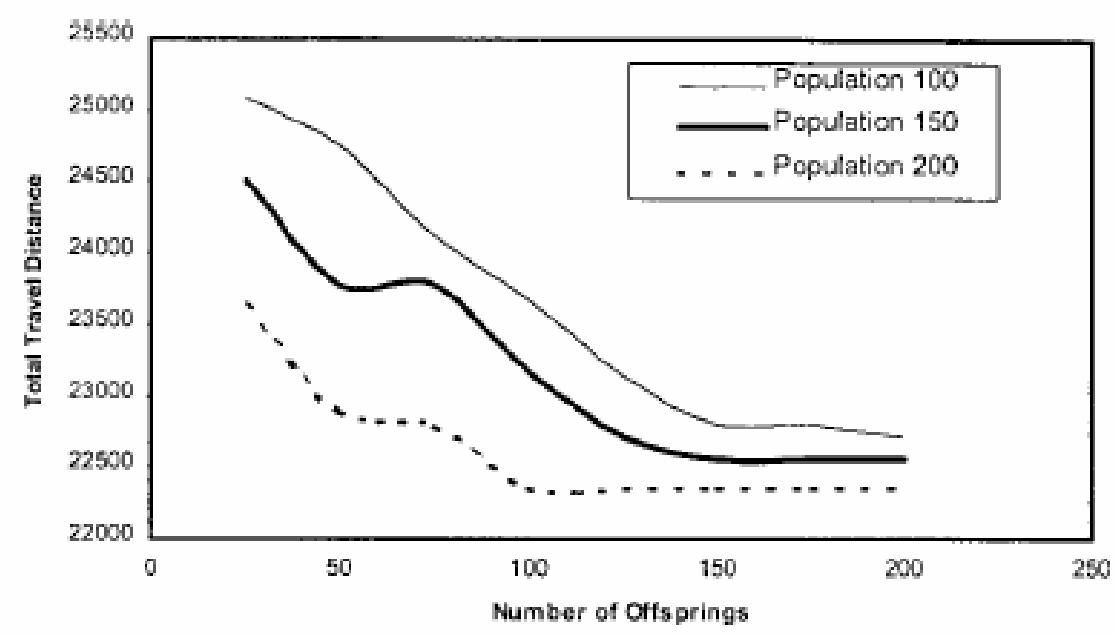


Figure 4-7 Effect of population size on optimum solution (Hegazy & Elbeltagi, 1999)

In order to assist the GA in its blind search, a slight enhancement has been proposed. Instead of working with a very large population throughout the GA, the initial population is selected as the best 'n' solutions from an initial pool of 'N' solutions

where n < N. Before running the GA, both the initial pool "N" and the required population size "n". Thus the GA benefits from the presence of a large initial population that assists its random search without paying the large computational penalty posed by dealing with a large population at each generation.

4.6.3 GA Generations

The generation process proposed in this research is a slight modification of that used by Hegazy & Elbeltagi (1999). Traditional GA's move from generation(i) to generation(i+1) via the generation of a new population. The approach proposed by Hegazy & Elbeltagi (1999) moved from one generation to the other via the introduction of two new offspring that would replace the worst two solutions in the population. In case the new offspring were not better than the worst solutions, they were discarded and two other offspring were chosen.

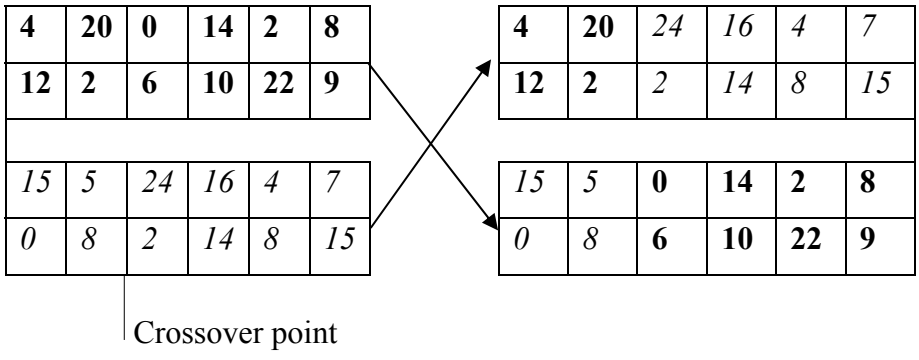
Elimination of the worst offspring meant that as each new solutions are introduced, the population as a whole would improve. It also meant a chance for randomly bred offspring to even outperform the best solution in the population. Generation of new offspring involves the 3 traditional genetic operators:

1- Replication

Traditional roulette wheel selection is performed based on the fitness value for individual solutions. Two random solutions are chosen to replace the worst two solutions in the population. There is no need to check for the feasibility of the solutions, as no modifications were performed (crossover, mutation).

2- Crossover:

The same selection procedure is applied to select the parents that will be crossed. Simple single-point crossover was used so as to minimize the disruption of the schemata.



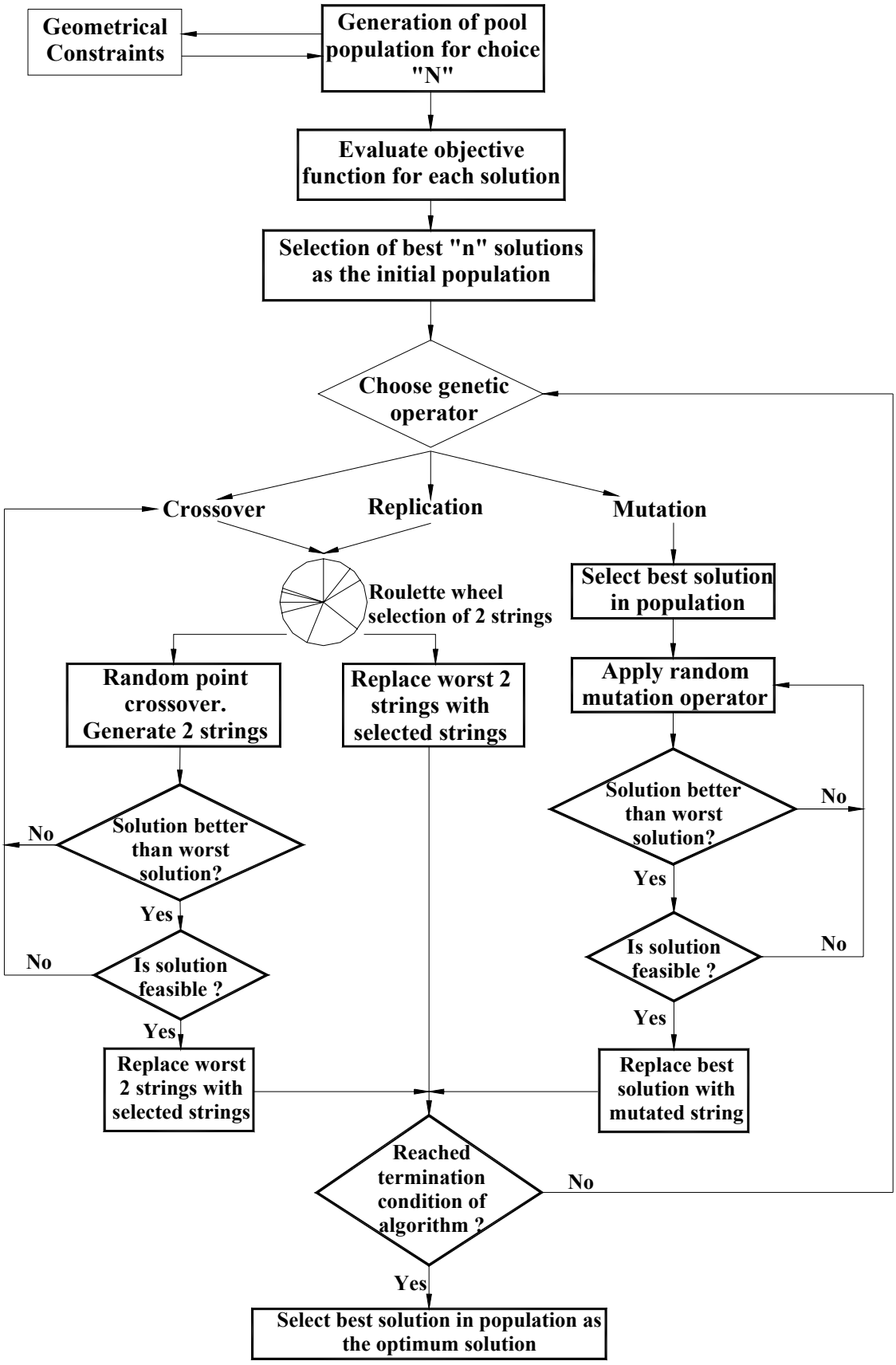


Figure 4-8 Genetic Algorithm Flowchart

After the crossing two checks are performed on the offspring:

1- Constraint satisfaction: To verify the feasibility of the new solutions.

2- Objective function improvement: To verify that the new solutions are not worse that those being replaced.

3- Mutation:

Mutation is used mainly to break current stagnation in improvement by introducing new genetic information into the population. It was noticed during the development of the system, that performing the GA without mutation led to solutions that required slight refinements to reach more optimum solutions. These refinements usually involved very small movements of one or more facilities in a specific direction.

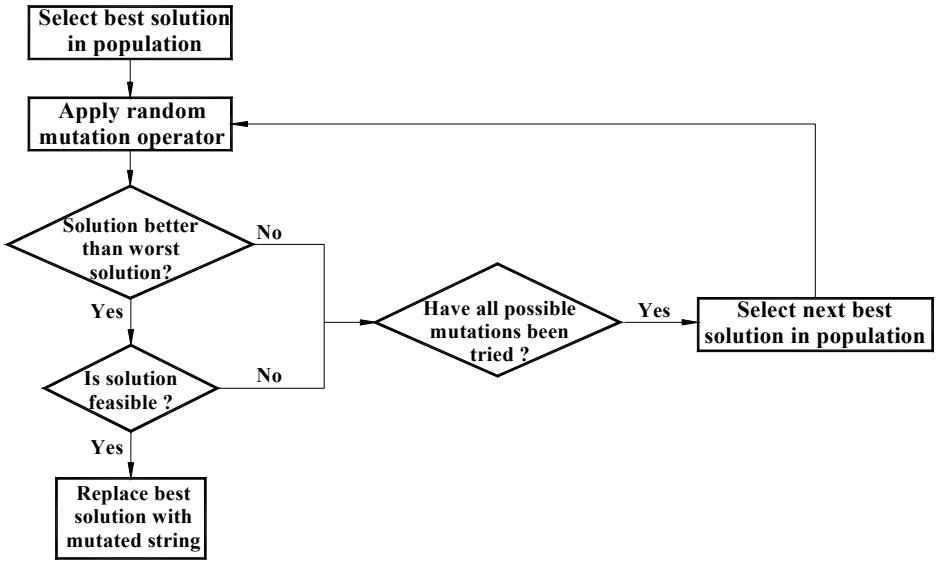


Figure 4-9 Mutation Operator Flowchart

A modified mutation operator is developed in this research to attain the required function. The mutation operator randomly choose the following:

1- The facility to be moved.

2- Whether the movement should be in the X or Y direction.

3- Whether the movement should be in the +ve or –ve direction.

And then applies a movement of 1 unit to the facility chosen and in the direction chosen. Similarly, the new solution is checked for constraint satisfaction and objective function improvement. If violated, the mutation procedure is repeated. The mutation operator flowchart is shown in Figure 4-9.

4.6.4 Convergence Condition

In any GA the generation process must proceed until a certain termination condition is reached. Researchers have used various convergence conditions:

1- Until no further improvement in the population occurs (Al-Tabtabi & Alex, 1998)

2- Until all offspring in the population are replaced (Hegazy & Elbeltagi, 1999)

3- Until there is very little variation within the population itself.

In our system, the generation process continues until the following convergence condition is reached:

$$\Delta < Convergence$$

Where

$$\Delta = \frac{Max - Min}{Max}$$

Convergence: User specified tolerance (usually 5-10%).

Min: minimum solution in current population

Max: Maximum solution in current population

4.6.5 Dynamic Optimization

Until now, all discussion of the optimization process was only for a single phase in the project. The layout process itself is dynamic in nature, thus the optimization process must also reflect this dynamics. The optimization process begins with the static optimization of all phases based solely on transportation costs. It treats each phase as if it were a completely separate phase with no interaction with preceding or succeeding phases. After static optimization of all layouts are complete, one of two dynamic optimization techniques are followed. The two techniques are called the critical phase approach and the mini-min approach (Figure 4-10).

In the past, researchers who have tried to tackle the problem of dynamic site layout planning have proceeded in chronological order (Zouein & Tommelien, 1999). This approach has its drawbacks. The main weakness lies in the fact that facilities that are assigned positions in early phases may:

1- Be placed in positions that will subsequently be occupied by permanent facilities, thus they will be forced to be relocated.

2- Be placed in positions that minimize transportation costs during early phases but in subsequent phases be in unfavorably far positions from other facilities.

As mentioned before, two approaches are employed to mitigate these costs of early assignment of facilities in positions that may seem favorable in early phases, but could turn out very costly in phases to come.

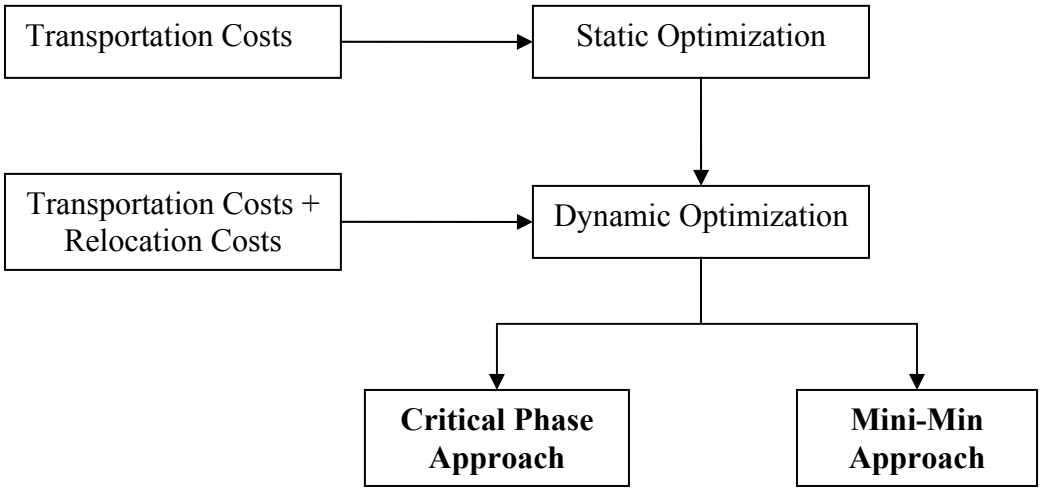


Figure 4-10 Flowchart for the optimization procedure

I- Critical Phase Approach

It is evident that the choice of the first phase to be the initial phase (where no relocation costs are calculated) may not necessarily yield the most optimum transportation + relocation cost for all phases combined. The key to finding the optimum solution for all phases lies in identifying an *initial* phase whose layout will be solely dependant on the transportation cost. Dynamic optimization should proceed in forward chronological order for succeeding phases and backward chronological order for preceding phases as illustrated in Figure 4-11.

This approach is based on the prevalence of a phase in the project having several on- site temporary facilities, a lot of ongoing site movement and a relatively long time span. Thus during this *critical phase*, transportation costs are more likely to be much higher than other phases. During planning, extra care should be provided to this phase. Performing facility layout based solely on transportation costs in phases other than the critical phase may lead to facilities being placed in a position that is not optimum during this critical phase. Transportation costs in this critical phase could greatly boost. The overall transportation costs for all phases could be far from optimum.

The critical phase approach highlights the importance of this critical phase. The approach selects the phase having the highest transportation costs (previously obtained from static optimization) as being the *initial* phase. Dynamic optimization proceeds in

backward chronological order for all phases preceding the critical phase and in forward chronological order for all phases succeeding the critical phase. It is to be noted that this approach does not necessarily give the most optimum solution. It is merely suggested as an enhancement to approaches that begin the layout process with the first phase in the project sequence.

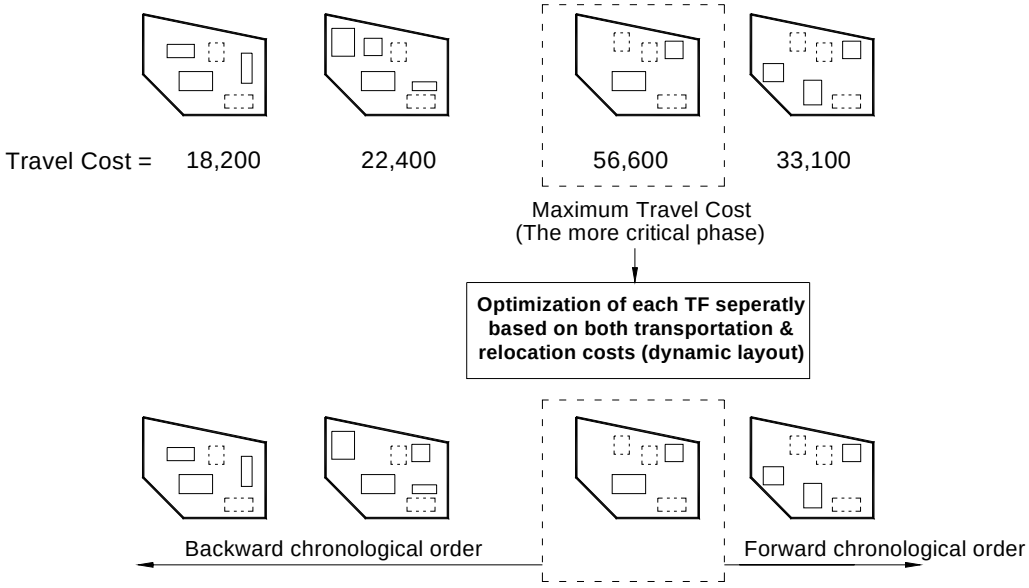


Figure 4-11 Critical phase approach in dynamic optimization

II- Mini-Min Approach

Due to the fact that the critical phase approach does not necessarily yield the most optimum solution, the Mini-Min approach is introduced. The main weakness in the critical phase approach is that the phase having the largest transportation cost might not necessarily be the initial phase. This approach is a slight enhancement of its critical phase counterpart.

The Mini-Min approach considers all possibilities for choosing the critical phase. It performs the dynamic optimization of all phases N_{phase} times, N_{phase} being the number of phases. It calculates the total costs for all phases N_{phase} times and chooses the trial having the least cost as the *Minimum-Minimum* solution. The Figure 4-12 illustrates this process for a four-phase project.

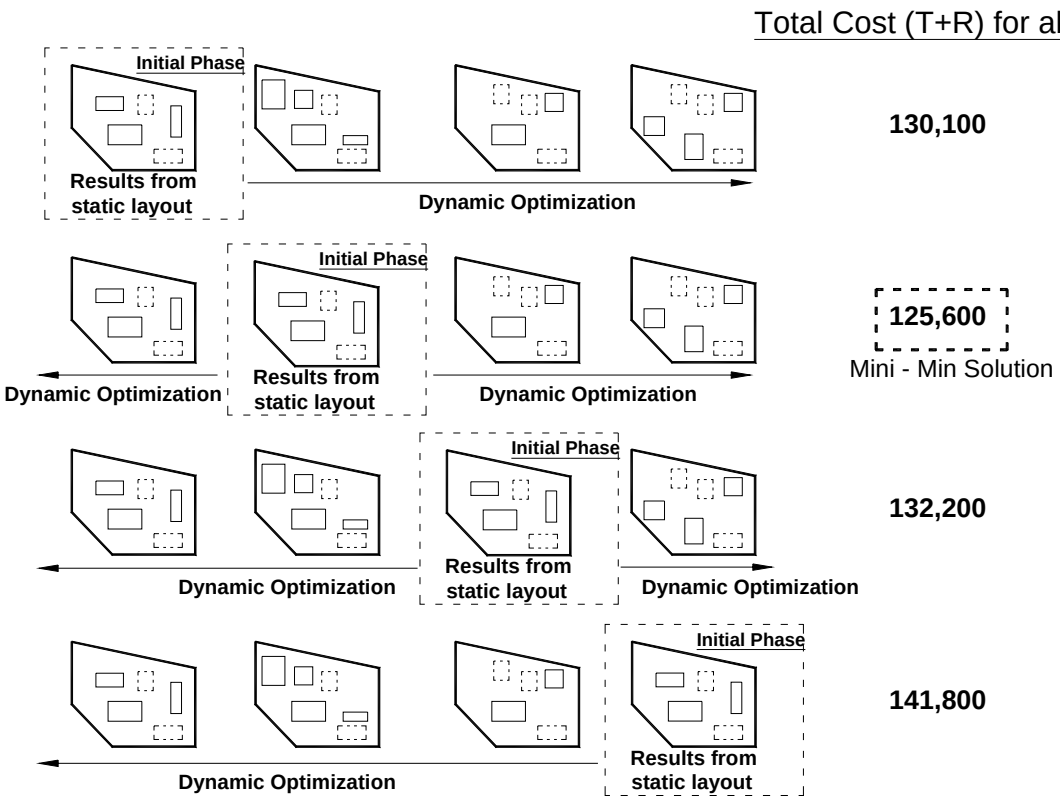


Figure 4-12Mini-Min approach for dynamic optimization

It may seem that the Mini-Min approach performs the dynamic optimization problem far too many times and that this may be computationally exhaustive. In fact, it is. For a project comprised of N_{phase} phases, our system is required to solve N_{phase}^{2} optimization problems. But as we shall see in chapter 6 of this thesis, running the system on PC's of regular speed is computationally possible, though may require large running times.

Figure 4-13 compares the number of optimization problems required to be solved for the two approaches.

4.7 SOLUTION REPRESENTATION

Regardless of the dynamic optimization technique used, the optimum solution must be represented in a user-comprehendible form. The final step in our system is the physical representation of the optimum solution.

The system, via the *drawfacility* module automatically opens all AutoCAD drawings and draws the temporary facilities in their optimum positions as depicted by the genetic algorithm. Having the output in graphical form allows the planner to easily

visualize the relationship between the permanent structures and temporary facilities on site. Graphical solution representation proceeds as follows

```
For all project phases
    Open CAD file representing current phase
    For all temporary facilities in phase
          Draw temporary facility in optimum postion
    Next temporary facility
    Close CAD file
Next project phase
```

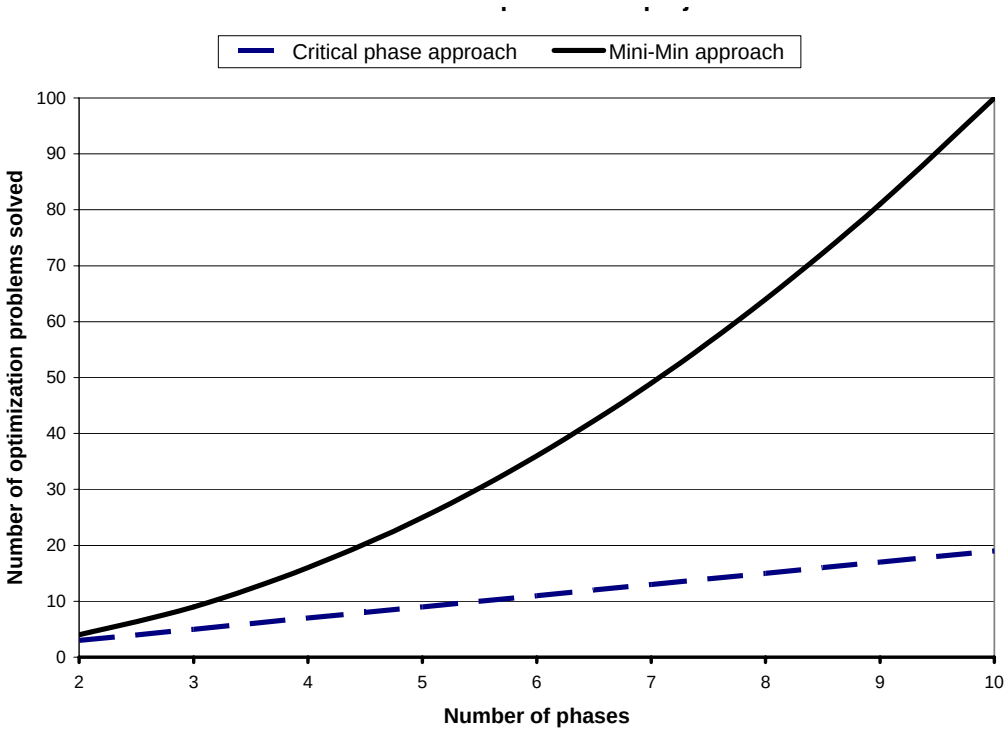


Figure 4-13 Effect of the dynamic approach used on the number of optimization problems solved

5 AUTOMATED SYSTEM & ILLUSTRATED EXAMPLE

In this chapter the automated system, EDSLP (*Evolutionary Dynamic Site Layout Planner*) is presented along with an illustrated example that demonstrates the various features of the automated system. The system has been implemented via MS VisualBasic™ 6.0. The CAD interface has been made possible through the programmable features of AutoCAD™ in the VBA environment. Code for the various modules can be found in the appendices of this thesis.

5.1 SYSTEM INPUT

The previous chapter discussed the types of data required as input to the system. Data can broadly be grouped into four categories namely, schedule data, temporary facility data, site geometrical data and facility cost data.

In the automated system, schedule data and temporary facility data are input simultaneously as shown in Figure 5-1. On each screen, the phase name, start date and end date are entered. For each phase, a list of all required temporary facilities and their anticipated dimensions is entered. Input continues until all project phases are entered.

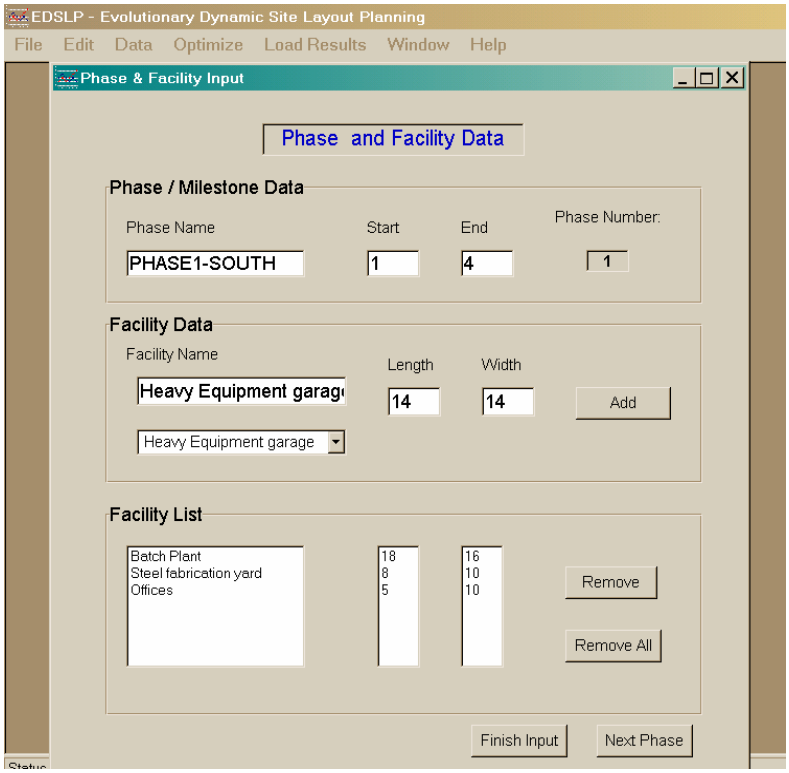


Figure 5-1 Schedule and temporary facility input screen

Meanwhile, site geometrical data input is made possible via the VisualBasic™ -

AutoCAD™ referencing capabilities. The automated system is capable of opening existing CAD drawings and reading the site layouts found in these drawings. The system detects the available site space, the fixed facilities present and any site obstacles. Hence, the available space for temporary facility placement is deduced.

CAD drawings for each phase should be appropriately prepared before the automated system process them. For each phase an individual drawing depicting the site geometry, fixed facilities and obstacles is required. The task of space detection completely takes place from within the CAD environment. Figure 5-2 shows the AutoCAD™ interactive capabilities present from within the automated system. Figure 5-3 shows the procedure for space detection operating from within the AutoCAD™ environment.

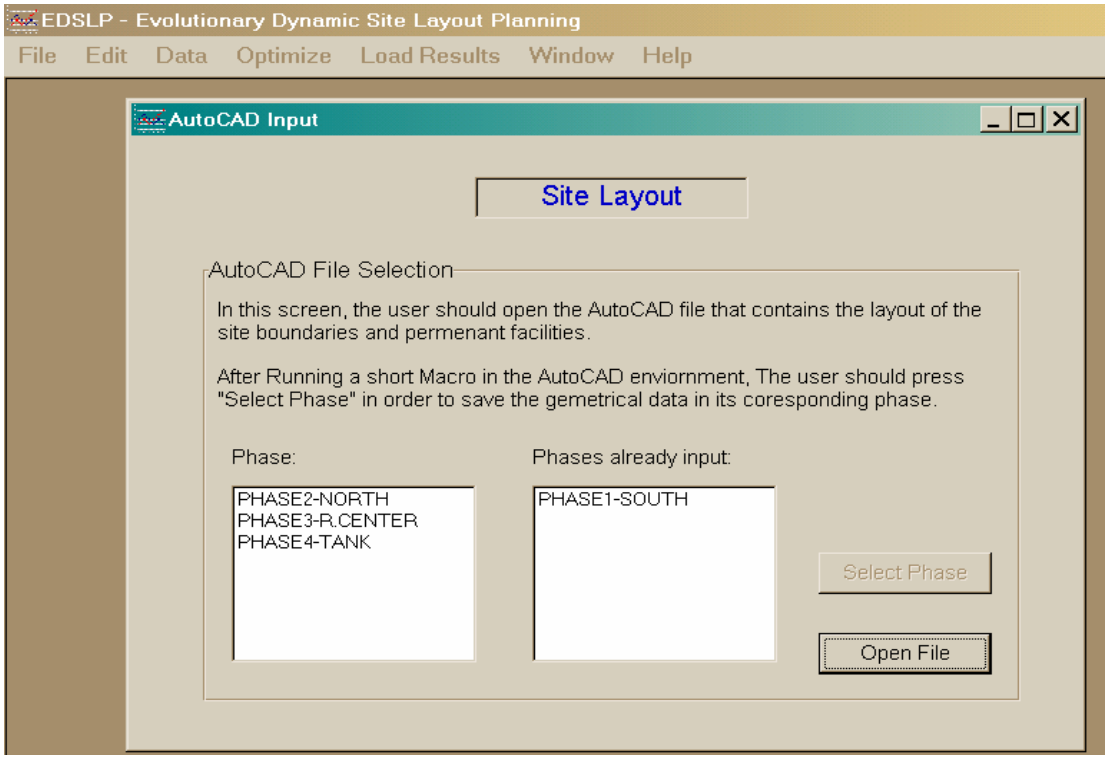


Figure 5-2 AutoCAD™ interactive capabilities from within the program environment

Facility cost data is comprised of:

1- Inter-facility transportation cost data: Input is through Excel environment in order to utilize the ready made cell divisions to mimic the required inter-facility matrix. (Figure 5-4)

2- Facility relocation cost data: A separate input screen is available from within the system's environment.

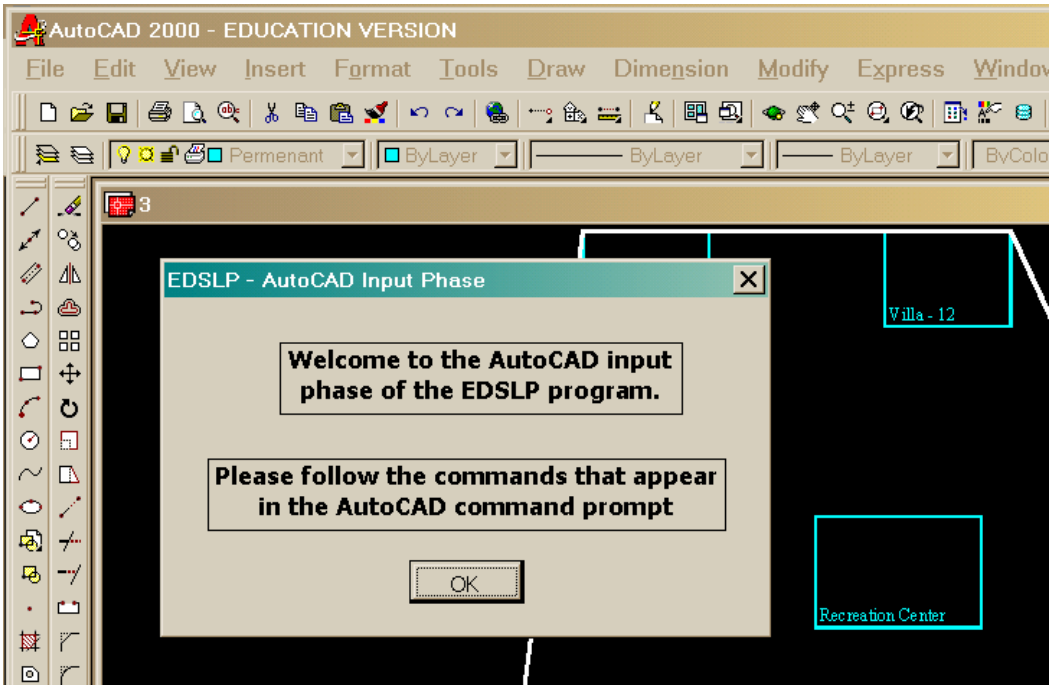


Figure 5-3 AutoCAD™ VBA macro for space detection of site layouts

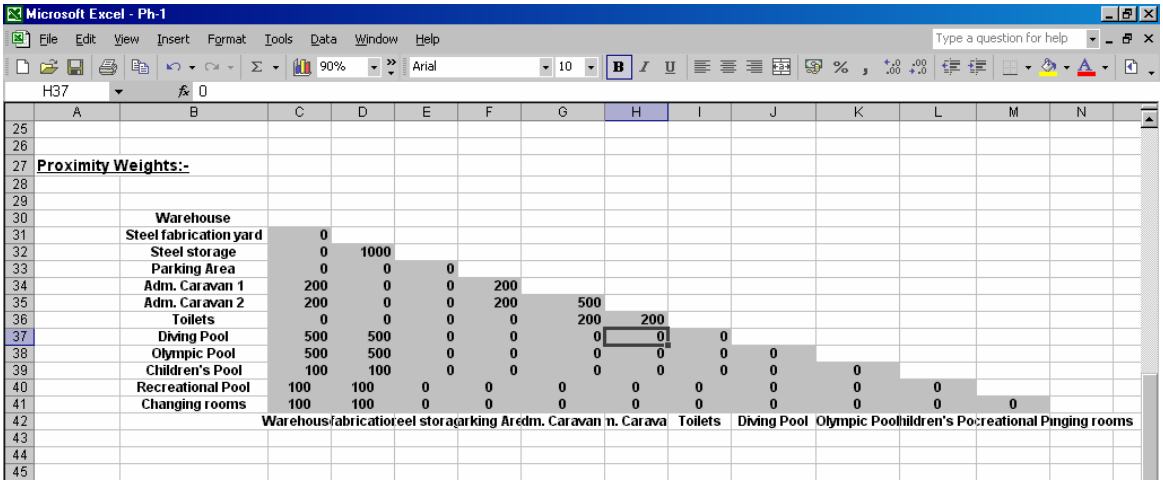


Figure 5-4 Inter-facility cost input from within the MSExcel™ environment

5.2 SYSTEM OPTIMIZATION

The optimization procedure progresses in two stages.

Stage-1: Static Optimization of individual phases taking only transportation costs into consideration.

Stage-2: Dynamic Optimization of all phases taking transportation as well as relocation costs into consideration.

The automated system offers two alternatives for performing the optimization process.

1- Start a new optimization performing Step-1 followed by Step-2.

2- Load the results of a previous static optimization and use it to conduct dynamic optimization.

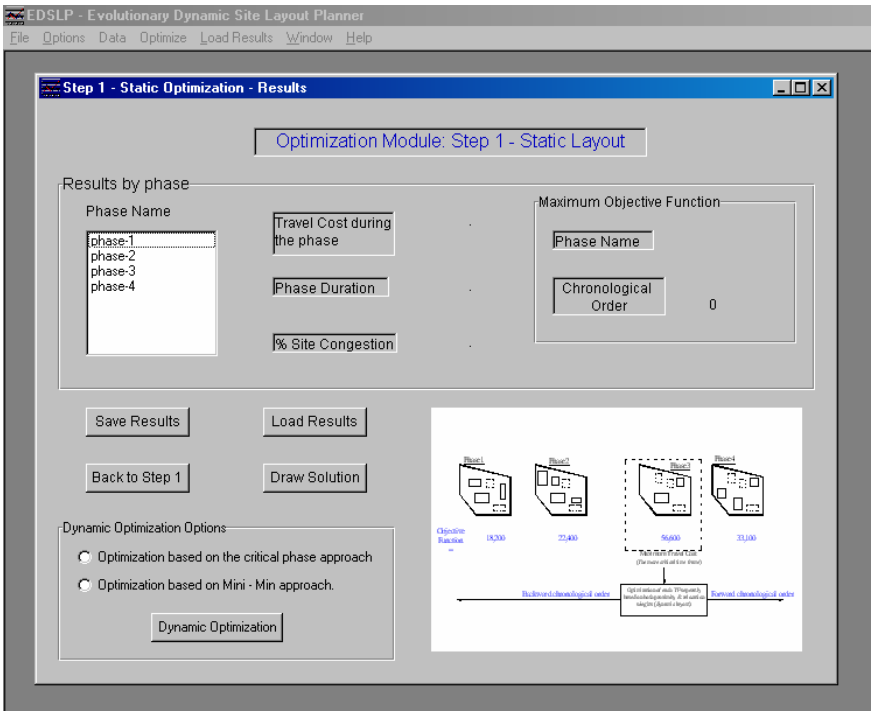


Figure 5-5 Static optimization results loading screen

5.3 SYSTEM OUTPUT

The system provides two distinct forms of output, namely graphical output and a cost summary. Graphical output is represented via automated assignment of temporary facilities in the AutoCAD™ environment (Figure 5-8). A summary for the total layout costs (transportation costs + relocation costs) can be presented and automated charting capabilities via MSExcel™ is available through the optimization results screen Figure 5-6.

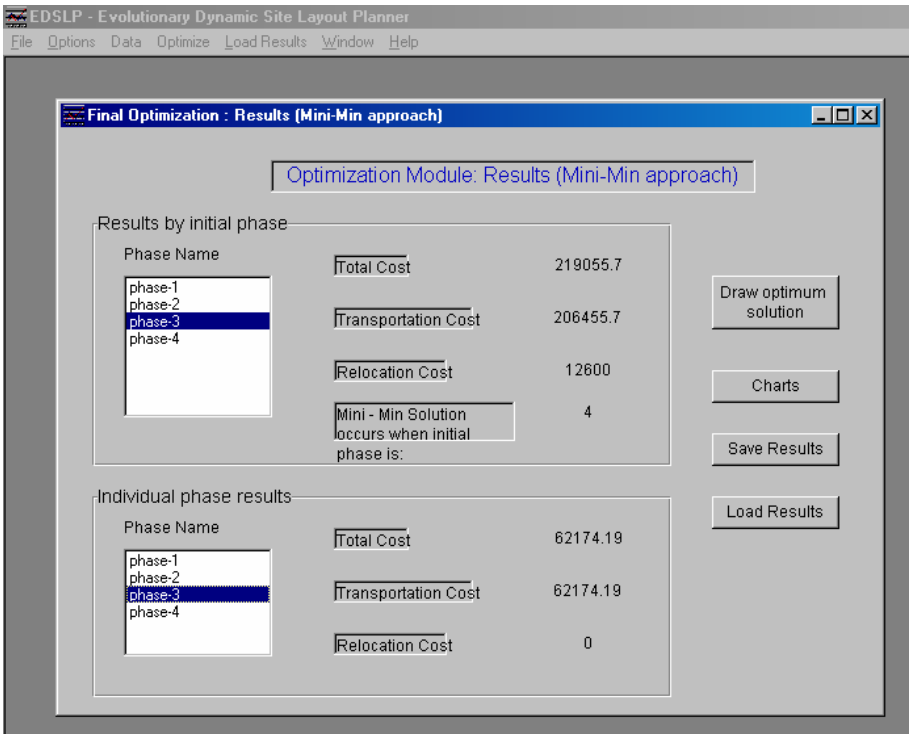


Figure 5-6 Dynamic optimization results screen (Mini-Min approach)

5.4 ILLUSTRATED EXAMPLE

The following example of a 4-phase project is used to demonstrate the system capabilities and evaluate the output. The illustrative project is comprised of a – 15,000 m^2 - residential compound of 4 villas, a swimming pool, in addition to the necessary infrastructure. The project is scheduled to be completed in 12 months. The first phase lasts for 2 months and involves the construction of the necessary infrastructure for the compound. The second phase involves the construction of the large, southern villas and lasts for 5 months. During the third phase of the project, construction operations mainly take place in the northern section of the site where the smaller villas are being constructed. This phase lasts for 3 months. The fourth phase of the project starts on the 11^{th} month of the project and involves the construction of a swimming pool in the center of the compound. The evolution of the site layout is shown in Figure 5-7.

5.4.1 Schedule & Temporary facility data

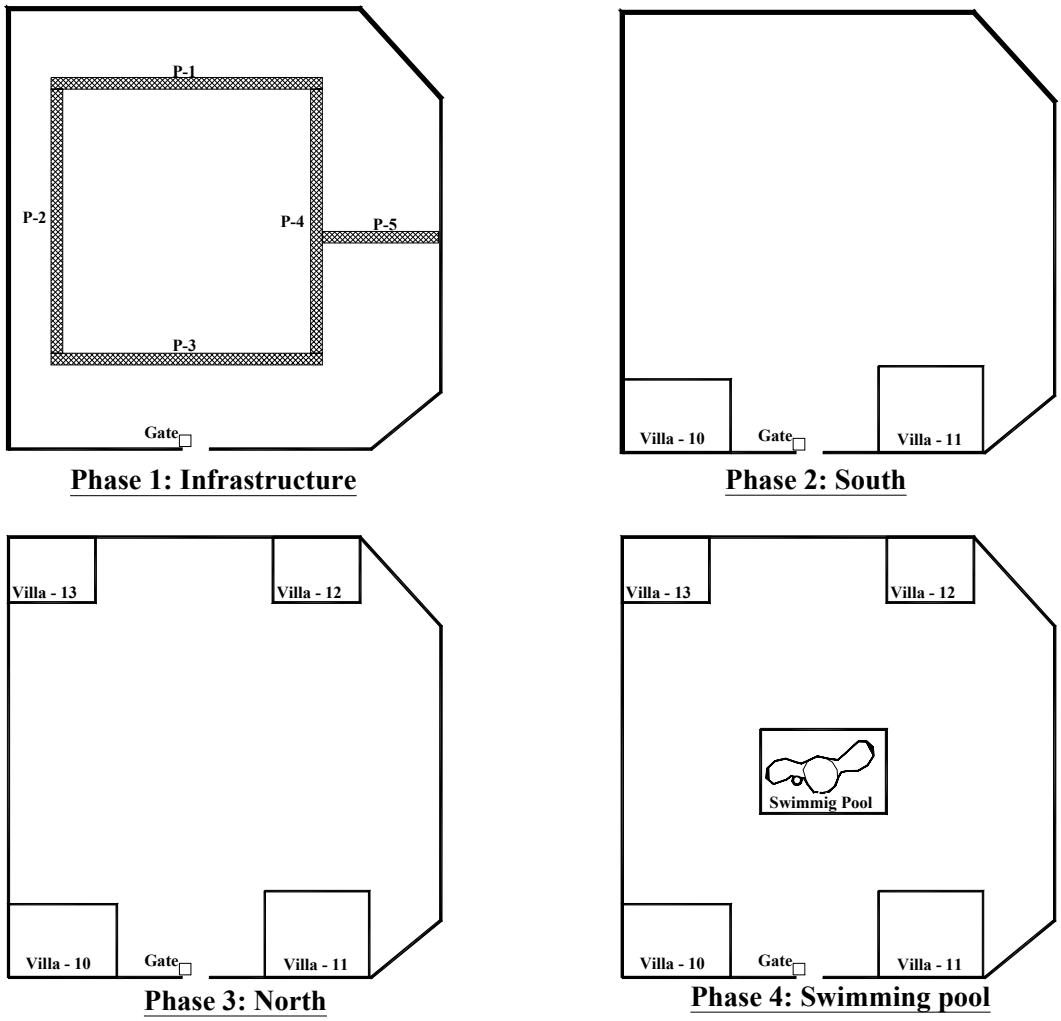
To sustain the required construction operations, 7 temporary facilities are required. Not all temporary facilities will be required throughout the project (Table 5-1). When infrastructure works are underway, only the caravans and the electromechanical warehouse are required. During phases 2 and 3, all temporary facilities are required. The

aggregate storage area can be dismantled during the last phase, when no major concrete works will be underway.

Table 5-1 Schedule and temporary facility data

Temporary Facility	Dimensions (m)	Phase1 Infrastructure	Phase2 South	Phase3 North	Phase4 Swimming pool
		2 months	*5 months*	*3 months*	*2 months*
Administrative Caravans	**10x8**				
Engineer's Caravans	**10x5**				
Steel Fabrication Yard	**14x14**				
Concrete Mixer	**10x10**				
Aggregate Storage	**12x6**				
Electromechanical Warehouse	**12x6**				
Wood Warehouse	**10x10**				

5.4.2 Site Layout Data:



Phase 1: Infrastructure **Phase 2: South**

Phase 3: North **Phase 4: Swimming pool**

Figure 5-7 Evolution of site layout throughout project phases

5.4.3 Facility cost data

1- Transportation cost data

The inter-facility cost matrices for facilities present during all project phases are illustrated in Table 5-2.

Inter-facility cost matrix – Phase 1

	Elec. Warehouse	Caravan 1	Caravan 2	P-1	P-2	P-3	P-4	P-5
Caravan 1	0							
Caravan 2	0	37						
P-1	81	0	37					
P-2	81	0	37	0				
P-3	81	0	37	0	0			
P-4	81	0	37	0	0	0		
P-5	81	0	37	0	0	0	0	
Gate	37	81	9	0	0	0	0	0

Inter-facility cost matrix – Phase 2

	Elec. Warehouse	Wood Warehouse	Steel fabrication yard	Agg. Storage	Caravan 1	Caravan 2	Mixer	Villa - 10	Villa - 11
Wood Warehouse	0								
Steel fabrication yard	0	0							
Agg. Storage	0	0	0						
Caravan 1	0	0	0	0					
Caravan 2	0	0	0	0	37				
Mixer	0	0	0	81	0	0			
Villa - 10	37	81	81	0	3	37	81		
Villa - 11	37	81	81	0	3	37	81	0	
Gate	37	37	9	37	81	9	0	0	0

Table 5-2 Inter-facility cost matrix for the four project phases

Inter-facility cost matrix – Phase 3

	Elec. Warehouse	Wood Warehouse	Steel fabrication yard	Agg. Storage	Caravan 1	Caravan 2	Mixer	Villa - 10	Villa - 11	Villa - 12	Villa - 13
Wood Warehouse	0										
Steel fabrication yard	0	0									
Agg. Storage	0	0	0								
Caravan 1	0	0	0	0							
Caravan 2	0	0	0	0	37						
Mixer	0	0	0	81	0	0					
Villa - 10	0	0	0	0	0	0	0				
Villa - 11	0	0	0	0	0	0	0	0			
Villa - 12	37	81	81	0	3	37	81	0	0		
Villa - 13	37	81	81	0	3	37	81	0	0	0	
Gate	37	37	9	37	81	9	0	0	0	0	0

Inter-facility cost matrix – Phase 4

	Elec. Warehouse	Wood Warehouse	Steel fabrication yard	Caravan 1	Caravan 2	Mixer	Villa - 10	Villa - 11	Villa - 12	Villa - 13	Swimming Pool
Wood Warehouse	0										
Steel fabrication yard	0	0									
Caravan 1	0	0	0								
Caravan 2	0	0	0	37							
Mixer	0	0	0	0	0						
Villa - 10	0	0	0	0	0	0					
Villa - 11	0	0	0	0	0	0	0				
Villa - 12	0	0	0	0	0	0	0	0			
Villa - 13	0	0	0	0	0	0	0	0	0		
Swimming Pool	81	81	81	3	37	37	0	0	0	0	
Gate	37	37	9	81	9	0	0	0	0	0	0

Table 5-2 Inter-facility cost matrix for the four project phases (continued)

2- Relocation cost data

Table 5-3 Facility Relocation Cost Data

Temporary Facility	Relocation Cost
Administrative Caravans	5000
Engineer's Caravans	5000
Steel Fabrication Yard	1000
Concrete Mixer	300
Aggregate Storage	1000
Electromechanical Warehouse	2500
Wood Warehouse	2000

5.5 OPTIMIZATION RESULTS

The output capabilities of the automated system are exemplified through the optimization results of the illustrated example presented in the preceding section. The system enables optimization results to be presented in various forms.

5.5.1 CAD presentation

The most obvious form of output is that in which data was input, CAD drawings. The automated system is capable of opening project drawings and placing temporary facilities in their optimal positions.

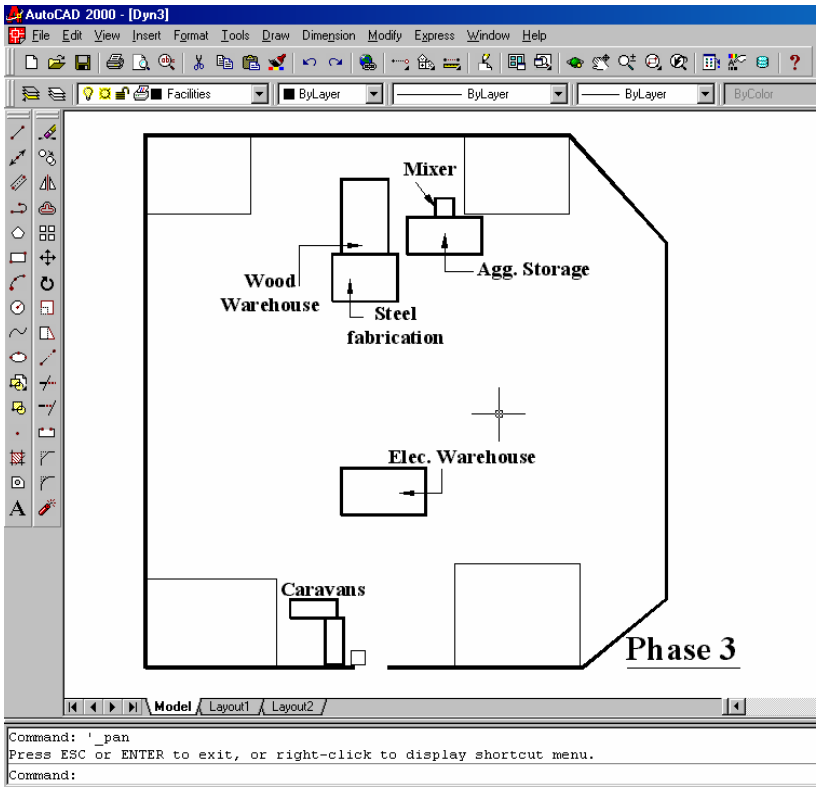


Figure 5-8 System automated drawing capabilities

5.5.2 Comparative graphical cost presentation

The system provides automated charting capabilities to be used for comparative analysis of cost related data. Table 5-4 compares the total layout costs associated with each initial phase. It is evident that choosing phase-4 as the initial phase yields the Minimum-Minimum solution.

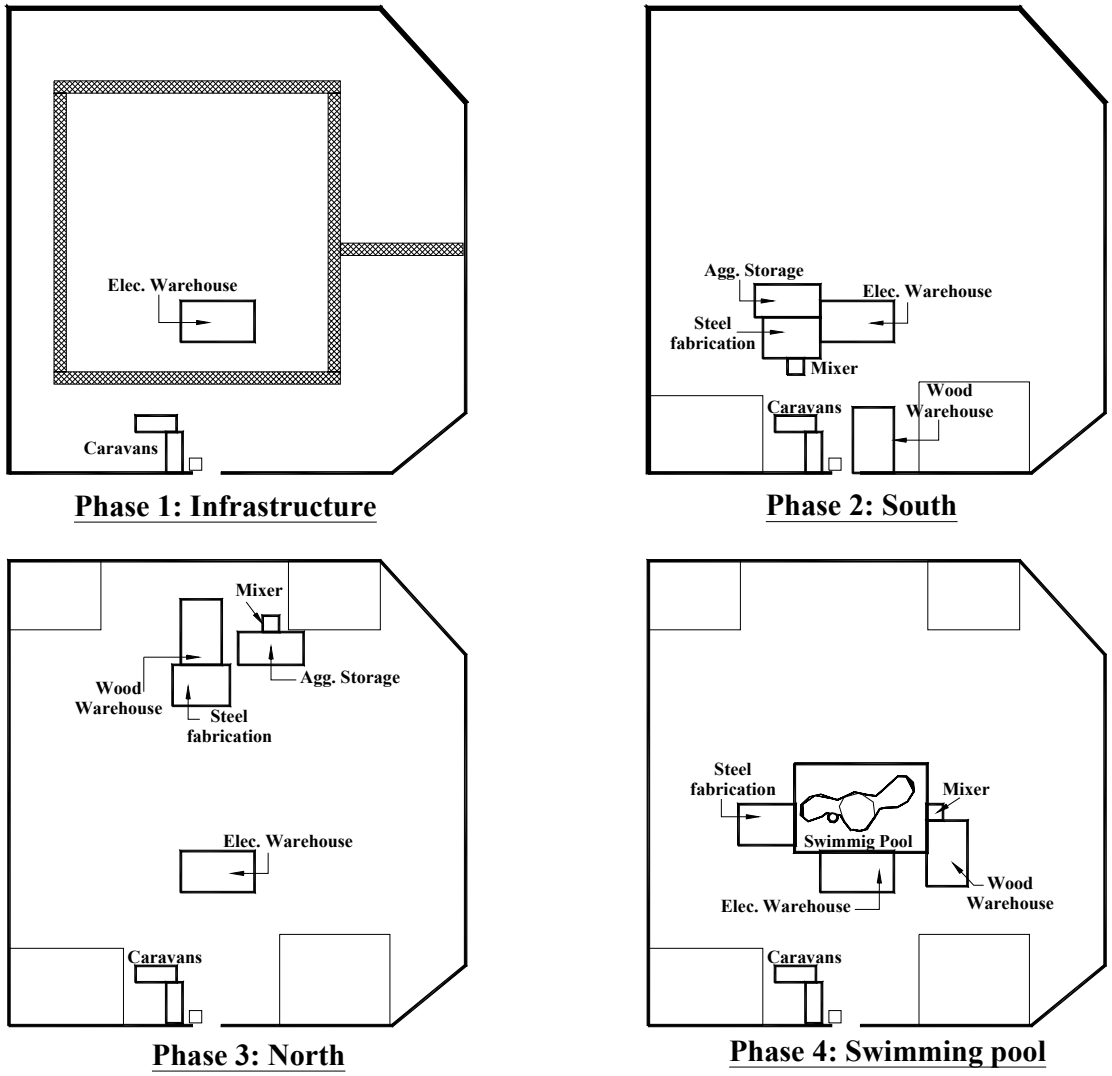


Figure 5-9 Automated system generated layouts – Mini-Min approach

Table 5-4 Layout cost data – Mini-Min approach

Initial Phase	Summary of layout costs

Phase-1

Phase	Transportation Cost	Relocation Costs	Total Costs
Phase-1	25993	0	25993
Phase-2	69964	2500	72464
Phase-3	63690	4300	67990
Phase-4	12367	8000	20367
Total Costs	172013	14800	186813

Phase-2

Phase	Transportation Cost	Relocation Costs	Total Costs
Phase-1	31043	2500	33543
Phase-2	72012	0	72012
Phase-3	72178	3300	75478
Phase-4	10999	8300	19299
Total Costs	186233	14100	200333

Phase-3

Phase	Transportation Cost	Relocation Costs	Total Costs
Phase-1	29396	2500	31896
Phase-2	98012	6800	104812
Phase-3	62174	0	62174
Phase-4	16874	3300	20174
Total Costs	206456	12600	219056

Phase-4

Phase	Transportation Cost	Relocation Costs	Total Costs
Phase-1	28678	0	28678
Phase-2	73835	4300	78135
Phase-3	62578	4300	66878
Phase-4	11839	0	11839
Total Costs	176929	8600	185529

6 SYSTEM VALIDATION & CASE STUDY

In this chapter the performance of the automated system is validated via a construction project with 24,000 m^2 site area. Based on the results, a comparison between the existing layouts and the layouts created by the system is conducted and comparisons are drawn

6.1 CASE STUDY

The selected project is a major swimming pool complex with several auxiliary buildings. The site is part of the new Heliopolois club in El-Shorouk city, Egypt. The contract for the complex was awarded for 14 million LE . Due to the immense size of the club's ground and facilities, it was divided into three main packages, the swimming pool complex, the social building, and the sports grounds. Each was awarded to a separate contractor. Each of the contractors occupies a portion of the club's site where his work is executed. All contractors independently sustain their construction sites and no sharing of facilities occurs between them.

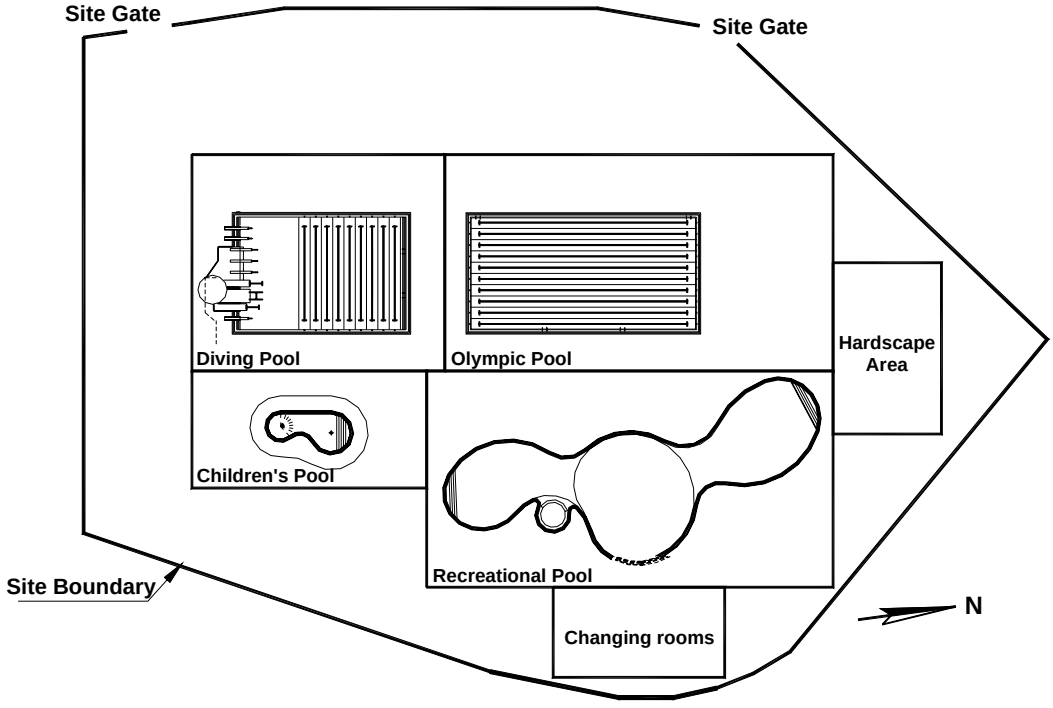


Figure 6-1 Layout of fixed facilities on the construction site

The case study focuses only on the swimming pool complex and considers it a completely independent site. The site layout showing all fixed facilities present is

shown in Figure 6-1.

During the project planning phases, the site layout was created by the project management group. Extensive effort was placed in the layout development which was thereafter approved by the engineer. The layout was developed solely using past experience and no formal approach was exploited.

During a site visit the following information was collected:

1- The project schedule.

2- The actual layout of the temporary facilities on site.

3- Inter-facility closeness relationships.

4- Facility relocation costs.

6.1.1 Project Schedule data

After thorough analysis of the project schedule and method statement, three distinct phases could be identified as shown in Table 6-1.

Table 6-1 Project phases with a brief description of the main construction operations

Phase	Duration (months)	Available Site Space (m^2)	Description
Phase 1	11	14632 m^2	Work is started in the Olympic and diving pools. Only the north section of the recreational pool has commenced to allow the concrete pump to reach the east walls of the Olympic pool
Phase 2	7	11676 m^2	Work is started in the changing rooms building and the south section of the recreational pool. Work in the children's pool and the hardscape area has not yet begun.
Phase 3	6	9408 m^2	Work in all 6 permanent structures of the project is underway.

6.1.2 Permanent Facilities data

The swimming pool complex is comprised of six main permanent structures as well as the site gate. An underground basement occupies a large portion of the Olympic and diving pools. Table 6-2 illustrates permanent facilities and their dimensions.

Table 6-2 Listing and approximate dimensions of permanent facilities

Permanent facility	Approx. Dimension (m)
Diving Pool	56 x 48
Olympic Pool	86 x 48
Children's Pool	52 x 26
Recreational Pool	90 x 48
Changing Rooms	38 x 20
Hardscape Area	38 x 24
Site Gate	3 x 3

6.1.3 Site Obstacles

An access road covering the site was considered as an obstacle. During the first phase, an area south of the recreational pool was reserved to allow the concrete pump to pour the east wall of the olympic pool. The arrangement of permanent facilities and site obstacles within the site boundaries during the three project phases is shown in Figure 6-2 At maximum site congestion (third phase), the permanent facilities and site obstacles occupy nearly 15000 m^2 of the site. The remaining 9000 m^2 is left for assigning the temporary facilities.

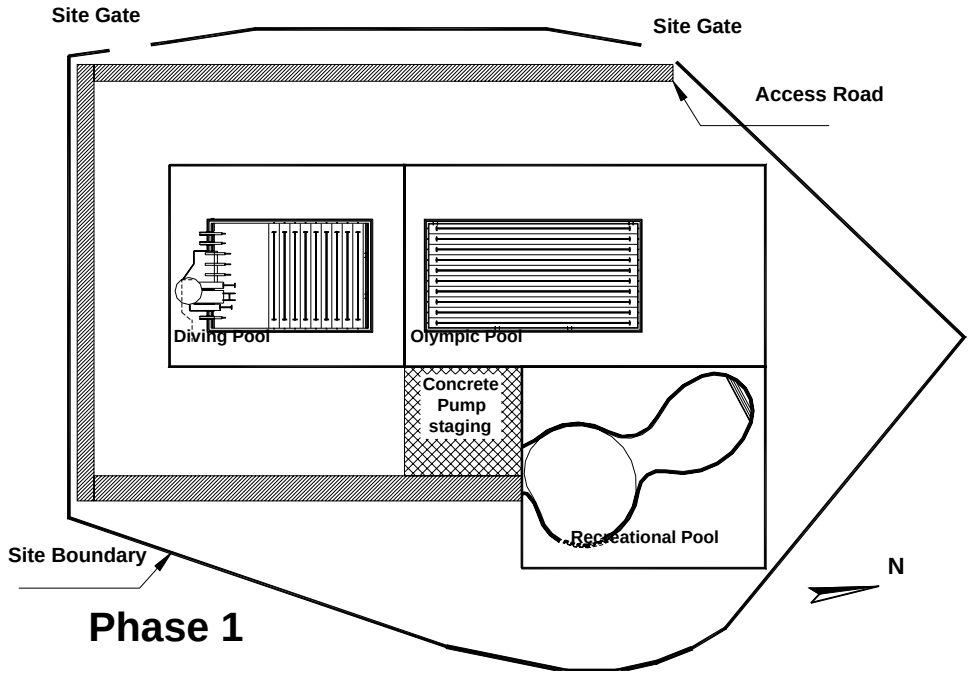
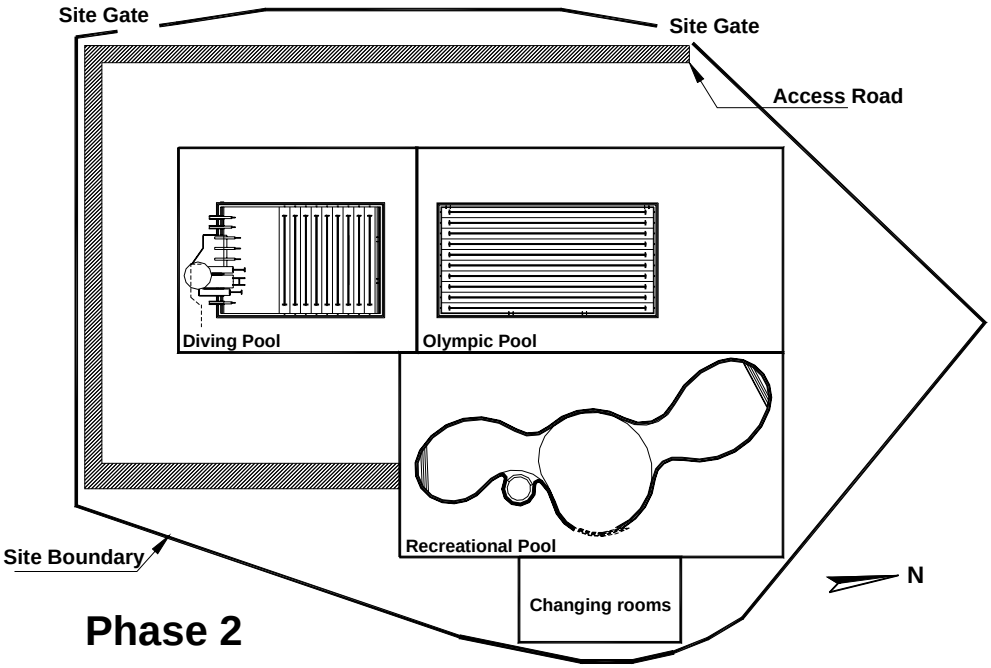


Figure 6-2 Arrangement of permanent facilities & obstacles on the swimming pool complex during different project phases



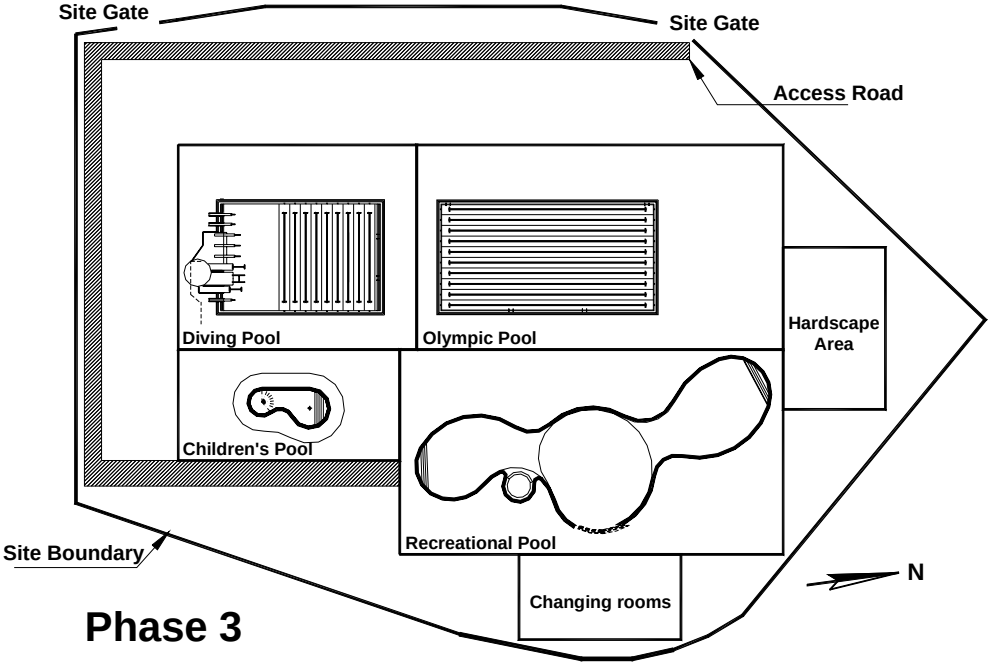


Figure 6-2 Arrangement of permanent facilities & obstacles on the swimming pool complex during different project phases (continued)

6.1.4 Temporary Facilities

The temporary facilities found on the site can be divided into four main categories:

1- Storage Areas: General storage Area, Steel storage

2- Staff Facilities: Caravans, Toilets, Prayer area, Parking

3- Supporting Facilities: Water Tanks, Generators

4- Fabrication Areas: Steel fabrication yard

The temporary facilities needed to sustain the construction requirements, their dimensions and relocation costs are shown in Table 6-3

Table 6-3 Dimensions and relocation costs of temporary facilities

Temporary Facility	Dimension (m)	Relocation Cost (LE)
Storage Area	42 x 20	2400
2 Administrative Caravans	10 x 3	1150
2 Engineer's Caravans	10 x 3	1150
Parking Area	12 x 4	150
Toilets	10 x 3	800
Steel fabrication yard	10 x 18	450
Steel storage	10 x 12	250
Generators	2 x 2	150

6.1.5 Proximity Matrix

An inter-facility proximity matrix was provided by the project manager during a short interview. This matrix was transformed into its quantitative equivalents before input to the system. (Table 3-4).

	Storage Area	Steel fabrication yard	Steel Storage	Car Parking	Adm Caravan 1	Adm Caravan 2	Eng Caravan 1	Eng Caravan 2	Diving Pool	Olympic Pool	Children's Pool	Recreational Pool	Changing rooms	HardScape Area
Steel fabrication yard	U													
Steel Storage	U	A												
Car Parking	U	X	X											
Adm Caravan 1	U	X	X	U										
Adm Caravan 2	U	X	X	I	A									
Eng Caravan 1	U	X	X	I	E	E								
Eng Caravan 2	U	X	X	I	E	E	A							
Diving Pool	A	I	E	E	X	X	I	I						
Olympic Pool	A	I	E	E	X	X	I	I	N/A					
Children's Pool	E	U	I	I	X	X	I	I	N/A	N/A				
Recreational Pool	E	U	I	I	X	X	I	I	N/A	N/A	N/A			
Changing rooms	E	U	I	I	X	X	I	I	N/A	N/A	N/A	N/A		
HardScape Area	I	U	I	I	X	X	I	I	N/A	N/A	N/A	N/A	N/A	
Gate	I	I	X	A	A	A	E	E	N/A	N/A	N/A	N/A	N/A	N/A

6.2 AUTOMATED SYSTEM OUTPUT

6.2.1 Step1: Static Layout

The system performed the GA-based optimization of the three project phases in a total of 92 minutes running on a Pentium-3 800 MHz processor. A termination condition of Δ = 5% was used. Table 6-4 summarizes the static optimization results. The following genetic parameters were used:

$P_{mutation} = 0.05$
$P_{crossover} = 0.7$
Population size = 250

Table 6-4 Summary of the GA-based optimization process for the three project phases

	Run-Time (minutes)	Number of GA generations	Optimum Transportation costs (LE)
Phase-1	43	1,664	13,141
Phase-2	22	558	8,142
Phase-3	27	1,168	8,726

The generated layouts based solely on inter-facility transportation costs are shown in Figure . It is noted that till this step the system deals with each layout as if it were a totally independent optimization problem. It is the dynamic optimization process that integrates the relocation costs of facilities from one phase to the next, creating the required chronological continuity between project phases.

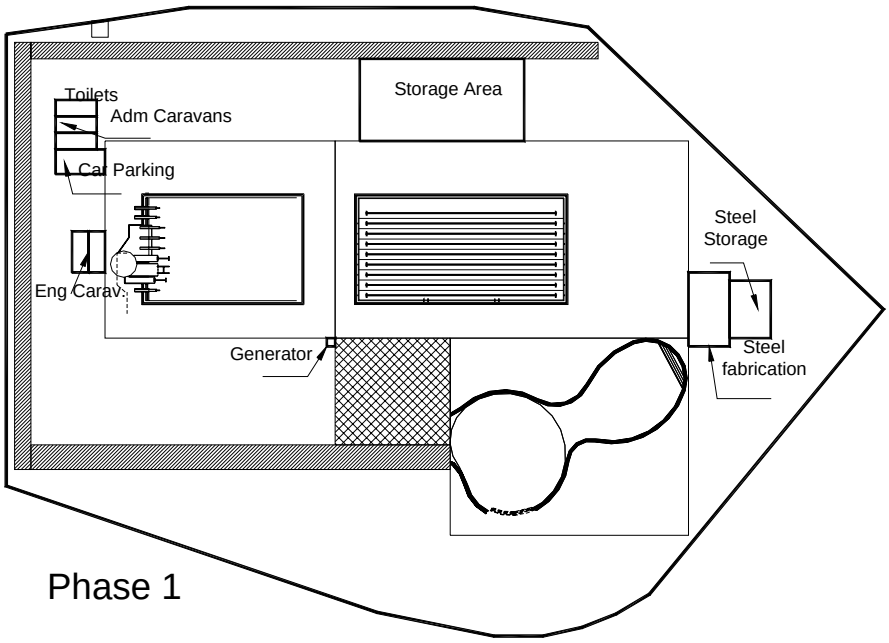


Figure 6-3 Automated system assignment of temporary facilities (Static Layout)

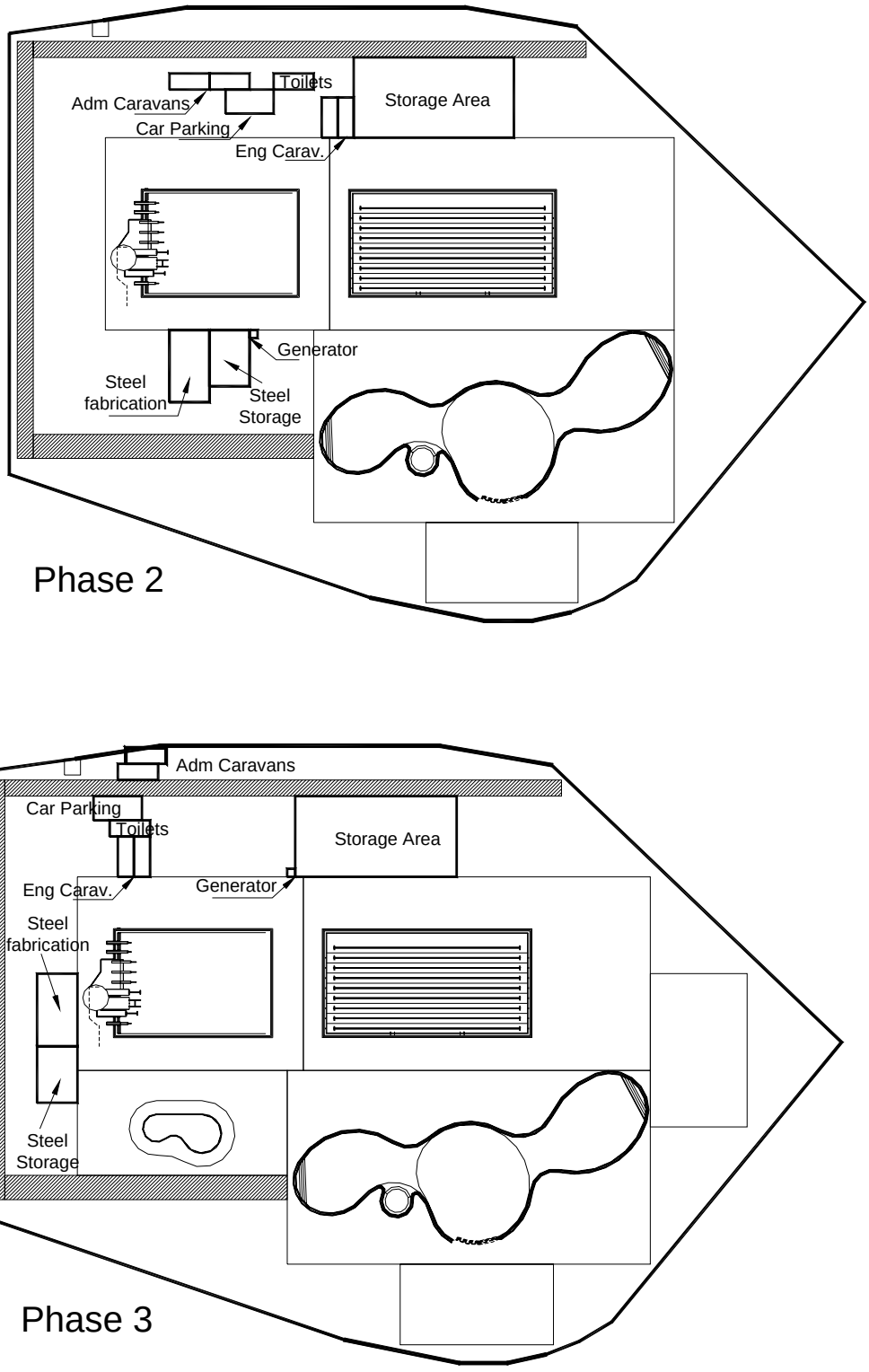


Figure 6-3 Automated system assignment of temporary facilities -Static Layout (continued)

6.2.2 Step2: Dynamic Layout

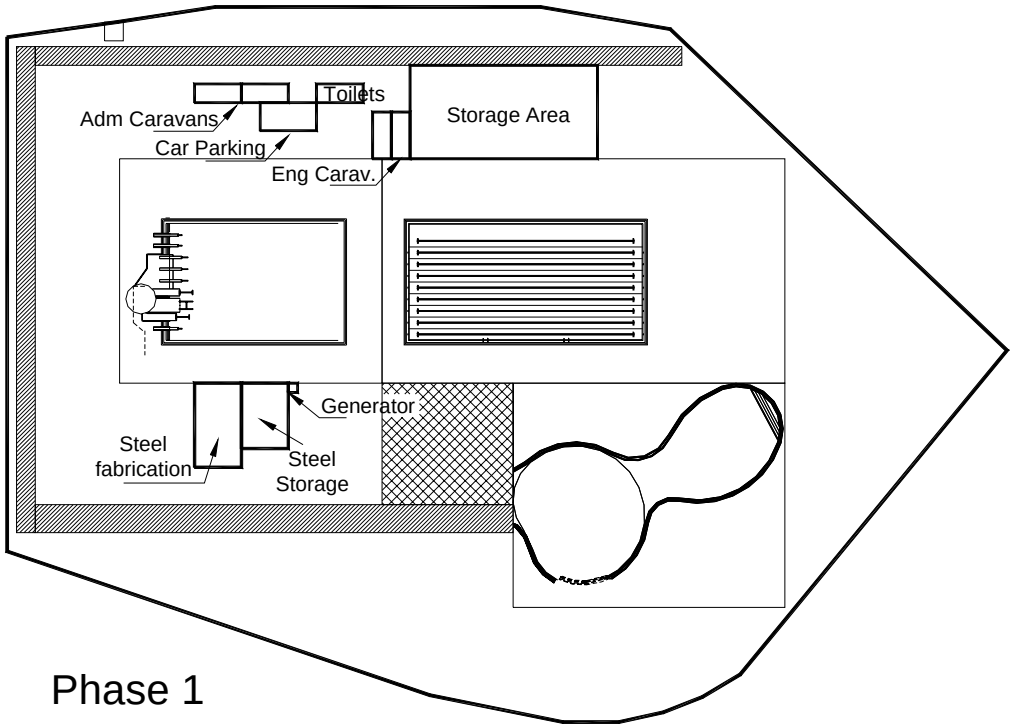
The second step is performing the optimization process taking into consideration the relocation costs of facilities from one layout to the next. As shown in Table

6-5Table 6-4, the maximum transportation costs occur during Phase-1 of the project. Total transportation and relocation costs are dependant on the choice of the initial phase.

The results, shown in Table 6-5, demonstrate the transportation and relocation costs for all project phases when each of the project phases is taken is the initial phase. When Phase-1 is taken as the initial phase for dynamic layout planning, the optimization process yields the highest values for both transportation and relocation costs. Adopting the critical phase approach, Phase-1 would be taken as the initial, *critical* phase and layouts would be constructed in forward chronological order. Adopting the Mini-Min approach will consider all possible combinations for choosing the initial phase and then choose the combination that yields the lowest total transportation and relocation costs. In this case, choosing Phase-2 as the initial phase.

Table 6-5 Summary of layout costs after dynamic optimization (Mini-Min approach)

Initial Phase	Summary of layout costs			
	Phase	**Transportation Cost (LE)**	**Relocation Costs (LE)**	**Total Costs (LE)**
Phase-1	**Phase-1**	13,141	0	13,141
	Phase-2	9,315	2,550	11,865
	Phase-3	8,875	3,100	11,975
	Total Costs	**31,331**	**5,650**	**36,981**
	Phase	**Transportation Cost (LE)**	**Relocation Costs (LE)**	**Total Costs (LE)**
Phase-2	**Phase-1**	11,856	0	11,856
	Phase-2	8,142	0	8,142
	Phase-3	8,871	850	9,721
	Total Costs	**28,870**	**850**	**29,720**
	Phase	**Transportation Cost (LE)**	**Relocation Costs (LE)**	**Total Costs (LE)**
Phase-3	**Phase-1**	11,770	700	12,470
	Phase-2	9,015	0	9,015
	Phase-3	8,727	0	8,727
	Total Costs	**29,512**	**700**	**30,212**



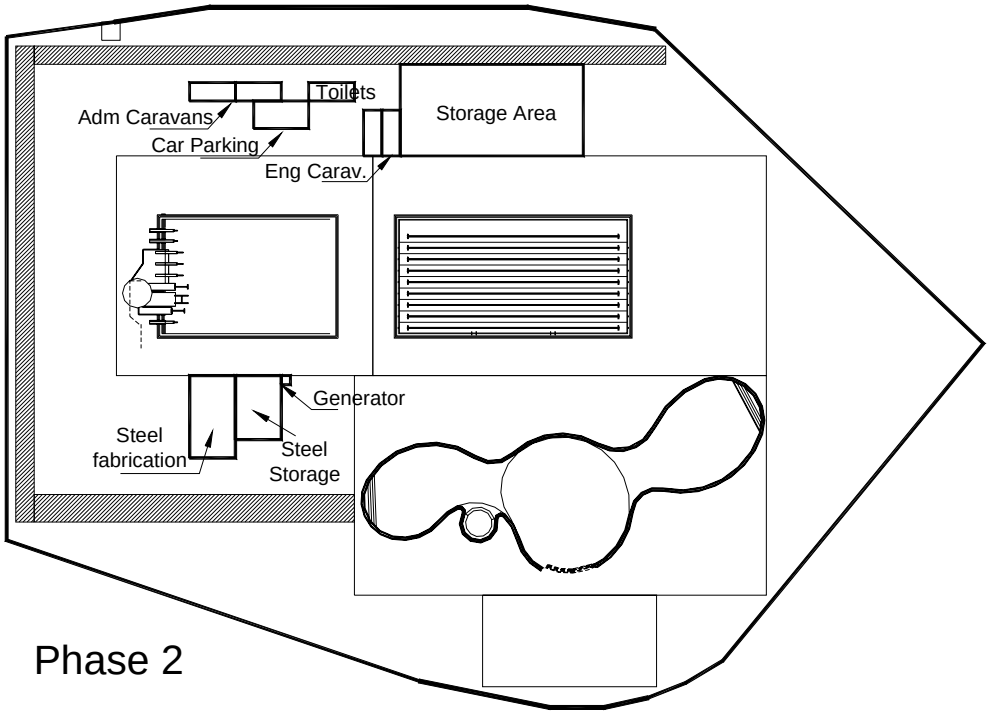


Figure 6-4 System assignment of temporary facilities (Dynamic Layout, Mini-Min approach)

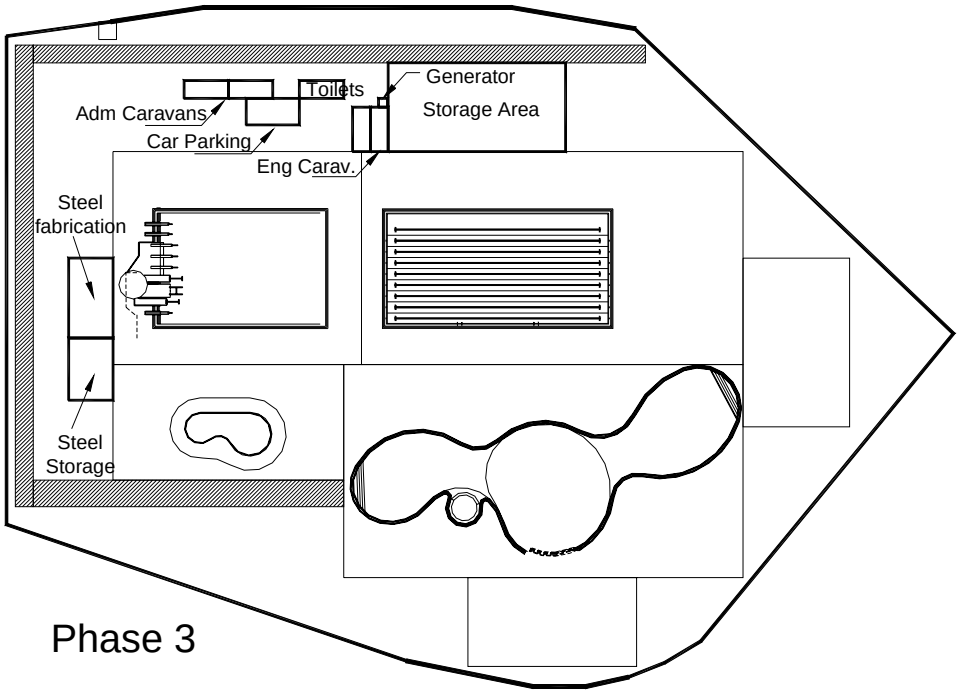


Figure 6-4 System assignment of temporary facilities (Dynamic Layout, Mini-Min approach), continued

The Mini-Min solution was attained when taking Phase-2 as the initial phase, yielding a total layout cost of 29,720 LE. The three layouts are depicted in Figure 6-4. It is noted that no relocation costs occur from Phase-2 to Phase-1, that is the layouts are identical during the first 18 months of the project. The third phase did undergo the relocation of the steel fabrication yards, the steel storage area and the generator.

6.3 ACTUAL SITE LAYOUTS

During a brief site interview, the project manager made clear the following facts pertaining to the site layout:

1. Site constraints forced the schedule to proceed in a certain order. The south section of the recreational pool was postponed until the east wall of the olympic pool was poured. The concrete pump's boom could not reach the east wall otherwise (Figure 6-2).
2. The engineer's caravans were placed in place of the hardscape area. Work in the hardscape area was scheduled to begin on the 18th month of the project. The caravans were to be relocated during that time.
3. During the 15th month of the project the engineer issued a change order. The west edge of the olympic and diving pools were shifted 6m to the west. This forced the

contractor to move the administrative caravans and car parking so as not to obstruct this shift. The storage area decreased in size due to this shift.

4. The contractor decided to relocate the engineer's caravans on the 15th instead of the scheduled 18th month date so as to minimize site disruption caused by several facility relocations.

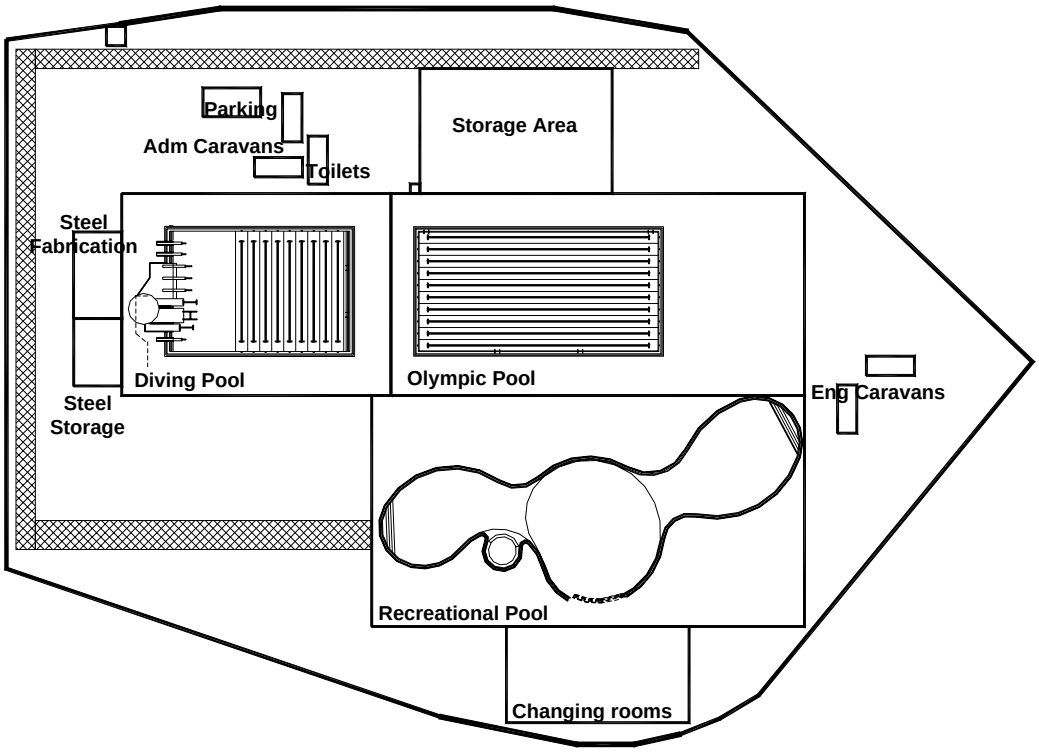


Figure 6-5 Actual site layout, first 15 months

Actual layouts adopted during the project differed to some extent from the system generated layouts: The site layout plan was intended to undergo only one change. The engineering caravans were to be moved during the last six months of the project to make room for the hardscape area. This change did occur, but ahead of schedule. During the 15th month, the owner issued a change order increasing the premises of the Olympic and diving pools by six meters to the west. The contractor was forced to relocate the 2 administrative caravans and the car parking. The contractor was reimbursed for the relocation. At the same time, and so as not to disrupt the layout any further, it was decided to relocate the engineer's caravans and make room for the hardscape area which was to undergo construction in only three months time.

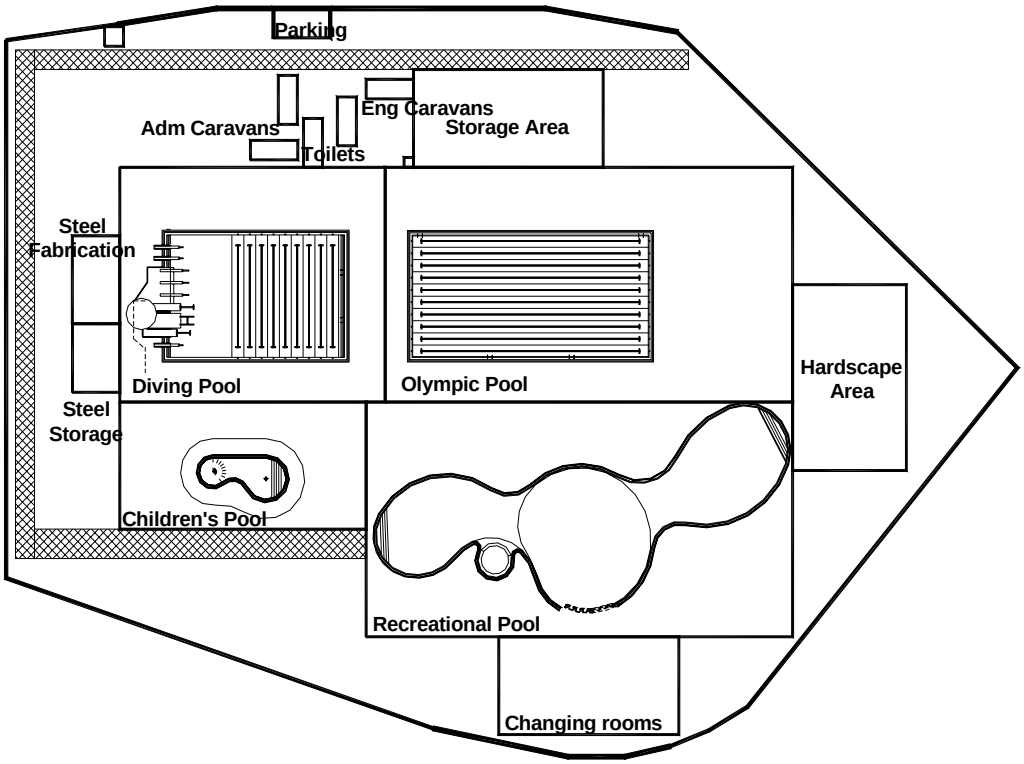


Figure 6-6 Actual site layout, last 9 months

6.4 COMPARATIVE ANALYSIS

The automated system dealt with the three main phases but yielded only two distinct layouts. One layout spanned the first and second phases, while the second layout spanned the last phase. A brief comparison between the assignment of temporary facilities in the actual and the system generated layouts is shown in Table 6-6. Total layout costs for the actual and the system generated layouts is shown in Table 6-7.

Table 6-6 Comparison between actual and system generated layouts

Temporary Facility	Comment
Storage Area	Assignment of the storage area was identical in both layouts. This consistency is mainly due to the large size of the facility and thus the unavailability of various site locations to be placed in.

Steel fabrication yard & Steel storage area	In the actual layout, no relocation occurred. In the system's layout, they were assigned in place of the children's pool during the first two phases as no construction had yet begun. During the last phase they had to be relocated. They were relocated south of the diving pool, which was the same position the contractor had assigned them in. This indicates a partial consistency between the generated layout and the actual layout.
Engineer's Caravans	In the actual layout, the caravans underwent relocation from the hardscape area to an area west of the diving pool. In the generated layout, no relocation occurred and the caravans were assigned west of the diving pool throughout the project. Although during the last phase the caravans were placed west of the diving pool in both layouts, their exact assignment did differ by approximately 4 meters.
Administrative Caravans	Both layouts assigned the administrative caravans west of the diving pools. Their exact assignment did differ by approximately 7 meters.
Parking Area	In the actual layout the car parking underwent relocation upon issue of the change order. In the generated layout, no relocation occurred. Both layouts assigned the parking west of the diving pool although their exact assignments did differ by 1 meter in the first two phases and 10 meters in the last phase.

Table 6-7 Comparative layout costs between actual and automated system layout

	Actual Layout	**Generated Layout**
Transportation costs (LE)	35,600	28,870
Relocation costs (LE)	4,750	850
Total Layout Cost (LE)	**40,350**	**29,720**

7 SUMMARY & CONCLUSIONS

7.1 SUMMARY

This research investigates the use of genetic algorithms in construction site layout planning. Although genetic algorithms have been previously utilized by other researchers in solving the site layout planning problem, this research aims to create an integrated CAD-based GA automated computer system for site layout planning. The system utilizes genetic algorithms (GA's), as function optimizers, in determining the temporary facility location according to the graphical information depicted in a CAD environment.

Due to the special nature of the problem at hand, a problem oriented GA procedure was developed in order to suit site layout planning problem. The developed GA uses steady state generation with a modified mutation operator. Due to the evident graphical nature of the problem, the GA was integrated with a CAD platform. The CAD environment is utilized in space detection of the site layout and in the satisfaction of geometrical constraints dictated by the facility assignment problem.

The system addresses the changing nature of construction sites via performing dynamic site layout planning. Two dynamic optimization approaches have been suggested to overcome the shortcomings found in the traditional dynamic layout techniques. The integrated system has been implemented via MS VisualBasic™ 6.0. The CAD interface has been made possible through the programmable features of AutoCAD™ in the VBA environment. The automated system EDSLP (*Evolutionary Dynamic Site Layout Planning System*) incorporates various data input facilities, a GA-based optimization engine and a CAD output facility.

In order to validate the performance of the system, it was tested on an actual 24,000 m^2 construction site. The system produced a site layout that accomplished nearly a 25% saving in total layout cost compared to the layout actually adopted.

7.2 CONCLUSIONS

1- Genetic algorithms can be satisfactorily used to solve the site layout problem: When tested with an actual case study the EDSLP system yielded highly optimum solutions with nearly 25% savings in total layout cost. Run times are feasible and reasonable compared to other genetic algorithms taking into account the iterative dynamic optimization approach used.

2- The integration between CAD platforms and genetic algorithms was successful in performing the site layout optimization process: Overall, the integrated system managed to benefit from the intricate search and optimization abilities of genetic algorithms and at the same time utilize the powerful graphical capabilities of CAD systems. Optimization results were very satisfactory as previously mentioned. Geometrical constraints dictated by the site geometry were strictly followed by the CAD platform as no overlap or out-of-boundary assignment of temporary facilities occurred.

3- The two suggested approaches for performing the dynamic layout process overcome the shortcomings found in previous models: Both the Critical Phase approach and the Mini-Min approach for dynamic layout aim to mitigate costs of assignment of facilities in positions that may seem favorable in early phases, but could turn out very costly in phases to come. Traditional chronological dynamic layout is proven to yield non-optimal solutions throughout the project phases.

4- Assigning facilities based solely on transportation & relocation costs neglects other secondary objectives: This was found when comparing the system generated layout with that created by the project manager. Although the system's layout scored higher than the adopted layout, the project manager considered the layout to create some minor local site congestions. Thus it can be concluded that other secondary objectives should be taken into consideration when performing site layout planning. These objectives include but are not limited to minimizing local site congestion and promoting safe working environments.

7.3 RECOMMENDATIONS FOR FURTHER RESEARCH

1- Formulate a comprehensive objective function that accounts for secondary, intangible objectives in addition to the main objective: Assigning facilities based solely on transportation & relocation costs neglects other secondary objectives as mentioned. Thus, a comprehensive multi-objective objective function should be formulated that takes all influencing factors into consideration according to their relative importance.

2- Provide the system with a knowledge-based facility identification module: This module will estimate the number of required temporary facilities and the dimension of each temporary facility. This estimation will be based on project related data such as: the type of construction activities involved, the number of personnel present on site, the amounts of material to be stored, required equipment, etc…

3- Enhance the system to be able to deal with actual travel distances between facilities instead of rectilinear distances: The present system utilizes rectilinear distances between facilities. These distances may not necessarily represent the realistic distances to travel inside the site. Especially in congested sites, maneuvering around facilities may be required during travel. Using CAD features, actual travel distances around facilities can be correctly estimated.

4- Enhance the system to be able to deal with irregular facility shapes instead of rectangular shapes: The present system is able to deal with irregular site boundaries but can only identify rectangular facility shapes. Although most temporary construction facilities tend to have rectangular shapes, permanent facilities may assume any irregular shape. Enhancing the system to be able to recognize irregular facilities will further increase its practicality for use.

5- Develop a system is capable of dealing with three dimensional spatial aspects. (3-D site layout planning): The present system performs site layout planning at the 2-D level. Some projects (eg. High-rise buildings built on congested sites) require temporary support facilities to be placed inside the building whilst construction. Enhancing the system to deal with the third dimension will make it more suitable for these types of projects, further increasing its practicality for use.

8 REFERENCES

Al-Tabtabi, H. and Alex, A. (1998). “An evolutionary approach to the capital budgeting of construction projects.” Cost Engineering, AACE, 40(10), 28-34

Chan, W.T., Chaua, D.K., Kannan, G. (1996). “Construction resource scheduling with genetic algorithms” Journal of Construction Engineering & Management, ASCE, 122(2), 125-132

Cheng, M.Y. and O’Connor, J.T. (1996). “ArcSite: Enhanced GIS for construction site layout” Journal of Construction Engineering & Management, ASCE, 122(4), 329-336

Cheung, S.-O., Thomas Kin-Lun Tong, Tam, C.-M. (2001) “Site pre-cast yard layout arrangement through genetic algorithms” Automation in Construction , 11(1), 35-46.

Elbeltagi, E. (1999) "Construction Site Planning" Ph.D. dissertation. Faculty of Engineering, El-Mansoura University, El-Mansoura, Egypt.

Elbeltagi, E. and Hegazy, T. (2001) “A hybrid AI-Based system for site layout planning in construction” Computer-Aided Civil and Infrastructure Engineering, Blackwell Publishers ,16(2), 79-93

Goldberg (1989), “Genetic Algorithms in Search, Optimization and machine learning” , Addison-Wesley Publishing

Hegazy, T. and Elbeltagi, E. (1999). “EvoSite: Evolution based model for site layout planning” Journal of Computing in Civil Engineering, ASCE, 13(3), 198-206

Hegazy, T. and Elbeltagi, E. (2000). “Simplified spreadsheet solutions: A model for site layout planning” Cost Engineering, AACE, 42(1), 24-30

Heng Li and Love, P.E. (1998). “Site level facilities using genetic algorithms” Journal of Computing in Civil Engineering, ASCE, 12(4), 227-231

Mahoney, J.J. and Tatum, C.B. (1994). "Construction site applications of CAD" Journal of Construction Engineering & Management, ASCE, 120(3), 617-631

Markhomihelakis, A. (1997). "Site layout planning and its relevance to safety

management and quality assurance". Msc dissertation. University of Manchester Institute of Science & Technology.

Rosenblant, M.J. (1986). “The dynamics of plant layout” Management Science, 32(1), 76-86

Schnecke, V. and Vornberger, O. (1997) “Hybrid genetic algorithms for constrained placement problems” IEEE transactions on evolutionary computation, 1(4), 266-277.

Taha (1971), “Operations Research, an Introduction”, Macmillan Publishing Company

Tam, C.M. ,Tong, T.K. and Chan, W.K. (2001). “Genetic algorithm for optimizing supply locations around tower crane” Journal of Construction Engineering & Management, ASCE, 127(4), 315-321

Tam, C.M. ,Tong, T.K., Lueng, A. and Chiu, G. (2002) "Site layout planning using nonstructural fuzzy decision support system" Journal of Construction Engineering & Management, ASCE, 128(3), 220-231.

Tommelien, I.D., Levit, R.E., and Hayes-Roth, B. (1992a). “Site-layout modeling: how can artificial intelligence help?” Journal of Construction Engineering & Management, ASCE, 118(3), 594-611

Tommelien, I.D., Levit, R.E., and Hayes-Roth, B. (1992b). “SitePlan model for site layout” Journal of Construction Engineering & Management, ASCE, 118(4), 749-766

Tommelien, I.D. and Zouein, P.P. (1993). “Interactive dynamic layout planning” Journal of Construction Engineering & Management, ASCE, 119(2), 266-287

Whitley, D. (1993). “A genetic algorithm tutorial” Tech. Rep. CS-93-103, Colorado State University, Fort Collins, Colorado.

Yeh, I-Cheng “Construction-site layout using annealed neural network” Journal of Computing in Civil Engineering, ASCE, 9(3), 201-208

Zouein, P.P. and Tommelien, I.D. (2001). “Improvement algorithm for limited space scheduling” Journal of Construction Engineering & Management, ASCE, 127(2), 116-

124

Zouein, P.P. and Tommelien, I.D. (1999). “Dynamic layout planning using a hybrid incremental solution method” Journal of Construction Engineering & Management, ASCE, 125(6), 400-408

Zouein, P.P., Harmanani, H. and Hajar, A. (2002)."A genetic algorithm for solving the site layout problem with unequal size and constrained facilities" Journal of Computing in Civil Engineering, ASCE, 16(2), 143-151

9 APPENDICES

APPENDIX A: OPTIMIZATION CODE

StaticOpt module

```
Const Maxpool = 1000
Const Maxpop = 1000
Const Maxstring = 10 'Also maximum number of facilities
Const Maxpoints = 10000 'Maximum number of grid squares

'GA Data
Public Type Individual
  ChromosomeX(Maxstring) As Byte
  ChromosomeY(Maxstring) As Byte
  ObjectiveFunc As Single
  Fitness As Single
End Type

Public Type Population
  TypePop As Individual
End Type

Public NewPop(2) As Population
Public PoolPop(Maxpool) As Population
Public CurrentPop(Maxpop) As Population
Public Current_SOF As Single

Public Popsize As Byte    'Integer global variables
Public Pcross As Single, Pmutation As Single   'Real global
varibles
Dim Action As String 'What has occured (crossover, mutation,
etc..)
Public Parent1 As Byte, Parent2 As Byte 'Parents that were
selected
Public Nmutation As Integer, Ncross As Integer, Ngener As
Integer    'Integer Statistics
Public Maximum As Single, Maximum2 As Single, Minimum As Single,
Average As Single 'Population statistics
Public MaxSolution As Byte, MaxSolution2 As Byte, MinSolution As
Byte
Public MinObjFunc(10) As Single
Public RelocationCost(10) As Integer
Public OptSol(10, 10, 2) As Byte
Public Percent_Occ(10) As Single
Public RunTime(10) As Single

'Optimization Data
Public Delta As Single, Convergence As Single
Public InitalChoice As Integer 'Initial pool of choice
```

```
Public A As Integer, B As Byte 'Transformation of Obj Func to
Fitness

Public CG_X(25) As Single
Public CG_Y(25) As Single
Public Prox(10, 20, 20) As Single 'Proximity Matrix
Public Reloc(10, 10) As Single 'Relocation Weights

'Facility & Activity Data
'Maximum of 10 phases and 10 temporary facilities per phase

Public Fac_Name(10, 10) As String
Public Fac_Length(10, 10) As Integer
Public Fac_width(10, 10) As Integer
Public Phase_Name(10) As String
Public Phase_num As Byte
Public NumTemp(10) As Byte, NumFixed(10) As Byte
Public TotalNumber(10) As Byte 'Sum of temporary and fixed
facilities
Public Start(10) As Byte
Public Finish(10) As Byte
Public PhaseLength(10) As Byte
Public PFac_Name(10, 10) As String
Public PCentroid(10, 10, 2) As Integer
Public Phase As Integer 'The phase number currently being
optimized
'Geometrical Data
Public NumofPoints(10) As Integer

Public AvailableX(10, Maxpoints) As Byte
Public AvailableY(10, Maxpoints) As Byte
Public OccupiedX(10, Maxpoints) As Byte
Public OccupiedY(10, Maxpoints) As Byte
Public ReservedPts As Integer

Sub StaticOptimization()
'Optimization Procedure
Dim S As Variant, F As Variant

S = Timer
Chromosome
ObjectiveFunc
PopSort
Initialize

Do
Sort 'done
Generate
Statistics
OutputData
frmOpmz2.ProgressBar1.Value = 400 * Delta ^ 2 - 400 * Delta +
100
Loop Until Delta < Convergence / 100
frmOpmz2.ProgressBar1.Value = 90
RefineSolution
SaveResults
```

```
MinObjFunc(Phase) = Minimum * PhaseLength(Phase)
frmOpmz2.ProgressBar1.Value = 100
frmOpmz2.Action_lbl = "Opimization Complete"
F = Timer
RunTime(Phase) = F - S
frmOpmz2.Time_lbl.Caption = RunTime(Phase)
End Sub


Private Sub Chromosome()
Dim random As Integer
Dim j As Byte, N As Integer
frmOpmz2.Action_lbl.Caption = "Initializing first population"
ReservedPts = 1
row = 0
For N = 1 To InitalChoice
For j = 1 To NumTemp(Phase)
10  Randomize
    random = Int((Rnd * NumofPoints(Phase)) + 1)
    PoolPop(N).TypePop.ChromosomeX(j) = AvailableX(Phase,
random)
    PoolPop(N).TypePop.ChromosomeY(j) = AvailableY(Phase,
random)
       'Check to make sure facility fits on site:
       If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
       'Check for no overlap, reserving places for placed
facilities:
       If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
Next j
EmptyOccupied
frmOpmz2.ProgressBar1.Value = (N / InitalChoice) * 100
frmOpmz2.Refresh
Next N
End Sub


Public Function CheckSite(Xmin As Byte, Ymin As Byte, X As
Integer, Y As Integer) As Boolean
Dim X1 As Byte, Y1 As Byte, i As Integer
For X1 = Xmin To Xmin + X - 1
   For Y1 = Ymin To Ymin + Y - 1
      For i = 1 To NumofPoints(Phase)
         If X1 = AvailableX(Phase, i) And Y1 = AvailableY(Phase,
i) Then GoTo 5
      Next i
      CheckSite = False
      GoTo 10
5     Next Y1
Next X1
CheckSite = True
10 End Function
```

```
Public Function CheckOverlap(Xmin As Byte, Ymin As Byte, X As
Integer, Y As Integer) As Boolean
Dim X1 As Byte, Y1 As Byte, i As Integer
For X1 = Xmin To Xmin + X - 1
   For Y1 = Ymin To Ymin + Y - 1
      For i = 1 To ReservedPts
         If X1 = OccupiedX(Phase, i) And Y1 = OccupiedY(Phase,
i) Then
         CheckOverlap = False
         GoTo 10
         End If
      Next i
    Next Y1
Next X1
CheckOverlap = True
'Reserve space for placed facility:
For X1 = Xmin To Xmin + X - 1
    For Y1 = Ymin To Ymin + Y - 1
        OccupiedX(Phase, ReservedPts) = X1
        OccupiedY(Phase, ReservedPts) = Y1
        ReservedPts = ReservedPts + 1
    Next Y1
Next X1
10 End Function


Public Sub EmptyOccupied()
Dim i As Integer
For i = 1 To ReservedPts
    OccupiedX(Phase, i) = 0
    OccupiedY(Phase, i) = 0
Next i
ReservedPts = 1
End Sub


Public Sub PopSort()
Dim Min As Single
Dim i As Integer
Dim j As Integer
Dim Flag As Integer
For i = 1 To Popsize
    Min = 100000000
    For j = 1 To InitalChoice
        If PoolPop(j).TypePop.ObjectiveFunc < Min Then
            Flag = j
            Min = PoolPop(j).TypePop.ObjectiveFunc
        End If
    Next j
    For j = 1 To NumTemp(Phase)
        CurrentPop(i).TypePop.ChromosomeX(j) =
PoolPop(Flag).TypePop.ChromosomeX(j)
        CurrentPop(i).TypePop.ChromosomeY(j) =
PoolPop(Flag).TypePop.ChromosomeY(j)
```

```
        CurrentPop(i).TypePop.ObjectiveFunc =
PoolPop(Flag).TypePop.ObjectiveFunc
    Next j
    CurrentPop(i).TypePop.Fitness = (A /
PoolPop(Flag).TypePop.ObjectiveFunc) ^ B
    PoolPop(Flag).TypePop.ObjectiveFunc = 100000000
Next i
End Sub


Private Sub ObjectiveFunc()
Dim i As Byte, j As Byte, k As Byte
Dim N As Integer
Dim ObjFunc As Single
Dim d(20, 20) As Single
For N = 1 To InitalChoice
    'Centroid Position
    For i = 1 To NumTemp(Phase)
        CG_X(i) = PoolPop(N).TypePop.ChromosomeX(i) +
Fac_Length(Phase, i) / 2
        CG_Y(i) = PoolPop(N).TypePop.ChromosomeY(i) +
Fac_width(Phase, i) / 2
    Next i
    j = 1
    For i = 1 + NumTemp(Phase) To TotalNumber(Phase)
        CG_X(i) = PCentroid(Phase, j, 1)
        CG_Y(i) = PCentroid(Phase, j, 2)
        j = j + 1
    Next i
    'objective function
    ObjFunc = 0
    k = 2
    For i = 1 To TotalNumber(Phase)
        For j = k To TotalNumber(Phase)
            d(i, j) = Distance(CG_X(i), CG_Y(i), CG_X(j),
CG_Y(j))
            ObjFunc = ObjFunc + d(i, j) * Prox(Phase, i, j)
        Next j
        k = k + 1
    Next i
    PoolPop(N).TypePop.ObjectiveFunc = ObjFunc
Next N
End Sub


Public Function Distance(X1 As Single, Y1 As Single, X2 As
Single, Y2 As Single) As Single
Distance = ((X1 - X2) ^ 2 + (Y1 - Y2) ^ 2) ^ 0.5
End Function


Public Sub Sort()
Dim i As Byte
Dim Max As Single, Min As Single
Max = CurrentPop(10).TypePop.ObjectiveFunc
Min = CurrentPop(10).TypePop.ObjectiveFunc
```

```
For i = 1 To Popsize
    If CurrentPop(i).TypePop.ObjectiveFunc < Min Then
        Min = CurrentPop(i).TypePop.ObjectiveFunc
        MinSolution = i
    End If
    If CurrentPop(i).TypePop.ObjectiveFunc > Max Then
        Maximum2 = Max
        Max = CurrentPop(i).TypePop.ObjectiveFunc
        MaxSolution2 = MaxSolution
        MaxSolution = i
    End If
Next i
Maximum = Max
Minimum = Min
End Sub


Public Sub Generate()
'This procedure generates a random offspring.
Dim i As Byte, j As Byte, k As Byte
Dim TempX As Integer, TempY As Integer
Dim jcross As Integer
Dim mate1, mate2 As Integer
Dim X As Single

'generates new offspring from oldpop
CurrentSOF
Select Case Flip(Pcross, Pmutation)
Case 1
Nmutation = Nmutation + 1
Action = "Mutation"
Mutation
Case 2
'crossover here
Ncross = Ncross + 1
Action = "Crossover"
Crossover
Case 3
    ' No crossover, just roulette wheel selection
Action = "Copy Parents"
CopyParents
End Select
End Sub


Public Sub Mutation()
Dim lop As Integer, h As Integer
Dim Best As Single
Dim m As Byte, T As Byte, d As Byte, j As Byte
lop = 0
  m = MinSolution 'The best solution so far
  Best = CurrentPop(m).TypePop.Fitness
  For T = 1 To NumTemp(Phase)
    For h = -1 To 1 Step 2
        For d = 0 To 1
        Select Case d
```

```
        'Change in the X direction
        Case 0
        On Error Resume Next
        CurrentPop(m).TypePop.ChromosomeX(T) =
CurrentPop(m).TypePop.ChromosomeX(T) + h
        ObjectiveFunction (m)
        'Check that the objective function has improved
        If CurrentPop(m).TypePop.Fitness < Best Then
            'The new offspring are not better than the worst
population member
            CurrentPop(m).TypePop.ChromosomeX(T) =
CurrentPop(m).TypePop.ChromosomeX(T) - h
            ObjectiveFunction (m)
            GoTo 5
        End If
   'CHECK MUTATION IS FEASIBLE
        EmptyOccupied
        For j = 1 To NumTemp(Phase)
        If CheckSite(CurrentPop(m).TypePop.ChromosomeX(j),
CurrentPop(m).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then
            CurrentPop(m).TypePop.ChromosomeX(T) =
CurrentPop(m).TypePop.ChromosomeX(T) - h
            GoTo 5
        End If

        If CheckOverlap(CurrentPop(m).TypePop.ChromosomeX(j),
CurrentPop(m).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then
            CurrentPop(m).TypePop.ChromosomeX(T) =
CurrentPop(m).TypePop.ChromosomeX(T) - h
            GoTo 5
        End If
        Next j
        GoTo 10
        'Change in the Y direction
        Case 1
        On Error Resume Next
        CurrentPop(m).TypePop.ChromosomeY(T) =
CurrentPop(m).TypePop.ChromosomeY(T) + h
        ObjectiveFunction (m)
        'Check that the objective function has improved
        If CurrentPop(m).TypePop.Fitness < Best Then
            'The new offspring are not better than the worst
population member
            CurrentPop(m).TypePop.ChromosomeY(T) =
CurrentPop(m).TypePop.ChromosomeY(T) - h
            ObjectiveFunction (m)
            GoTo 5
        End If
   'CHECK MUTATION IS FEASIBLE
        EmptyOccupied
        For j = 1 To NumTemp(Phase)
        If CheckSite(CurrentPop(m).TypePop.ChromosomeX(j),
CurrentPop(m).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then
```

```
            CurrentPop(m).TypePop.ChromosomeY(T) =
CurrentPop(m).TypePop.ChromosomeY(T) - h
            GoTo 5
        End If
        If CheckOverlap(CurrentPop(m).TypePop.ChromosomeX(j),
CurrentPop(m).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then
            CurrentPop(m).TypePop.ChromosomeY(T) =
CurrentPop(m).TypePop.ChromosomeY(T) - h
            GoTo 5
        End If
        Next j
        GoTo 10
        End Select
5         Next d
    Next h
  Next T
10 End Sub


Public Sub Crossover()
Dim i As Byte, j As Byte, k As Byte
Dim temp1(Maxstring)  As Byte, temp2(Maxstring) As Byte,
temp3(Maxstring) As Byte, temp4(Maxstring) As Byte
Dim TempX As Byte, TempY As Byte
Dim jcross As Integer
Dim mate1, mate2 As Integer
Dim Worst As Single 'Worst fitness so far
'i and k are the chromosome number to replace
i = MaxSolution
k = MaxSolution2
Worst = Maximum2
10 mate1 = SelectChrom(Popsize, Current_SOF)
   mate2 = SelectChrom(Popsize, Current_SOF)
    Randomize
    jcross = Int(((NumTemp(Phase) - 1) * Rnd) + 1)
    '1st half of exchange
      For j = 1 To jcross
      NewPop(1).TypePop.ChromosomeX(j) =
CurrentPop(mate1).TypePop.ChromosomeX(j)
      NewPop(1).TypePop.ChromosomeY(j) =
CurrentPop(mate1).TypePop.ChromosomeY(j)
      NewPop(2).TypePop.ChromosomeX(j) =
CurrentPop(mate2).TypePop.ChromosomeX(j)
      NewPop(2).TypePop.ChromosomeY(j) =
CurrentPop(mate2).TypePop.ChromosomeY(j)
      Next j
    '2nd half of exchange
      For j = jcross + 1 To NumTemp(Phase)
      TempX = CurrentPop(mate1).TypePop.ChromosomeX(j)
      TempY = CurrentPop(mate1).TypePop.ChromosomeY(j)
      NewPop(1).TypePop.ChromosomeX(j) =
CurrentPop(mate2).TypePop.ChromosomeX(j)
      NewPop(1).TypePop.ChromosomeY(j) =
CurrentPop(mate2).TypePop.ChromosomeY(j)
      NewPop(2).TypePop.ChromosomeX(j) = TempX
```

```
      NewPop(2).TypePop.ChromosomeY(j) = TempY
      Next j
    'Check that solution is feasible
    EmptyOccupied
    For j = 1 To NumTemp(Phase)
    If CheckSite(NewPop(1).TypePop.ChromosomeX(j),
NewPop(1).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
    If CheckOverlap(NewPop(1).TypePop.ChromosomeX(j),
NewPop(1).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
    Next j
    EmptyOccupied
    For j = 1 To NumTemp(Phase)
    If CheckSite(NewPop(2).TypePop.ChromosomeX(j),
NewPop(2).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
    If CheckOverlap(NewPop(2).TypePop.ChromosomeX(j),
NewPop(2).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
    Next j
    'Place Newpop in place of Oldpop
    For j = 1 To NumTemp(Phase)
        temp1(j) = CurrentPop(i).TypePop.ChromosomeX(j)
        temp2(j) = CurrentPop(i).TypePop.ChromosomeY(j)
        temp3(j) = CurrentPop(k).TypePop.ChromosomeX(j)
        temp4(j) = CurrentPop(k).TypePop.ChromosomeY(j)
        CurrentPop(i).TypePop.ChromosomeX(j) =
NewPop(1).TypePop.ChromosomeX(j)
        CurrentPop(i).TypePop.ChromosomeY(j) =
NewPop(1).TypePop.ChromosomeY(j)
        CurrentPop(k).TypePop.ChromosomeX(j) =
NewPop(2).TypePop.ChromosomeX(j)
        CurrentPop(k).TypePop.ChromosomeY(j) =
NewPop(2).TypePop.ChromosomeY(j)
    Next j
    'Place new offspring on the spreadsheet
    ObjectiveFunction (i)
    ObjectiveFunction (k)
    'Check that the objective function has improved
    If CurrentPop(i).TypePop.ObjectiveFunc > Worst Or
CurrentPop(k).TypePop.ObjectiveFunc > Worst Then
    'The new offspring are not better than the worst population
member
     For j = 1 To NumTemp(Phase)
        CurrentPop(i).TypePop.ChromosomeX(j) = temp1(j)
        CurrentPop(i).TypePop.ChromosomeY(j) = temp2(j)
        CurrentPop(k).TypePop.ChromosomeX(j) = temp3(j)
        CurrentPop(k).TypePop.ChromosomeY(j) = temp4(j)
     Next j
     ObjectiveFunction (i)
     ObjectiveFunction (k)
     GoTo 10
    End If
Parent1 = mate1
Parent2 = mate2
```

```
ObjectiveFunction (i)
ObjectiveFunction (k)
End Sub


Public Sub CopyParents()
Dim i As Byte, j As Byte, k As Byte
Dim mate1, mate2 As Integer
'i and k are the chromosome number to replace
i = MaxSolution
k = MaxSolution2
    mate1 = SelectChrom(Popsize, Current_SOF)
    mate2 = SelectChrom(Popsize, Current_SOF)
    For j = 1 To NumTemp(Phase)
      CurrentPop(i).TypePop.ChromosomeX(j) =
CurrentPop(mate1).TypePop.ChromosomeX(j)
      CurrentPop(i).TypePop.ChromosomeY(j) =
CurrentPop(mate1).TypePop.ChromosomeY(j)
      CurrentPop(k).TypePop.ChromosomeX(j) =
CurrentPop(mate2).TypePop.ChromosomeX(j)
      CurrentPop(k).TypePop.ChromosomeY(j) =
CurrentPop(mate2).TypePop.ChromosomeY(j)
    Next j
    'Place new offspring on the spreadsheet
Parent1 = mate1
Parent2 = mate2
ObjectiveFunction (i)
ObjectiveFunction (k)
End Sub


Public Function SelectChrom(Popsize As Byte, Current_SOF As
Single) As Byte
'This function selects a chromosome based on roulette wheel
selection
'Note: currentpop is the population to select from
Dim rand, partsum As Single
Dim j As Byte
partsum = 0
j = 0
Randomize
rand = Rnd * Current_SOF
Do
  j = j + 1
  partsum = partsum + CurrentPop(j).TypePop.Fitness
Loop Until partsum >= rand Or j = Popsize
SelectChrom = j
End Function


Private Sub Statistics()
Dim i As Byte
Dim Max As Single, Min As Single, Total As Single
Max = 0
Min = 100000000
Total = 0
```

```
For i = 1 To Popsize
    If CurrentPop(i).TypePop.ObjectiveFunc < Min Then
        Min = CurrentPop(i).TypePop.ObjectiveFunc
        MinSolution = i
    End If
    If CurrentPop(i).TypePop.ObjectiveFunc > Max Then
        Max = CurrentPop(i).TypePop.ObjectiveFunc
        MaxSolution = i
    End If
    Total = Total + CurrentPop(i).TypePop.ObjectiveFunc
Next i
Average = Total / Popsize
Maximum = Max
Minimum = Min
Delta = (Maximum - Minimum) / Maximum
Ngener = Ngener + 1
End Sub
Sub OutputData()
Dim i As Byte
frmOpmz2.Min_lbl.Caption = Minimum  'Min
frmOpmz2.Avg_lbl.Caption = Average 'Avg
frmOpmz2.Ncross_lbl.Caption = Ncross   'Number of crossovers
frmOpmz2.Nmut_lbl.Caption = Nmutation    'number of mutations
frmOpmz2.Gnr_lbl.Caption = Ngener    'number of generations
frmOpmz2.Refresh
End Sub


Public Function Flip(ProbCr As Single, ProbMu As Single) As Byte
Randomize
If Rnd < ProbMu Then
Flip = 1 'Mutation
ElseIf Rnd < ProbCr Then
Flip = 2 'Crossover
Else
Flip = 3 'Neither
End If
End Function

Public Sub ObjectiveFunction(StringNum As Byte)
Dim i As Byte, j As Byte, k As Byte
Dim ObjFunc As Single
Dim d(20, 20) As Single
    'Centroid Position
    For i = 1 To NumTemp(Phase)
        CG_X(i) = CurrentPop(StringNum).TypePop.ChromosomeX(i) +
Fac_Length(Phase, i) / 2
        CG_Y(i) = CurrentPop(StringNum).TypePop.ChromosomeY(i) +
Fac_width(Phase, i) / 2
    Next i
    j = 1
    For i = 1 + NumTemp(Phase) To TotalNumber(Phase)
        CG_X(i) = PCentroid(Phase, j, 1)
        CG_Y(i) = PCentroid(Phase, j, 2)
        j = j + 1
    Next i
```

```
  'objective function
    ObjFunc = 0
    k = 2
    For i = 1 To TotalNumber(Phase)
        For j = k To TotalNumber(Phase)
            d(i, j) = Distance(CG_X(i), CG_Y(i), CG_X(j),
CG_Y(j))
            ObjFunc = ObjFunc + d(i, j) * Prox(Phase, i, j)
        Next j
        k = k + 1
    Next i
    CurrentPop(StringNum).TypePop.ObjectiveFunc = ObjFunc
    CurrentPop(StringNum).TypePop.Fitness = (A / ObjFunc) ^ B
End Sub


Public Sub CurrentSOF()
Dim i As Byte
Current_SOF = 0
For i = 1 To Popsize
    Current_SOF = Current_SOF + CurrentPop(i).TypePop.Fitness
Next i
End Sub


Sub Initialize()
frmOpmz2.Action_lbl.Caption = "Running GA"
Nmutation = 0
Ncross = 0
Ngener = 0
End Sub

Private Sub RefineSolution()
Dim i As Byte
frmOpmz2.Action_lbl.Caption = "Refining Solution"
For i = 1 To 20
    Mutation
    Statistics
    OutputData
Next i
End Sub


Private Sub SaveResults()
Dim i As Byte
For i = 1 To NumTemp(Phase)
    OptSol(Phase, i, 1) =
CurrentPop(MinSolution).TypePop.ChromosomeX(i)
    OptSol(Phase, i, 2) =
CurrentPop(MinSolution).TypePop.ChromosomeY(i)
Next i
End Sub
```

DynaOpt1 (critical phase approach)

```
Const Maxpop = 8000
Const Maxstring = 10 'Also maximum number of facilities
Const Maxpoints = 10000 'Maximum number of grid squares
'A variable that specifies wether a facility is present in a
previous phase or not
Dim PR(10, 10) As Byte
Dim AmountofReloc(Maxpop) As Integer
Public TypeofOpt As Byte 'Indicates which type of optimization
is performed
Public CrPhase As Byte 'The most critical phase
Public Stp As Integer 'value indicates wether we are moving in
forward or backward order


Sub FacPresence()
Dim i As Integer, j As Byte, k As Byte
'Backward Order
    For i = CrPhase - 1 To 1 Step -1
        For j = 1 To NumTemp(i)
            For k = 1 To NumTemp(i + 1)
                If Fac_Name(i + 1, k) = Fac_Name(i, j) Then
                    PR(i, j) = k
                    GoTo 10
                End If
            Next k
10            Next j
    Next i
'Forward order
For i = CrPhase + 1 To Phase_num
    For j = 1 To NumTemp(i)
        For k = 1 To NumTemp(i - 1)
                 If Fac_Name(i - 1, k) = Fac_Name(i, j) Then
                    PR(i, j) = k
                    GoTo 20
                End If
            Next k
20        Next j
    Next i
End Sub


Sub DynamicOptimization1()
Dim m As Variant
FacPresence
'Backward
frmOpmz4.Progress_lbl = "Backward Chronological Order"
If CrPhase = 1 Then GoTo 10
For Phase = CrPhase - 1 To 1 Step -1
    Stp = -1
    frmOpmz4.Phase_lbl.Caption = Phase_Name(Phase)
    frmOpmz4.Dur_lbl.Caption = Finish(Phase) - Start(Phase)
    DynamicPro
    frmOpmz4.ProgressBar2.Value = frmOpmz4.ProgressBar2.Value +
(100 / Phase_num - 1)
    frmOpmz4.List3.AddItem frmOpmz4.List1.List(0)
    frmOpmz4.List1.RemoveItem (0)
```

```
Next Phase
'Forward
10 If CrPhase = Phase_num Then GoTo 20
frmOpmz4.Progress_lbl = "Forward Chronological Order"
For Phase = CrPhase + 1 To Phase_num

    frmOpmz4.Phase_lbl.Caption = Phase_Name(Phase)
    frmOpmz4.Dur_lbl.Caption = PhaseLength(Phase)
    Stp = 1
    DynamicPro
    frmOpmz4.ProgressBar2.Value = frmOpmz4.ProgressBar2.Value +
(100 / Phase_num - 1)
    frmOpmz4.List3.AddItem frmOpmz4.List2.List(0)
    frmOpmz4.List2.RemoveItem (0)
Next Phase
20 End Sub


Private Sub DynamicPro()
'Optimization Procedure
Dim S As Variant, F As Variant
S = Timer
Chromosome  'done
ObjectiveFunc
PopSort
Initialize
Do
Sort 'done
Generate
Statistics
OutputData
frmOpmz4.ProgressBar1.Value = 400 * Delta ^ 2 - 400 * Delta +
100
Loop Until Delta < Convergence / 100
frmOpmz4.ProgressBar1.Value = 90
RefineSolution
SaveResults
frmOpmz4.ProgressBar1.Value = 100
frmOpmz4.Action_lbl = "Opimization Complete"
F = Timer
RunTime(Phase) = F - S
frmOpmz4.Time_lbl.Caption = RunTime(Phase)
End Sub


Private Sub Chromosome()
Dim random As Integer
Dim j As Byte, N As Integer
frmOpmz4.Action_lbl.Caption = "Initializing first population"
ReservedPts = 1
row = 0
For N = 1 To InitalChoice
For j = 1 To NumTemp(Phase)
If PR(Phase, j) = 0 Then 'New facility
10  Randomize
    random = Int((Rnd * NumofPoints(Phase)) + 1)
```

```
    PoolPop(N).TypePop.ChromosomeX(j) = AvailableX(Phase,
random)
    PoolPop(N).TypePop.ChromosomeY(j) = AvailableY(Phase,
random)
       'Check to make sure facility fits on site:
       If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
       'Check for no overlap, reserving places for placed
facilities:
       If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
Else 'Facility was present in previous phase
    Randomize
    If Rnd > 0.5 Then 'Facility will be placed in its same
position
        i = PR(Phase, j)
        PoolPop(N).TypePop.ChromosomeX(j) = OptSol(Phase - Stp,
i, 1)
        PoolPop(N).TypePop.ChromosomeY(j) = OptSol(Phase - Stp,
i, 2)
        'Check to make sure facility fits on site:
        If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
        'Check for no overlap, reserving places for placed
facilities:
        If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
    Else
    'Facility is placed in random order
20  Randomize
    random = Int((Rnd * NumofPoints(Phase)) + 1)
    PoolPop(N).TypePop.ChromosomeX(j) = AvailableX(Phase,
random)
    PoolPop(N).TypePop.ChromosomeY(j) = AvailableY(Phase,
random)
       'Check to make sure facility fits on site:
       If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
       'Check for no overlap, reserving places for placed
facilities:
       If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
    End If
End If
Next j
EmptyOccupied
frmOpmz4.ProgressBar1.Value = (N / InitalChoice) * 100
frmOpmz4.Refresh
Next N
End Sub
```

```
Private Sub PopSort()
Dim Min As Single
Dim i As Integer
Dim j As Integer
Dim Flag As Integer
For i = 1 To Popsize
    Min = 1000000
    For j = 1 To InitalChoice
        If PoolPop(j).TypePop.ObjectiveFunc < Min Then
            Flag = j
            Min = PoolPop(j).TypePop.ObjectiveFunc
        End If
    Next j
    For j = 1 To NumTemp(Phase)
        CurrentPop(i).TypePop.ChromosomeX(j) =
PoolPop(Flag).TypePop.ChromosomeX(j)
        CurrentPop(i).TypePop.ChromosomeY(j) =
PoolPop(Flag).TypePop.ChromosomeY(j)
        CurrentPop(i).TypePop.ObjectiveFunc =
PoolPop(Flag).TypePop.ObjectiveFunc
    Next j
    CurrentPop(i).TypePop.Fitness = (A /
PoolPop(Flag).TypePop.ObjectiveFunc) ^ B
    PoolPop(Flag).TypePop.ObjectiveFunc = 1000000
Next i
End Sub


Private Sub ObjectiveFunc()
Dim i As Byte, j As Byte, k As Byte
Dim N As Integer
Dim ObjFunc As Single
Dim d(20, 20) As Single
For N = 1 To InitalChoice
    'Centroid Position
    For i = 1 To NumTemp(Phase)
        CG_X(i) = PoolPop(N).TypePop.ChromosomeX(i) +
Fac_Length(Phase, i) / 2
        CG_Y(i) = PoolPop(N).TypePop.ChromosomeY(i) +
Fac_width(Phase, i) / 2
    Next i
    j = 1
    For i = 1 + NumTemp(Phase) To TotalNumber(Phase)
        CG_X(i) = PCentroid(Phase, j, 1)
        CG_Y(i) = PCentroid(Phase, j, 2)
        j = j + 1
    Next i
    'objective function
    ObjFunc = 0
    k = 2
    For i = 1 To TotalNumber(Phase)
        For j = k To TotalNumber(Phase)
```

```
            d(i, j) = Distance(CG_X(i), CG_Y(i), CG_X(j),
CG_Y(j))
            ObjFunc = ObjFunc + d(i, j) * Prox(Phase, i, j)
        Next j
        k = k + 1
    Next i
    For i = 1 To NumTemp(Phase)
         j = PR(Phase, i)
         If j = 0 Then GoTo 10
         If PoolPop(N).TypePop.ChromosomeX(i) <> OptSol(Phase -
Stp, j, 1) Or PoolPop(N).TypePop.ChromosomeY(i) <> OptSol(Phase
- Stp, j, 2) Then
            ObjFunc = ObjFunc + Reloc(Phase, i)
         End If
10  Next i
    PoolPop(N).TypePop.ObjectiveFunc = ObjFunc
Next N
End Sub


Private Sub Sort()
Dim i As Byte
Dim Max As Single, Min As Single
Max = CurrentPop(10).TypePop.ObjectiveFunc
Min = CurrentPop(10).TypePop.ObjectiveFunc
For i = 1 To Popsize
    If CurrentPop(i).TypePop.ObjectiveFunc < Min Then
        Min = CurrentPop(i).TypePop.ObjectiveFunc
        MinSolution = i
    End If
    If CurrentPop(i).TypePop.ObjectiveFunc > Max Then
        Maximum2 = Max
        Max = CurrentPop(i).TypePop.ObjectiveFunc
        MaxSolution2 = MaxSolution
        MaxSolution = i
    End If
Next i
Maximum = Max
Minimum = Min
End Sub



Private Sub Statistics()
Dim i As Byte
Dim Max As Single, Min As Single, Total As Single
Max = 0
Min = 100000000
Total = 0
For i = 1 To Popsize
    If CurrentPop(i).TypePop.ObjectiveFunc < Min Then
        Min = CurrentPop(i).TypePop.ObjectiveFunc
        MinSolution = i
    End If
    If CurrentPop(i).TypePop.ObjectiveFunc > Max Then
        Max = CurrentPop(i).TypePop.ObjectiveFunc
```

```
        MaxSolution = i
    End If
    Total = Total + CurrentPop(i).TypePop.ObjectiveFunc
Next i
Average = Total / Popsize
Maximum = Max
Minimum = Min
Delta = (Maximum - Minimum) / Maximum
Ngener = Ngener + 1
End Sub
Sub OutputData()
Dim i As Byte
frmOpmz4.Min_lbl.Caption = Minimum  'Min
frmOpmz4.Avg_lbl.Caption = Average 'Avg
frmOpmz4.Ncross_lbl.Caption = Ncross   'Number of crossovers
frmOpmz4.Nmut_lbl.Caption = Nmutation    'number of mutations
frmOpmz4.Gnr_lbl.Caption = Ngener    'number of generations
frmOpmz4.Refresh
End Sub


Public Sub ObjectiveFunction(StringNum As Byte)
Dim i As Byte, j As Byte, k As Byte
Dim ObjFunc As Single
Dim d(20, 20) As Single
AmountofReloc(StringNum) = 0
    'Centroid Position
    For i = 1 To NumTemp(Phase)
        CG_X(i) = CurrentPop(StringNum).TypePop.ChromosomeX(i) +
Fac_Length(Phase, i) / 2
        CG_Y(i) = CurrentPop(StringNum).TypePop.ChromosomeY(i) +
Fac_width(Phase, i) / 2
    Next i
    j = 1
    For i = 1 + NumTemp(Phase) To TotalNumber(Phase)
        CG_X(i) = PCentroid(Phase, j, 1)
        CG_Y(i) = PCentroid(Phase, j, 2)
        j = j + 1
    Next i
  'objective function
    ObjFunc = 0
    k = 2
    For i = 1 To TotalNumber(Phase)
        For j = k To TotalNumber(Phase)
            d(i, j) = Distance(CG_X(i), CG_Y(i), CG_X(j),
CG_Y(j))
            ObjFunc = ObjFunc + d(i, j) * Prox(Phase, i, j)
        Next j
        k = k + 1
    Next i
    For i = 1 To NumTemp(Phase)
         j = PR(Phase, i)
         If j = 0 Then GoTo 10
         If CurrentPop(StringNum).TypePop.ChromosomeX(i) <>
OptSol(Phase - Stp, j, 1) Or
```

```
CurrentPop(StringNum).TypePop.ChromosomeY(i) <> OptSol(Phase -
Stp, j, 2) Then
            ObjFunc = ObjFunc + Reloc(Phase, i)
            AmountofReloc(StringNum) = AmountofReloc(StringNum)
+ Reloc(Phase, i)
         End If
10  Next i
    CurrentPop(StringNum).TypePop.ObjectiveFunc = ObjFunc
    CurrentPop(StringNum).TypePop.Fitness = (A / ObjFunc) ^ B
End Sub


Sub Initialize()
frmOpmz4.Action_lbl.Caption = "Running GA"
Nmutation = 0
Ncross = 0
Ngener = 0
End Sub

Private Sub RefineSolution()
Dim i As Byte
frmOpmz4.Action_lbl.Caption = "Refining Solution"
For i = 1 To 20
    Mutation
    Statistics
    OutputData
Next i
End Sub

Private Sub SaveResults()
Dim i As Byte
For i = 1 To NumTemp(Phase)
    OptSol(Phase, i, 1) =
CurrentPop(MinSolution).TypePop.ChromosomeX(i)
    OptSol(Phase, i, 2) =
CurrentPop(MinSolution).TypePop.ChromosomeY(i)
Next i
MinObjFunc(Phase) = Minimum * PhaseLength(Phase)
RelocationCost(Phase) = AmountofReloc(MinSolution)
End Sub
```

Dyna_Opt2 (Mini-Min approach)

```
Const Maxpop = 1000
Const Maxstring = 10 'Also maximum number of facilities
Const Maxpoints = 10000 'Maximum number of grid squares
'A variable that specifies wether a facility is present in a
previous phase or not
Dim PR(10, 10) As Byte
Dim AmountofReloc(Maxpop) As Integer
Public OptSol2(10, 10, 10, 2) As Byte  'for 2nd type of
optimization
Public RelocationCost2(10, 10) As Integer, RelocationCost3(10)
```

```
Public TotalCost2(10, 10) As Single, TotalCost3(10) As Single


Sub FacPresence()
Dim i As Integer, j As Byte, k As Byte
'Backward Order
    For i = CrPhase - 1 To 1 Step -1
        For j = 1 To NumTemp(i)
            For k = 1 To NumTemp(i + 1)
                If Fac_Name(i + 1, k) = Fac_Name(i, j) Then
                    PR(i, j) = k
                    GoTo 10
                End If
            Next k
10            Next j
    Next i
'Forward order
For i = CrPhase + 1 To Phase_num
    For j = 1 To NumTemp(i)
        For k = 1 To NumTemp(i - 1)
                 If Fac_Name(i - 1, k) = Fac_Name(i, j) Then
                    PR(i, j) = k
                    GoTo 20
                End If
            Next k
20        Next j
    Next i
End Sub


Sub DynamicOptimization2()
Dim m As Variant
OptSolChange
For CrPhase = 1 To Phase_num
    Unload frmOpmz5
    frmOpmz5.Show
    FacPresence
    'Backward
    frmOpmz5.Progress_lbl = "Backward Chronological Order"
    If CrPhase = 1 Then GoTo 5
    For Phase = CrPhase - 1 To 1 Step -1
        Stp = -1
        frmOpmz5.Phase_lbl.Caption = Phase_Name(Phase)
        frmOpmz5.Dur_lbl.Caption = Finish(Phase) - Start(Phase)
        DynamicPro
        frmOpmz5.ProgressBar2.Value =
frmOpmz5.ProgressBar2.Value + (100 / Phase_num - 1)
        frmOpmz5.List3.AddItem frmOpmz5.List1.List(0)
        frmOpmz5.List1.RemoveItem (0)
    Next Phase
    'Forward
5   If CrPhase = Phase_num Then GoTo 15
    frmOpmz5.Progress_lbl = "Forward Chronological Order"
    For Phase = CrPhase + 1 To Phase_num
        frmOpmz5.Phase_lbl.Caption = Phase_Name(Phase)
        frmOpmz5.Dur_lbl.Caption = PhaseLength(Phase)
```

```
        Stp = 1
        DynamicPro
        frmOpmz5.ProgressBar2.Value =
frmOpmz5.ProgressBar2.Value + (100 / Phase_num - 1)
        frmOpmz5.List3.AddItem frmOpmz5.List2.List(0)
        frmOpmz5.List2.RemoveItem (0)
    Next Phase
15      ResultsByInitial
Next CrPhase
20 End Sub


Private Sub DynamicPro()
'Optimization Procedure
Dim S As Variant, F As Variant
S = Timer
Chromosome  'done
ObjectiveFunc
PopSort
Initialize
Do
Sort 'done
Generate
Statistics
OutputData
frmOpmz5.ProgressBar1.Value = 400 * Delta ^ 2 - 400 * Delta +
100
Loop Until Delta < Convergence / 100
frmOpmz5.ProgressBar1.Value = 90
RefineSolution
SaveResults
frmOpmz5.ProgressBar1.Value = 100
frmOpmz5.Action_lbl = "Optimization Complete"
F = Timer
frmOpmz5.Time_lbl.Caption = F - S
End Sub


Private Sub Chromosome()
Dim random As Integer
Dim j As Byte, N As Integer, row As Integer
frmOpmz5.Action_lbl.Caption = "Initializing first population"
ReservedPts = 1
row = 0
For N = 1 To InitalChoice
For j = 1 To NumTemp(Phase)
If PR(Phase, j) = 0 Then 'New facility
10  Randomize
    random = Int((Rnd * NumofPoints(Phase)) + 1)
    PoolPop(N).TypePop.ChromosomeX(j) = AvailableX(Phase,
random)
    PoolPop(N).TypePop.ChromosomeY(j) = AvailableY(Phase,
random)
       'Check to make sure facility fits on site:
```

```
       If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
       'Check for no overlap, reserving places for placed
facilities:
       If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 10
Else 'Facility was present in previous phase
    Randomize
    If Rnd > 0.5 Then 'Facility will be placed in its same
position
        i = PR(Phase, j)
        PoolPop(N).TypePop.ChromosomeX(j) = OptSol2(CrPhase,
Phase - Stp, i, 1)
        PoolPop(N).TypePop.ChromosomeY(j) = OptSol2(CrPhase,
Phase - Stp, i, 2)
        'Check to make sure facility fits on site:
        If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
        'Check for no overlap, reserving places for placed
facilities:
        If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
    Else
    'Facility is placed in random order
20  Randomize
    random = Int((Rnd * NumofPoints(Phase)) + 1)
    PoolPop(N).TypePop.ChromosomeX(j) = AvailableX(Phase,
random)
    PoolPop(N).TypePop.ChromosomeY(j) = AvailableY(Phase,
random)
       'Check to make sure facility fits on site:
       If CheckSite(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
       'Check for no overlap, reserving places for placed
facilities:
       If CheckOverlap(PoolPop(N).TypePop.ChromosomeX(j),
PoolPop(N).TypePop.ChromosomeY(j), Fac_Length(Phase, j),
Fac_width(Phase, j)) = False Then GoTo 20
    End If
End If
Next j
EmptyOccupied
frmOpmz5.ProgressBar1.Value = (N / InitalChoice) * 100
frmOpmz5.Refresh
Next N
End Sub


Sub OutputData()
Dim i As Byte
frmOpmz5.Min_lbl.Caption = Minimum  'Min
```

```
frmOpmz5.Avg_lbl.Caption = Average 'Avg
frmOpmz5.Ncross_lbl.Caption = Ncross   'Number of crossovers
frmOpmz5.Nmut_lbl.Caption = Nmutation    'number of mutations
frmOpmz5.Gnr_lbl.Caption = Ngener    'number of generations
frmOpmz5.Refresh
End Sub


Sub Initialize()
frmOpmz5.Action_lbl.Caption = "Running GA"
Nmutation = 0
Ncross = 0
Ngener = 0
End Sub


Private Sub RefineSolution()
Dim i As Byte
frmOpmz5.Action_lbl.Caption = "Refining Solution"
For i = 1 To 20
    Mutation
    Statistics
    OutputData
Next i
End Sub

Private Sub SaveResults()
For i = 1 To NumTemp(Phase)
    OptSol2(CrPhase, Phase, i, 1) =
CurrentPop(MinSolution).TypePop.ChromosomeX(i)
    OptSol2(CrPhase, Phase, i, 2) =
CurrentPop(MinSolution).TypePop.ChromosomeY(i)
Next i
MinObjFunc(Phase) = Minimum * PhaseLength(Phase)
RelocationCost(Phase) = AmountofReloc(MinSolution)
RelocationCost2(CrPhase, Phase) = RelocationCost(Phase)
TotalCost2(CrPhase, Phase) = MinObjFunc(Phase)
End Sub


Private Sub ResultsByInitial()
RelocationCost3(CrPhase) = 0
TotalCost3(CrPhase) = 0
For i = 1 To Phase_num
    RelocationCost3(CrPhase) = RelocationCost3(CrPhase) +
RelocationCost2(CrPhase, i)
    TotalCost3(CrPhase) = TotalCost3(CrPhase) +
TotalCost2(CrPhase, i)
Next i
End Sub


Private Sub OptSolChange()
For i = 1 To Phase_num
    For j = 1 To Phase_num
        For k = 1 To NumTemp(j)
```

```
            OptSol2(i, j, k, 1) = OptSol(j, k, 1)
            OptSol2(i, j, k, 2) = OptSol(j, k, 2)
        Next k
    Next j
Next i
For i = 1 To Phase_num
    TotalCost2(i, i) = MinObjFunc(i)
Next i
End Sub
```

APPENDIX B: SPACE IDENTIFICATION CODE

```
Const Max_Lines = 50
Dim Num_Eqn As Byte, X(Max_Lines) As Double, Y(Max_Lines) As
Double
Dim InteriorPoint As Variant
Dim A(Max_Lines) As Double, B(Max_Lines) As Double
Dim Xmax As Integer, Ymax As Integer, Xmin As Integer, Ymin As
Integer
Dim NumofPoints As Integer
Dim PoolX(30000) As Integer, PoolY(30000) As Integer
Dim Centroid(15, 2) As Single
Public NumFixed As Byte, NumObst As Byte
Dim PFac_Name(10) As String 'Names of permenant/fixed facilities
Public Pitch As Integer
Dim Flag(Max_Lines) As Byte


Sub MainProgram()
UserForm1.Show
On Error GoTo ErrorHandler
BoundarySelection
GetEquations
GetPoints
GetPermenantFac
GetObstacles
Redimension
SendToFile
Exit Sub
ErrorHandler:
UserForm3.Show
End Sub


Private Sub BoundarySelection()
Dim SitePolyLine As AcadLWPolyline
Dim Pickedpt As Variant
Dim PolyLineVertices As Variant
Dim Vertix(100) As Double
Dim i As Byte 'Counter
With ThisDrawing.Utility
.GetEntity SitePolyLine, Pickedpt, vbCr & "Select site
boundaries (Must be polyline)"
InteriorPoint = .GetPoint(, vbCr & "Select any point inside the
boundary")
End With
SitePolyLine.GetBoundingBox Min, Max
Xmin = Min(0)
Ymin = Min(1)
Xmax = Max(0)
Ymax = Max(1)
PolyLineVertices = SitePolyLine.Coordinates
For i = 0 To 2 * Max_Lines Step 2
  On Error GoTo 10
```

```
  X(i / 2) = PolyLineVertices(i)
  Y(i / 2) = PolyLineVertices(i + 1)
Next i

10 Num_Eqn = (i - 2) / 2 '# of lines of polygon
End Sub


Private Sub GetEquations()
For i = 1 To Num_Eqn
   If X(i) = X(i - 1) Then
            B(i) = X(i)
            If X(i) < InteriorPoint(0) Then
            Flag(i) = 3
            Else: Flag(i) = 4
            End If
            GoTo 10
   End If
   If Y(i) = Y(i - 1) Then
            B(i) = Y(i)
            If Y(i) < InteriorPoint(1) Then
            Flag(i) = 5
            Else: Flag(i) = 6
            End If
            GoTo 10
   End If
    A(i) = (Y(i) - (Y(i - 1))) / (X(i) - X(i - 1))
   B(i) = Y(i) - (A(i) * X(i))
      'If Py > a*Px + b then "greater"
      If InteriorPoint(1) > A(i) * InteriorPoint(0) + B(i) Then
         Flag(i) = 1
      Else
         Flag(i) = 2
      End If
10  Next i
 End Sub


Private Sub GetPoints()
Dim Counter As Byte
Dim j As Integer
Dim X As Integer, Y As Integer
For X = Xmin To Xmax - 1
    For Y = Ymin To Ymax - 1
        Counter = 0
        For i = 1 To Num_Eqn
            Select Case Flag(i)
                Case 1
                If Y < A(i) * X + B(i) Then GoTo 10
                Case 2
                If Y > A(i) * X + B(i) Then GoTo 10
                Case 3
                If X < B(i) Then GoTo 10
                Case 4
                If X > B(i) Then GoTo 10
                Case 5
```

```
                If Y < B(i) Then GoTo 10
                Case 6
                If Y > B(i) Then GoTo 10
            End Select
        Next i
             PoolX(j) = X
             PoolY(j) = Y
             j = j + 1
10     Next Y
Next X
NumofPoints = j + 1
End Sub


Private Sub FacilityBoundary()
Dim FacilityPolyLine As AcadLWPolyline
Dim X_min As Integer, Y_min As Integer, X_max As Integer, Y_max
As Integer
With ThisDrawing.Utility
.GetEntity FacilityPolyLine, Pickedpt, vbCr & "Select site
boundaries (Must be polyline)"
End With
FacilityPolyLine.GetBoundingBox Min, Max
X_min = Min(0)
Y_min = Min(1)
X_max = Max(0)
Y_max = Max(1)
End Sub


Private Sub GetPermenantFac()
    Dim X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As
Integer
    Dim FacilityBorder(10) As AcadLWPolyline
    Dim PermenantFac(10) As AcadText
    'delete the selection set if it already exists
For i = 1 To NumFixed
    With ThisDrawing.Utility
.GetEntity FacilityBorder(i), Pickedpt, vbCr & "Select Permenant
Facility (must be polyline)"
FacilityBorder(i).Color = acMagenta
FacilityBorder(i).Update
End With
Next i
For i = 1 To NumFixed
FacilityBorder(i).Color = acWhite
FacilityBorder(i).Update
Next i
    For i = 1 To NumFixed
      FacilityBorder(i).GetBoundingBox Min, Max
      '-1 so that facilities can be placed exactly adjacent to
permenant objects
      X1 = Min(0)
      Y1 = Min(1)
      X2 = Max(0) - 1
      Y2 = Max(1) - 1
```

```
      Centroid(i, 1) = X1 + ((X2 - X1 + 1) * 0.5)
      Centroid(i, 2) = Y1 + ((Y2 - Y1 + 1) * 0.5)
      Call RemoveOccupiedPlaces(X1, Y1, X2, Y2)
    Next i
For i = 1 To NumFixed
ThisDrawing.Utility.GetEntity PermenantFac(i), Pickedpt, vbCr &
"Select Permenant Facility Name (Follow the same order of
previous selection!)"
PFac_Name(i) = PermenantFac(i).TextString
PermenantFac(i).Color = acMagenta
Next i
For i = 1 To NumFixed
PermenantFac(i).Color = acWhite
PermenantFac(i).Update
Next i
End Sub


Private Sub GetObstacles()
    Dim X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As
Integer
    Dim FacilityBorder(10) As AcadLWPolyline
    Dim PermenantFac As AcadText
    'delete the selection set if it already exists
For i = 1 To NumObst
    With ThisDrawing.Utility
.GetEntity FacilityBorder(i), Pickedpt, vbCr & "Select Site
Obstacles (must be polyline)"
FacilityBorder(i).Color = acMagenta
FacilityBorder(i).Update
End With
Next i
For i = 1 To NumObst
FacilityBorder(i).Color = acWhite
FacilityBorder(i).Update
Next i
   For i = 1 To NumObst
      FacilityBorder(i).GetBoundingBox Min, Max
      '-1 so that facilities can be placed exactly adjacent to
permenant objects
      X1 = Min(0)
      Y1 = Min(1)
      X2 = Max(0) - 1
      Y2 = Max(1) - 1
      Call RemoveOccupiedPlaces(X1, Y1, X2, Y2)
    Next i
End Sub


Private Sub RemoveOccupiedPlaces(X1 As Integer, Y1 As Integer,
X2 As Integer, Y2 As Integer)
Dim Ywidth As Integer, Xwidth As Integer
Ywidth = Y2 - Y1
Xwidth = X2 - X1
con = 1
For j = 1 To NumofPoints
```

```
        If PoolX(j) = X1 And PoolY(j) = Y1 Then
        For m = 0 To Ywidth
            PoolX(j + m) = 9999
            PoolY(j + m) = 9999
        Next m
        j = j + Ywidth
        X1 = X1 + con
        If X1 = X2 + 1 Then GoTo 10
    End If
Next j
10  End Sub


Private Sub Redimension()
c = 0
For j = 1 To NumofPoints
    If PoolX(j) = 9999 Then
        For i = j To NumofPoints
            PoolX(i) = PoolX(i + 1)
            PoolY(i) = PoolY(i + 1)
        Next i
        j = j - 1
        c = c + 1
    End If
Next j
NumofPoints = NumofPoints - c
End Sub


Private Sub SendToFile()
    'sending line data to Text File
    '.lin is the line data file
    '.per is the permenant facilities file
    Open "c:\Temp\EDSLP.LIN" For Output As #1
    Write #1, NumofPoints - 1
    For i = 1 To NumofPoints - 1
        Write #1, PoolX(i - 1), PoolY(i - 1)
    Next i
    Close #1
    Open "c:\Temp\EDSLP.PER" For Output As #1
    Write #1, NumFixed
    For i = 1 To NumFixed
        Write #1, PFac_Name(i), Centroid(i, 1), Centroid(i, 2)
    Next i
    Close #1
End Sub


Sub CheckPoints()
Dim PPoints As AcadPoint
Dim P(2) As Double
P(2) = 0
Open "c:\temp\edslp.lin" For Input As #1
Input #1, NumofPoints
For i = 1 To NumofPoints
    Input #1, P(0), P(1)
```

```
    Set PPoints = ThisDrawing.ModelSpace.AddPoint(P)
Next i
Close #1
End Sub
```

APPENDIX C: SOLUTION REPRESENTATION CODE

```
Public ACADFileName(10) As String
Public fMainForm As frmMain
Option Explicit
'This module sends line data to autocad


Public Sub DrawFacilities(N As Byte)
Dim AutoCADapplication As AcadApplication
Dim PLinePoints(14) As Double
Dim TextPoint(2) As Double
Dim Point1(2) As Double, Point2(2) As Double, Point3(2) As
Double, Point4(2) As Double
Dim Facility As AcadPolyline
Dim FacilityText As AcadText
Dim i As Byte, j As Byte
Set AutoCADapplication = CreateObject("AutoCAD.Application")
AutoCADapplication.Visible = True
AutoCADapplication.Documents.Open ACADFileName(N)
Point1(2) = 0
Point2(2) = 0
Point3(2) = 0
Point4(2) = 0
For j = 1 To NumTemp(N)
Point1(0) = OptSol(N, j, 1)
Point1(1) = OptSol(N, j, 2)
Point2(0) = Point1(0) + Fac_Length(N, j)
Point2(1) = Point1(1)
Point3(0) = Point2(0)
Point3(1) = Point2(1) + Fac_width(N, j)
Point4(0) = Point1(0)
Point4(1) = Point3(1)
For i = 0 To 2
PLinePoints(i) = Point1(i)
PLinePoints(i + 3) = Point2(i)
PLinePoints(i + 6) = Point3(i)
PLinePoints(i + 9) = Point4(i)
PLinePoints(i + 12) = Point1(i)
Next i
TextPoint(0) = Point1(0) + 0.1
TextPoint(1) = Point1(1) + 0.1
TextPoint(2) = 0
AutoCADapplication.ActiveDocument.ActiveLayer =
AutoCADapplication.ActiveDocument.Layers("Facilities")
Set Facility =
AutoCADapplication.ActiveDocument.ModelSpace.AddPolyline(PLinePo
ints)
```

```
Set FacilityText =
AutoCADapplication.ActiveDocument.ModelSpace.AddText(Fac_Name(N,
j), TextPoint, 1)
Next j
TextPoint(0) = 0
TextPoint(1) = 0
Set FacilityText =
AutoCADapplication.ActiveDocument.ModelSpace.AddText(MinObjFunc(
N), TextPoint, 1.5)
End Sub

Public Sub DrawFacilities2(Ini As Byte, N As Byte)
Dim AutoCADapplication As AcadApplication
Dim PLinePoints(14) As Double
Dim TextPoint(2) As Double
Dim Point1(2) As Double, Point2(2) As Double, Point3(2) As
Double, Point4(2) As Double
Dim Facility As AcadPolyline
Dim FacilityText As AcadText
Dim i As Byte, j As Byte
Set AutoCADapplication = CreateObject("AutoCAD.Application")
AutoCADapplication.Visible = True
AutoCADapplication.Documents.Open ACADFileName(N)
Point1(2) = 0
Point2(2) = 0
Point3(2) = 0
Point4(2) = 0
For j = 1 To NumTemp(N)
Point1(0) = OptSol2(Ini, N, j, 1)
Point1(1) = OptSol2(Ini, N, j, 2)
Point2(0) = Point1(0) + Fac_Length(N, j)
Point2(1) = Point1(1)
Point3(0) = Point2(0)
Point3(1) = Point2(1) + Fac_width(N, j)
Point4(0) = Point1(0)
Point4(1) = Point3(1)
For i = 0 To 2
PLinePoints(i) = Point1(i)
PLinePoints(i + 3) = Point2(i)
PLinePoints(i + 6) = Point3(i)
PLinePoints(i + 9) = Point4(i)
PLinePoints(i + 12) = Point1(i)
Next i
TextPoint(0) = Point1(0) + 0.1
TextPoint(1) = Point1(1) + 0.1
TextPoint(2) = 0
AutoCADapplication.ActiveDocument.ActiveLayer =
AutoCADapplication.ActiveDocument.Layers("Facilities")
Set Facility =
AutoCADapplication.ActiveDocument.ModelSpace.AddPolyline(PLinePo
ints)
Set FacilityText =
AutoCADapplication.ActiveDocument.ModelSpace.AddText(Fac_Name(N,
j), TextPoint, 1.5)
Next j
```

```
Set FacilityText =
AutoCADapplication.ActiveDocument.ModelSpace.AddText(MinObjFunc(
i), TextPoint, 1.5)
End Sub
```