

Addis Ababa University
Addis Ababa Institute of Technology
School of Civil and Environmental Engineering



Cost Optimization of Reinforced Concrete Frames Using Genetic Algorithms

by

Elias Yilma

Sponsor: Ethiopian Roads Authority(ERA)

March 21, 2017

A Thesis submitted at Addis Ababa Institute of Technology
in partial fulfillment of the requirements for
the Degree of Master of Science in Structural Engineering

The undersigned have examined the thesis entitled '**Cost Optimization of Reinforced Concrete Frames Using Genetic Algorithms**' presented by **Elias Yilma**, a candidate for the degree of Master of Science and hereby certify that it is worthy of acceptance.

Advisor

Signature

Date

Internal Examiner

Signature

Date

External Examiner

Signature

Date

Chair Person

Signature

Date

UNDERTAKING

I certify that the research work titled “**Cost Optimization of Reinforced Concrete Frames Using Genetic Algorithms**” is my own work. The work has not been presented elsewhere for assessment. Where material has been used from other sources, it has been properly acknowledged /referred.

Elias Yilma

Abstract

In this research, cost optimization of reinforced concrete plane and space frames using genetic algorithms is presented. Previous research endeavors have used highly simplified continuous and discrete optimization models, while ignoring constructibility. Additionally, the optimization of 3D frames was done without due consideration to shear, torsional effects and sizing constraints. This paper constructs an optimization model with these points in mind. An integrated software architecture has been developed to implement the method, and benchmark examples were used to compare its feasibility and efficiency with the traditional assume-check design method as well as other heuristic methods. Design check procedures were based on Eurocode-2(2004). Performance and computational complexity of the algorithm is developed along with a thorough study assessing the effect of genetic parameters such as mutation and crossover on the optimization process. The method developed proves to be superior as compared to the traditional design paradigm and heuristic methods.

Keywords: Concrete Frame Optimization, Structural Optimization, Genetic Algorithms, Eurocode , Sizing Optimization, Object-Oriented Model

ACKNOWLEDGMENTS

First and foremost, I cannot thank Dr. Bedilu Habte enough for his relentless involvement and effort in guiding me from the inception of this research through its completion, for his timely feedback and suggestions and for steering me in the right direction whenever I got lost .

I would also like to acknowledge my friends and colleagues for providing me with valuable insight in the thesis work. I can boldly say some of their comments were critical enough to shape this paper to its present state.

Finally, I'm without words to express my heartfelt appreciation and gratefulness to my family, for their undying love and support is what got me through tough and frustrating times while doing this work.

Contents

1	Introduction	1
1.1	Background	1
1.2	Statement of the problem	2
1.3	Objectives	2
1.3.1	General Objectives	2
1.3.2	Specific Objectives	2
1.4	Scope	3
2	Basics	4
2.1	General	4
2.2	Deterministic Methods	4
2.2.1	Mathematical Programming	4
2.2.2	The Lagrangian Multiplier Method and Kuhn-Tucker Conditions	5
2.2.3	Optimality Criteria Methods	6
2.3	Meta-heuristic Methods	6
2.4	Genetic Algorithms	7
2.4.1	Description	7
2.4.2	Genetic Operators	8
2.4.2.1	Reproduction	8
2.4.2.2	Crossover	9
2.4.2.3	Mutation	9
2.4.3	Genetic Algorithms and Optimization	9
3	State of the Art	11
3.1	Limitations of Mathematical Optimization Methods	11
3.2	Genetic Modeling of Concrete Frames	11
3.2.1	Representation and Modeling for Concrete Frames	11
3.2.2	Problem Formulation	12
3.2.3	Genetic Encoding	13
3.2.4	Design Variables	14
3.2.5	Design and Detailing	14

3.2.6	Optimization of Space Frames	15
3.2.7	Convergence	15
3.2.8	Computational Complexity	16
3.3	Observations	16
4	Implementation	18
4.1	Introduction	18
4.1.1	General Program Structure	18
4.2	Structural Analysis and Design Actions(Alg. 4.1)	19
4.2.1	Structural Analysis	19
4.2.2	Improvements to the Stiffness Solver	21
4.2.3	Computation of Design Actions	22
4.3	Design and Constraint Algorithms	23
4.3.1	Assumptions	23
4.3.2	Design Checks for Bending(Alg. 4.14, 4.15)	23
4.3.3	Design Checks for Shear(Alg. 4.18)	25
4.3.4	Design Checks for Torsion (Alg. 4.19, 4.20)	25
4.3.5	Sizing Constraints	26
4.4	Optimization Formulation	28
4.5	Genetic Modeling	28
4.5.1	Member Sizing	29
4.5.2	Reinforcement Encoding	29
4.6	Genetic Optimization	31
4.6.1	Cost Estimation and Penalty (Alg. 4.26)	31
4.6.2	The Fitness Function (Alg. 4.27)	31
4.6.3	Simulation(Alg. 4.21)	32
4.7	HELIX: A Structural Optimization Platform	32
4.7.1	Why Java?	33
4.7.2	Main Interface and Grid Definition	33
4.7.3	Ribbon Functions	34
4.7.4	Sub-menus	36
4.7.5	Input and Output Data	39
4.7.6	Evolutionary Simulation	39
4.7.7	The Result and Section Dialogs	40
4.8	Algorithm Listings	41
4.8.1	Structural Analysis	41
4.8.2	Structural Constraints	48
4.8.3	Genetic Simulation	52
4.8.4	Flow Charts	57

5	Results and Discussion	59
5.1	Introduction	59
5.2	Models for Design Results Verification	59
5.3	Verification Environment	60
5.4	MODEL 1: Validation using a Benchmark	60
5.4.1	Problem Description	60
5.4.2	Design Space	61
5.4.3	Genetic Simulation	62
5.4.4	Comparison	62
5.5	MODEL 2: Validation using ETABS	65
5.5.1	Problem Description	65
5.5.2	Results and Comparison	66
5.6	Summary of Results	68
5.7	Effect of GA Parameters	68
5.7.1	Effect of Population Size	69
5.7.2	Effect of Mutation	70
5.7.3	Effect of Evolution Period (Generation Size)	71
5.8	Performance and Profiling	72
5.8.1	Computation Time	72
5.8.2	Memory Usage	73
6	Conclusions	75
6.1	General	75
6.2	Achievements and Observations	75
6.2.1	Achievements	75
6.2.2	Observations	76
6.3	Future Work	77
6.3.1	Basics	77
6.3.2	Multi-objective Optimization	77
6.3.3	Structural Analysis	78
6.3.4	Parallel Computing	78
6.3.5	Custom Constraint Formulation	78
A	Cost Evaluation for Benchmark	80
A.1	Volume of Concrete	80
A.2	Quantity of Reinforcement Steel	81
A.3	Formwork and Scaffolding	83
A.4	Cost summary	84

List of Algorithms

4.1	Summary of the Stiffness Method	41
4.2	Computation of Member Parameters	42
4.3	Computation of Local Stiffness Matrix	43
4.4	Member Rotation Matrix	43
4.5	Computation of Member Global Stiffness Matrix	44
4.6	Computation of Member End Actions	44
4.7	Computation of Equivalent End Actions	44
4.8	Stiffness Matrix Assembly	45
4.9	Application of Boundary Conditions	46
4.10	Reaction Stiffness Matrix Assembly	46
4.11	Computation of Displacements	47
4.12	Computation of Reactions	47
4.13	The Conjugate Gradient Method	47
4.14	Algorithm for Determining Biaxial Interaction Boundary Coordinates of Members	48
4.15	Efficiency Computation for Biaxial Capacity	49
4.16	Check for a Point Residing in a Closed Boundary	49
4.17	Check for a Point Residing in a Triangle	50
4.18	Shear Resistance Penalty for Reinforced Concrete Beam Elements	50
4.19	Torsion and Shear-Torsion Interaction Penalties	51
4.20	Combined Interaction Capacity for Shear and Torsional Actions	51
4.21	A Simple Genetic Algorithm	52
4.22	Uniform Crossover	53
4.23	Mutation	53
4.24	Tournament Selection of Best Individuals	54
4.25	Gene Decoding	54
4.26	The Penalty Function	55
4.27	Individual Fitness Computation	55
4.28	Total Penalty of a Structure	56
4.29	Total Cost of a Structure	56

List of Tables

2.1	Reproduction	8
3.1	Comparison of Past Research Endeavors	17
4.1	Memory Consumption by Matrices	21
4.2	Penalty Values for Biaxial Bending	24
4.3	Maximum and Minimum Reinforcement Amounts for Structural Members	26
4.4	Value Encoding for the Sizing of Members	29
4.5	Continuous Reinforcements	30
4.6	Extra Reinforcement Encoding	30
4.7	Shear Link Spacing Indices	31
4.8	Examples of Fitness Evaluation	32
4.9	Main Ribbon Functions	35
4.10	Input and Output Data for Analysis, Design and Optimization	39
5.1	Benchmark Computer Specifications	60
5.2	Material Parameters	60
5.3	Cost Parameters	61
5.4	Optimally Designed Columns for the GA simulation	63
5.5	Optimally Designed Beams for the GA simulation	64
5.6	Optimally Designed Beams for the GA simulation	67
5.7	ETABS Design Results	67
5.8	Optimally Designed Columns for the GA simulation	68
5.9	Summary of Models 1 and 2	68
5.10	Simulation results for varying Population Size	70
5.11	Simulation results for varying mutation probabilities	71
5.12	Simulation time for different frame sizes	72
A.1	Total Concrete Volume	80
A.2	Flexural Reinforcement Schedule(Beams)	81
A.3	Reinforcement Schedule(Columns)	82
A.4	Shear Reinforcement Schedule(Beams)	83

A.5 Formwork and Scaffolding	83
A.6 Cost Summary	84

List of Figures

2.1	Crossover	9
2.2	Mutation	9
4.1	Efficiency Computation for Biaxial Bending Interactions	25
4.2	Main Canvas Components	34
4.3	Sub-Menus 1	37
4.4	Sub-Menus 2	38
4.5	Part of the Plot Dialog	40
4.6	Main Program Flow Chart	57
4.7	Fitness Computation Flow Chart	58
5.1	Benchmark 2D Frame	62
5.2	Reinforcement Configuration for Beams	64
5.3	2-story 1-bay Space Frame	65
5.4	Population Size vs. Fitness	69
5.5	Mutation Probability vs. Fitness	71
5.6	Total Time Vs. NNZ	73
5.7	Memory Usage Vs. NNZ	74

Chapter 1

Introduction

1.1 Background

Traditional practices of reinforced concrete structural design follow an estimate-analyze-check paradigm so as to minimize computational time and effort. Optimization, if included, would comprise of adjusting a couple of section sizes and reinforcement details without due regard to any form of scientific based approach. Recent trends in research, however, have focused more on grounding design to its performance and resource consumption roots. This shift in the design paradigm has thus resulted in the development of new methods and tools, and modification of old ones, to allow for extensive optimization of resource and form for structures. With the computational power and availability of computers ever increasing, recent years have yielded substantial progress into non-deterministic search-based optimization methods. Synchronously, structural optimization methods have also embraced these tools as they are truly designed for managing complex problem spaces.

Optimization of structural frames is a complex problem involving several variables and constraints, especially as the topology of the said structure grows large. Mathematical methods (such as the Lagrangian family of methods, the Runge-Kutta family of methods, and several other numerical procedures) for such problems become unmanageable, as their symbolic rather than numerical nature prohibits their computerization.

On the flip side, heuristic search-based methods (such as bee/ant algorithms, simulated annealing, neural networks and genetic algorithms), when applied in optimization problems, consider a large set of possibilities for a formulated problem, assess each possible solution and search for the best individual out of these possibilities. Past implementations of these methods have yielded in successful, though, unrealistic results. This arose from the fact that researchers often used extremely simplified structural design models to optimize their problems in the hope of avoiding computational explosion. The research thus focuses on factoring in what has been left out of past approaches and will try, in

parallel, plug in newer and more efficient methods for obtaining optimal and constructible solutions .

1.2 Statement of the problem

The computational problem of structural optimization is a difficult one to tackle, especially if realistic topology, design and detailing models are assumed. For this reason, recent studies have adopted the use of search based methods, primarily genetic algorithms, for optimizing concrete and steel structures. Previous works in search-based structural optimization of concrete structures defined their design problem using very simplified detailing and design assumptions in favor of ensuring time and memory efficiency during simulation. The inclusion of shear design has also not been given much emphasis. This would result in skewed and unrealistic cost and weight values even for the optimized solution. Thus, the inclusion of these parameters will result in a better and more grounded estimate of the objective function.

Additionally, even with the advent of newer genetic encoding schemes and design facilitating techniques, current research is still using traditional formulation schemes such as binary encoding and single point crossovers for carrying out the selection process. The fusion of these new techniques will help remedy the time and memory complexity that has festered in previous implementations. The study of previous algorithms will also identify which processing schemes are the most resource intensive.

1.3 Objectives

1.3.1 General Objectives

- ▷ To *develop* and *solve* the optimization formulation for the minimum *cost* design of structurally detailed *two and three dimensional concrete frames* with the use of *genetic algorithms* and with due consideration to shear design and torsion.

1.3.2 Specific Objectives

- ▷ Formulate and solve the structural optimization problem for finding the least-cost shear- and flexure- efficient solution of two and three dimensional concrete frames with the use of genetic algorithms.
- ▷ Develop an integrated software architecture for testing and using the genetic optimization procedure developed.

- ▷ Study the behavior of convergence and quality of solutions on optimized frames with varying genetic algorithm parameters.
- ▷ Study the computational complexity of the algorithm developed.

1.4 Scope

To limit feature creep, this thesis will not address:

- ▷ ***Inclined and Curved Members:*** All members are assumed to have either horizontal or vertical alignments. Member axes are also assumed to be aligned along the global axes.
- ▷ ***Topological constraints:*** Topology optimization will not be addressed: this implies structural members can not have near zero values for their material and geometric properties
- ▷ ***Design details and algorithms for other types of sections and structural elements:*** Rectangular sections will be the only ones utilized in the work.
- ▷ ***Dynamic loads:*** Time-dependent and oscillatory loadings will not be addressed. Earthquake and wind effects will be transformed to their static counterparts using the appropriate procedures
- ▷ ***Second order effects on columns:*** All columns will be assumed to be short and stocky.

Chapter 2

Basics

2.1 General

This section briefly discusses about the merits and limitations of the traditional mathematical optimization methods and the state of the art on meta-heuristic algorithms. It also gives introductory information about genetic algorithms and their application in optimization.

2.2 Deterministic Methods

2.2.1 Mathematical Programming

In general, optimization problems encountered in design and engineering can be grouped as linear and non-linear based on their function form .The generic form for most optimization problems can be written as:

$$\begin{aligned} \text{minimize objective:} & \quad f(x), & \quad x \in \mathbb{R}^n \\ \text{subject to:} & \quad \phi_j(x) = 0, & \quad (j = 1, 2, \dots, M), \\ & \quad \psi_k(x) \leq 0, & \quad (k = 1, 2, \dots, K), \end{aligned} \tag{2.1}$$

where $f(x)$, $\phi_j(x)$ and $\psi_k(x)$ are functions of the design vector

$$\mathbf{x} = (x_1, x_2, \dots, x_n)^T$$

$f(x)$ is called the objective function, while variables in \mathbf{x} are called decision variables. $\phi_j(x)$ and $\psi_k(x)$ denote constraints of equality and inequality respectively.

Based on the function forms of $f(x)$, $\phi_j(x)$ and $\psi_k(x)$, an optimization problem can be classified as linear or non-linear. Some literatures[30] conclude that this distinction

can be made regardless of $f(x)$. In linear problems, often called linear programming, the relationship between the constraint with the design variables needs to be linear i.e. $f(x)$, $\phi_j(x)$ and $\psi_k(x)$ need to be defined as a linear combination of variables in $\{x\}$ [7]. Linear programming techniques have been applied to simple problems involving a manageable number of design variables. Plastic design of frames and simple truss weight optimizations are primary examples. Most real-world structural optimization problems, however, are non-linear in nature since most relations in stress, deflection, stiffness and material behavior are of that form. Thus, other methods, primarily based on multi-variable calculus, have been developed to directly or indirectly tackle these problems. Most techniques applicable to structural engineering problems are based on the Lagrangian multiplier procedures.

2.2.2 The Lagrangian Multiplier Method and Kuhn-Tucker Conditions

In the Lagrangian multiplier method, an expression, called the Lagrangian is formed as a linear combination of the objective function and the set of active constraints i.e.

$$\mathcal{L}(x, \lambda) = f(x) + \sum \lambda_j \phi_j^*(x) + \sum \mu_k \psi_k^*(x) \quad (2.2)$$

where the asterisk denotes those constraints that are active. The solution then involves finding the unknowns i.e. the decision variables \vec{x} and the Lagrangian multipliers $\vec{\lambda}$ and $\vec{\mu}$. The necessary conditions for finding the unknowns would be to apply the operators ∇_x , ∇_λ and ∇_μ on $\mathcal{L}(x, \lambda, \mu)$ to get a system of equations, after which the system can be solved[23]. The difficulty of obtaining a solution for the equations depends on the nature of the Lagrangian. In addition, as the number of decision variables gets large, which is usually the case in engineering problems, finding an optimum solution and visualizing it would be next to impossible. To accommodate this, many variants of the Lagrangian method have been developed, the most widely used being the Kuhn-Tucker conditions[19]. These conditions assure solutions found by satisfying them are optimal. But then again, applying these conditions to obtain solutions can be difficult in problems as these conditions are valid if the formulated system is a convex programming problem[8]. It should thus be clear that unless for the simplest of design optimization problems, concavity can not be guaranteed in the majority of engineering problems rendering the direct use of these conditions as limited.

2.2.3 Optimality Criteria Methods

The optimality criteria (OC) method is another variant of the Lagrangian multiplier approach, where instead of solving the aforementioned system of equations ($\nabla(\mathcal{L})$) directly, a set of criteria are established and then formulated for the purpose of updating the values of \vec{x} , $\vec{\lambda}$ and $\vec{\mu}$ and afterwards developing a general iterative procedure for the problem[23]. The procedure would start with appropriately assumed values for each variable in \vec{x} , $\vec{\lambda}$ and $\vec{\mu}$. Upon further value updates, the variation of each variable is expected to converge to 0. This procedure is, either manually calculated, or programmed into a computer. OC methods have been successfully applied for engineering optimization problems where the decision variables are continuous. Applying Optimality Criteria methods for problems involving discrete variables, however, requires assuming the discrete variables to be continuous.

2.3 Meta-heuristic Methods

Optimization problems with continuous variables, usually the case in structural optimization problems, have infinite possible solutions but only one global optimum. The process of finding a solution for such problems can then be seen as a problem of search[30]. Mathematics based methods, as discussed above, use pure function forms to study and manipulate the problem's behavior to reach a solution. The main caveat for these methods, however, is that, in order to arrive at the solution, the behavior of the formulation needs to be known. The modeling of discrete variable problems using such methods can also be cumbersome.[7]

In search-based methods, one randomly searches different regions of the search space while being guided towards the optimum point by some mechanism. This allows for the search method not to spend too much time searching for solutions in regions where there is no potential global optimum. In other words, it avoids being stuck at local optimum points for the subsequent iterations. Meta-heuristic methods can also be applied to problems that have a discrete search space. Some algorithms that fall into this category are simulated annealing(SA), swarm algorithms, artificial neural networks(ANNs) and genetic algorithms(GAs). Swarm algorithms[17], simulated annealing[25] and neural networks[1] have been applied successfully in structural design and optimization problems but are not discussed here.

All meta-heuristic methods have the following properties in common:

- ▷ All start at a random point (or several random points) in the search space
- ▷ All have probabilistic convergence behavior

- ▷ All have mechanisms that allows them to propagate through large separate quadrants of the search space in parallel

2.4 Genetic Algorithms

Genetic algorithms(GAs) are a family of meta-heuristic search algorithms based on the Darwinian theory of natural selection and genetics[12]. They are based on selecting solutions that are best fit to the problem at hand while maintaining a diverse set of possible solutions which change and evolve upon subsequent iterations. Developed by John Holland in the 1970s[16], these algorithms have been applied to fields extending from machine learning and artificial intelligence to complex optimization and forecasting problems.

Genetic Algorithms derive their power from the fact that they operate well irrespective of having any information about the problem space[30]. This makes GAs ideal for modeling problems that do not have “nice” mathematical properties such as continuity, concavity and determinism, or for problems that are simply too difficult to describe to a computer.

Genetic Algorithms have been successfully applied in several structural design and optimization problems, including but not limited to weight and topology optimization problems. [4, 7, 9, 10, 15, 24, 26, 27, 28, 31]

2.4.1 Description

Genetic algorithms operate on a collection of individuals, usually called a population. Individuals are sets containing values of the design variables, or in other terms represent single points in a problem’s search space. In order to make the selection and evolution process manageable, the information contained in individuals is stored as encoded strings of character sequences called chromosomes. Several types of encoding design information have been proposed[4].

The process of genetic evolution generally consists of three stages namely, initialization,selection and generation. In initialization, an encoded and randomized set of n individuals is generated. This randomization allows for the exploration of the search space and for maintaining diversity in the population.[12, 16, 18]

In the selection stage, a genetic operator called reproduction is applied, where a percentage of the population that consists of individuals that best fit the problem are selected while the rest of the individuals are discarded. The selection process varies from implementation to implementation[9, 15, 26, 31].

The generation stage consists of applying genetic operators on the selected individuals from the previous stage. Genetic operators allow for new individuals with mixed genetic

materials to be formed, and also randomly modify existing individuals. The most common genetic operators used include reproduction, crossover and mutation. Other operators include dominance, inversion, segregation and migration.

There are several variants of genetic algorithms. The variant used for this work are called Simple Genetic Algorithms(SGAs), while other derivatives, such as memetic, parallel and micro-genetic algorithms, have been developed and used by researchers[24, 20].

2.4.2 Genetic Operators

Genetic operators are the main mechanism through which the mimicking of natural selection processes can be achieved. Simple genetic algorithms, first implemented by Holland[16], was the first one to use reproduction, crossover and mutation as genetic operators.

2.4.2.1 Reproduction

Reproduction is a process of selecting fit individuals for possible genetic recombination and consists of a variety of methods. Its primary purpose is to identify weak and strong individuals, discarding the weak ones and selecting the strong ones. The criteria on which the selection is based on is called fitness. Fitness is a value that weighs if an individual is better suited to solve the problem or not. The higher the fitness, the least suited the individual is. Selection is thus, based on choosing $x\%$ of the individuals with the least fitness factor. Table 2.1 shows the reproduction process for 5 individuals. In the i th generation, the individuals(first column), are shown along with their fitness factor. It can be seen that individuals 1, 4 and 5 have the least fitness factor values out of the 5. Thus, these individuals pass through for the next generation, if a 60% elitism is selected. Only two individuals(1 and 4) would pass through, had a 40% elitism been chosen. The rest of the slots(i.e. slots 2 and 3) would be filled by either replica of the fit individuals, or newly formed offspring. Sometimes A few unfit individuals are added into the mix to diversify the population and avoid uniformity, which in turn helps in evading local optima.

Individual ID	individuals(generation: i)	fitness factor	individuals(generation: $i + 1$)
i1	1 0 2 4 7 ...	0.412	1 0 2 4 7 ...
i2	3 7 5 0 3 ...	1.503	3 7 5 4 7 ... *
i3	4 7 1 0 2 ...	1.205	1 13 4 7 11 ...
i4	1 13 4 7 11 ...	0.335	1 13 4 7 11 ...
i5	4 1 1 2 9 ...	0.679	4 1 1 2 9 ...

*New Individual formed by allowing crossover between individuals i1 and i2 of generation i

Table 2.1: Reproduction

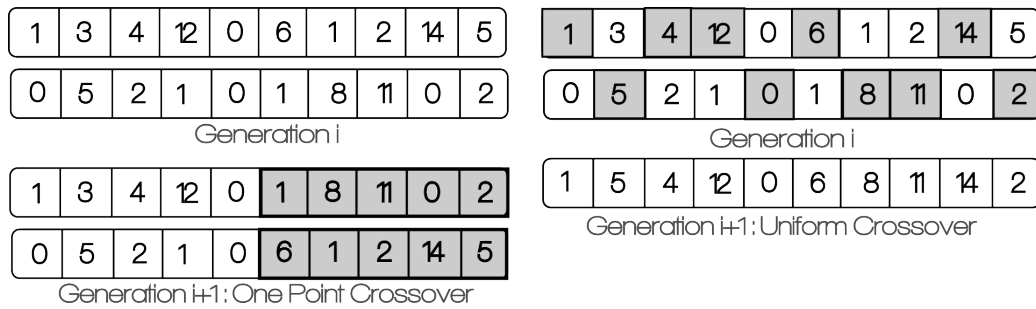


Figure 2.1: Crossover

2.4.2.2 Crossover

Crossover is the primary means of genetic information exchange and is performed with a large probability[12]. Crossover is carried out by interchanging corresponding parts of an individuals chromosome with another. The crossover points can be single or multiple. Uniform crossover is another form where individual gene indices get manipulated instead of segments of the chromosome.

2.4.2.3 Mutation

Mutation is a random modification in an isolated portion of an individuals gene sequence. Mutation is applied as a means of avoiding premature convergence and uniformity among individuals. It is therefore “an insurance policy against premature loss of important notions.”[12]

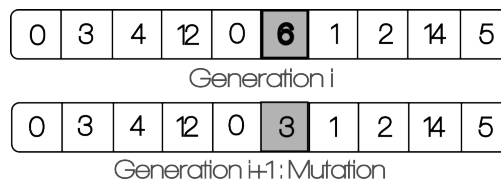


Figure 2.2: Mutation

2.4.3 Genetic Algorithms and Optimization

The previous sub-sections have briefly introduced the basic components of a simple genetic algorithm. Such types of genetic algorithms can be seen as improvised search mechanisms, and can thus be applied on problems with large search spaces. They differ from other mathematical optimization methods in that, GAs have mechanisms for adopting to several problems encountered in optimization iterations. For example, their probabilistic behavior is compatible for problems that have several local optima points. Additionally, GAs have the power to cover most of the solution space without wasting too much time on any particular sub-region. The next chapter discusses how GAs have been used for

several types of structural optimization problems, and their performance and flexibility as compared to other mathematical optimization methods.

Chapter 3

State of the Art

3.1 Limitations of Mathematical Optimization Methods

The previous chapter discussed calculus-based methods that have been developed for the optimization of problems in several fields, including structural design. The most notable ones, the Lagrangian method and the optimality criteria method, could be used for optimizing problems that exhibit continuity in their function forms.

Goldberg[12] notes that these methods have crippling behavior that prohibits their use in complex problem formulations. First, both methods search for solutions locally. If global solutions are required, then several restarts of the same operation need to be performed by some means of a pseudo-random method. Second, these methods strictly require the availability of well-defined gradients and slope values, requiring continuity in their definition.

The following sections critically discuss how previous research attempts have applied genetic-algorithm-based structural optimization for several type of structural systems, mainly steel and concrete frames. Gaps in research and assumption in application of this technique have also been concisely addressed.

3.2 Genetic Modeling of Concrete Frames

3.2.1 Representation and Modeling for Concrete Frames

Concrete frames(usually) consist of horizontal members(beams) and vertical members(columns). Both beams and columns are characterized by their size and reinforcement quantities, though how the reinforcement is arranged depends upon the nature of loading they are exposed to. Then, the optimization problem for concrete frames could be stated as follows:

“For each member, what combination of section size, flexural and shear reinforcement quantities results in the smallest overall material cost for the structure, with all members compliant with the imposed strength, serviceability or constructibility constraints ?”

The mapping of this problem to a genetic algorithm model is achieved by assuming a single size and reinforcement configuration of a structure as an individual. The population would then consist of several of these randomized configurations. The evolution process can be consequently setup by:

- ▷ Generating a population of randomized individuals
- ▷ Decoding: The encoded design information (which contains data about the possible size and reinforcement quantities of each member) needs to be decoded and assigned to the structure. This step is done for every individual in the population.
- ▷ Evaluation: The newly formed structures are then checked if their constituent members violate any constraints. In parallel, the objective function(in this case, cost) is also computed. the individuals are then tagged with the resulting total fitness value.
- ▷ Selection: Based on results obtained from the evaluation, the fit individuals from the populations are selected and are allowed to form new individuals, through crossover. Some individuals also mutate.
- ▷ Loop: The new population (now consisting of the fit individuals from the previous generation and the newly formed offspring) will then go through the previous four steps, until the number of iterations is reached

3.2.2 Problem Formulation

The problem formulation for the optimization of frames depends on two properties, namely material and objective. If the objective that is to be achieved is minimum weight, then the formulation is independent of the materials used. If instead of weight, however, cost is the objective, then the formulation will be dependent on the material used. If structural steel is used, for instance, the cost function can be simply formed as a product of the unit cost of steel with the total weight of the frame. In the case of concrete, though, a linear combination of two materials needs to be utilized, as concrete and reinforcement steel have different unit cost index[26]. The variation of unit costs greatly affects optimal results for concrete frames because the change in cross-section size is(usually) inversely proportional to the amount of reinforcement provided.

Flexure and shear checks in the case of beams, and uniaxial interaction envelopes in the case of columns are used as violable strength constraints for 2D frames, whereas biaxial interaction envelopes are utilized for 3D frames. Researchers have used modified versions

of beam and column design procedures as constraints of the optimization problem. In addition, serviceability criteria have also been included in some studies while maintaining simplicity. The inclusion of shear design, however, has been absent in the reviewed literature.

Structural grouping is the process of classifying similar frame elements (in orientation, loading and position). All elements in a group would then be assigned the same geometric and loading properties. Commendable studies[26, 21] have used structural grouping to help reduce the number of design variables used in the optimization process. This has been a key factor in the development of discrete methods for optimization of concrete structures. For example, column members at the ground level of a frame can be grouped into one, where as roof beams can have another. This removes the necessity for maintaining continuity in addition to reducing the number of design variables. The criteria used in constructing groups vary. A few examples include similarities in detailing of reinforcement, uniformity of cross-sections along an axis,[26] and loading and magnitude of internal forces[21].

3.2.3 Genetic Encoding

Genetic modeling is the process of representing and reformulating a problem so that it can be simulated using genetic algorithms. In its simplest form, it comprises of four subprocesses:

- ▷ a reformulation of the objective function and constraints to form a fitness evaluation function.
- ▷ choosing which design variables need optimization
- ▷ modeling of design variables as either discrete or continuous
- ▷ Selection of type of encoding for representing the design variable vector

Before any selection or generation process begins, the design variables are to be encoded into genetic sequences. Some encoding schemes include binary, quaternary, octal, hexadecimal and real value encodings.

The dominant encoding type, binary digits, were used in the early versions of genetic algorithms for truss optimization, implemented by Holland and Goldberg as cited by Rajeev et al.[26] Several others have, since then, used binary sequences to encode design variables in structural optimization problems[26, 28, 21]. Some[26] have noted that, the larger the number of variables, the longer the chromosome length, which in turn, would require longer encoding and decoding time. It can be realized that some researchers[26, 21] have devised techniques for reducing the number of design variables for the sole purpose

of reducing chromosome length, such that processing can be done faster. Others have chosen to simplify their genetic models in favor of shorter processing time

Later on, other studies[9], have adopted value encoding to successfully perform discrete optimization of structures. Dede et.al.[4] have shown that the type of encoding cannot necessarily guarantee better solutions and depends on several factors such as the nature of mutation, type of crossover and population size. It is also dependent on the size and complexity of the problem.

3.2.4 Design Variables

Most reinforced concrete structural optimization problems are treated as continuous in their function form, and the design variables can theoretically assume any positive real number. Hence, solutions usually consist of a set of real numbers. In practice however, exact values are approximated into discrete sizes and quantities, and are used for construction. Therefore, a better approach would be to list out possible discrete values that can be assumed for each design variable and selecting a combination that would best satisfy the problem. This behavior lends itself to the suitability of applying search based methods. Such techniques have been employed in the past[25, 26, 31] to get discretely optimized values for the design problem, which can then be directly applied.

Most attempts on structure optimization problems assumed only the most critical variables, common ones being section type and material quantity. Later approaches tried to include variables such as classification of reinforcement as negative and positive [15]. Rajeev and Krishnamoorthy[26] have shown that the inclusion of simple reinforcement detailing patterns as a design variable in the formulation of the optimization problem, though complex, can be achieved by factoring in reinforcement arrangements used in current construction practices or from guidelines found from detailing manuals. It should, however, be noted that the detailing arrangements taken into consideration did not reflect actual detailing practices such as anchoring, placement and development lengths, bends and overlaps. Additionally, as such arrangements take more complex forms, the number of design variables and thus, chromosome length also increases. This makes the decoding of the chromosomes generated complex, time-consuming and memory intensive. To alleviate this problem, other researchers[9] have used other types of encoding instead of the traditional binary encoding, to achieve an overall short chromosome length.

3.2.5 Design and Detailing

Studies[15, 24, 27, 31, 29] have used simple top and bottom reinforcement quantities(in addition to breadth and height) at a single section, to represent the detailing for a single structural member in their formulations. Even though, this assertion could lead to theoretically optimized solutions, the results can not be usable as such detailing schemes do

not exist.

Rajeev and Krishnamoorthy[26] have shown that reinforcement detailing arrangements can be considered as design variables without compromising the speed or performance of the algorithm. But in doing so, the chromosome length they have used was observed to be significantly large(in the order of 100's). To alleviate this problem, they grouped similar structural elements as identical in section and materials so as to reduce the number of design variables used in the simulation. The study have also factored in section sizing and reinforcement arrangement of the current construction practice.

Vidoso et.al.[25] have used a structural modeling scheme that incorporated symmetry in structures as well as usage of realistic reinforcement details and section definitions in their optimization formulation.

The inclusion of shear design in concrete structures optimization is of paramount importance for two reasons. First, it would be possible to achieve an end-to-end usable design from the simulation. Second, shear reinforcement holds a considerable percentage of the total weight of reinforcing steel that would be needed in most structural frames. The author notes that the inclusion of shear reinforcement in structural optimization is rarely considered in the literature.

3.2.6 Optimization of Space Frames

The models used to optimize plane portal frames in [25, 26, 31] can be extended to three dimensional portal frames. This involves the addition of torsional forces (which become relevant if bracing elements are existent in the frame), as well as biaxial bending effects to the design procedures. Additionally, structural analysis and topology modelling would need to be done in three dimensions. Reinforcement details for the minor directions in shear and bending would also need to be added as design variables.

3.2.7 Convergence

All genetic algorithms are iterative in nature. Most problems require a certain number of iterations to arrive at a reasonably accurate solution, after which variance between solutions will no longer exist. The number of iterations can be limited by specifying a certain condition or group of conditions called convergence criteria. The convergence criteria specify when an acceptable solution is found and thus when the GA should stop simulating. Convergence criteria for optimization problems have been defined in several ways including:

- ▷ Checking if x% of the population represents the final solution
- ▷ Explicitly stating the number of generations to be simulated for

- ▷ Checking the difference in magnitude between the fittest individuals of the current generation and the previous generation.

That said, it should be noted that convergence criteria don't necessarily ensure the correct solution to be found. This is because genetic algorithms have a probabilistic rate of convergence and a tendency to diverge especially if genetic operator parameters are not selected carefully[4].

3.2.8 Computational Complexity

The problem of execution time is a major issue in most large scale optimization fields especially in iterative or search-based methods. This problem emanates in the fact that the more involved the objective function and the constraints are, the more time it will take for the computer to go through and evaluate parameters for a single solution which would then get compounded for several iterations. Thus, balancing out the formulation of the problem and the assumptions it uses, among other factors, is very important.

Many techniques are available for optimizing memory and time of execution. Perea et.al.[24] have employed massively parallel genetic algorithms to simultaneously evaluate many solutions. Several researchers[15, 21, 26] have used groups to minimize the design check process. Dede et al.[9] used value encoding to reduce chromosome length which would directly affect the encoding/decoding process. Bekiroglu et al.[4] have compared the effect of using different encoding types for structural optimization and its effect on convergence. It can be realized from their result that the performance of the encoding will depend on the size and complexity of the structure considered, in addition to the size of the population and the number of iterations selected for the simulation.

3.3 Observations

For the optimization of complex structural systems, search based methods such as Genetic algorithms need to be employed because calculus based methods creep to a halt when managing the complexities introduced by the nature and sheer quantity of the design variables. Researchers have used very simplified models of structures to apply genetic optimization. This was done to avoid computational explosion that would otherwise occur in the simulations, had a realistic model been used. In particular, previous optimization formulations for concrete frames never considered very important variables such as shear design. Additionally, most ignored the effect of structural detailing norms in their designs which are quintessential in the cost/weight estimation procedures in quantity surveying. These design variables need to be included to arrive at a reasonable and realistic estimate of the objective function, be it cost, weight or structural safety.

Research	Objective	Encoding	Crossover	Structural Detailing	Shear
Rajeev et al.[26]	Cost	Binary	Single-point	Yes*	No
Guerra et al.[15]	Cost	Binary	Single-point	No	No
Yousif[31]	Cost	Binary	Single-point	No	No
Perea et al.[24]	Cost	Binary	Single-point	No	No
Current Research	Cost	Value	Uniform	Yes**	Yes
*For flexure only					
**For flexure, shear and constructibility					

Table 3.1: Comparison of Past Research Endeavors

Several attempts have been made to reduce the search space for general structural optimization problems. A primary example is the discretization of design variables in the formulation of an optimization problem, which would result in a large shrinkage of the search space, thus resulting in quicker searches. Other techniques include structural grouping, which can be used to reduce the number of structural elements for which design checks are to be made.

Another culprit researchers have faced is managing the number of design variables, which indirectly affect the length of chromosome used in the natural selection process. Value and other types of encoding have been used to alleviate this problem. Such efficient forms of genetic encoding can be used to manage the problem of having extremely long chromosomes.

The researcher believes that while most of the problems encountered in this field of research have been partially or fully solved, almost all efforts have comprised their solutions by assuming impractical assertions or leaving out necessary ones, thus not moving far from the status quo. Therefore, in this research, the techniques discussed in previous sections will be combined to form a generalized approach that will be expected to not only achieve *realistic* results but also be *efficient* in doing so.

Chapter 4

Implementation

4.1 Introduction

This chapter is dedicated to explaining the development of the optimization model for highlighted in previous chapters. Additionally, it also discusses implementation details of a program that simulates this optimization model, written in Java. The chapter is not intended to be a comprehensive guide for these details, but is to be used as concise description of most of the algorithms used in the structural analysis, design and optimization phases. Thus, auxiliary topics such as mathematical and geometric algorithms will not be covered.

It is worth to note that most subsections have a corresponding algorithm, located in section 4.8, associated with them. Thus, the reader is encouraged to refer to these algorithms as frequently as possible.

4.1.1 General Program Structure

The software follows a top-down object-oriented class structure where one primary routine can inherit all of its superclasses' attributes. This approach eliminates the need for extremely long and hard-to-debug scripts familiar in programming languages such as FORTRAN. Thus, as dictated by Java's object-oriented paradigm, the program flows in such a way that one entity or data structure contains the other. Highest in the organization are packages, which contain class definitions, data files, images and other files necessary for the isolated operation of the module/package.

The software implemented comprises of four main packages, namely:

- ▷ The Structural Analysis Package: responsible for computing element forces and moments acting on the structure.
- ▷ The Design Package: checks assigned attributes of individual structural member attributes for adequacy in strength and sizing.

- ▷ The Genetic Optimization Package: iterates through a population of candidates and perform evolutionary steps to refine a search space.
- ▷ Auxiliaries: graphical utilities and mathematical libraries constantly utilized by the three other modules.

4.2 Structural Analysis and Design Actions(Alg. 4.1)

All design actions required for the optimization algorithms are obtained from elastic structural analysis. Before structural analysis is carried out, however, loads acting on the structure should be defined and an appropriate method for analysis should be adopted.

In the present version of the program, loads are not differentiated as live, dead or seismic, but instead are to be applied as general actions, either as distributed or point loadings, and as moments or forces. Additionally, the location and extent of these actions needs to be known. All forces should have a unit of kilo Newtons(kN), while all moments are to have units of kilo-Newton-Meters(kN-m).

While several methods are available for the analysis of framed structures, only algorithms based on the stiffness family of methods(such as the finite element method(FEM) and the direct stiffness method(DSM)) can achieve the scalability required for analyzing large structures . Out of the two, the direct stiffness method has been selected for two reasons:

1. Finite element methods perform poorly(time and memory-wise) when used on substantially large structures and especially if the mesh used for defining the structure is very fine.
2. Implementation of the stiffness method is much more manageable and targeted than FEM, though FEM can operate on elements with varying cross-sections and shapes, and can solve several problems with unique boundary conditions.

4.2.1 Structural Analysis

For this work, a straight forward implementation of the direct stiffness method has been used. Details of its implementation can be found in [13]. It should be noted that the algorithm has a limitation in that it doesn't recognize members which are not oriented in the default upwards direction. Thus, the roll angle, α , for all structural members is assumed to be 0° .

The implementation of the analysis program has been modified to satisfy the following performance criteria:

1. **Speed:** In almost all structural design programs, structural analysis takes up the most amount of time as compared to other processes, such as design and post-processing. This is so because most of the steps involved in reaching the solution require large matrix operations. Out of these steps, the notable time consuming processes are:

- (a) **Computation of Element Stiffness Matrices**($O(2n * 12^3)$): This involves the double multiplication of a 12x12 rotation matrix with a 12x12 stiffness matrix for every element in the structure.
- (b) **Assembly of the Structure Stiffness Matrix(SSM)** ($O(n * 12^2)$): The element stiffness matrices formed in step (a) need to be assembled into one stiffness matrix with a shape of $DOF \times DOF$.
- (c) **Application of Boundary Conditions**($O(DOF^2)$): Boundary conditions such as restrained nodes need to be imposed on the structural stiffness matrix. This involves removal of all rows and columns in the SSM corresponding to node indices that are restrained in one or more directions. The reduced stiffness matrix has a size of $uDOF \times uDOF$. $uDOF$ is the total number of unrestrained degrees of freedom.
- (d) **Formation of the Reaction Stiffness Matrix**($O(DOF^2)$): For the purposes of calculating support reactions, a modified version of the SSM needs to be formed. The procedure is similar to step (c) except for, instead of removing unrestrained rows from the SSM, only restrained degrees of freedom are removed row-wise. This results in a matrix of size $cDOF \times uDOF$.
- (e) **Computation of Displacements:** After the reduced stiffness matrix has been formed, the displacements need to be computed as usual by solving the linear system $[K][D] = [F]$, where $[F]$ is the reduced force vector. This step is one of the most time consuming procedures in the analysis algorithm, since the stiffness matrix $[K]$ needs to be appropriately transformed into either a triangular or diagonal form. This step could be achieved through direct as well as iterative methods. The difficulty of reaching the solution within a reasonable time is dependent on the size and regularity of the matrix. Direct methods such as Gauss-Jordan elimination and LU decomposition are well suited for small sized matrices, but start to take up exponentially large amounts of time as this size increases. Iterative methods such as the conjugate gradient method and its preconditioned variants are used for large and usually sparse linear systems, though they are convergent only on specific types of matrices. Proper care needs to be taken on which method is utilized as the wrong choice of technique might have a crippling effect. In addition, some techniques are more adoptive to parallel computing while others are less so.

2. **Memory:** Values obtained from structural analysis consist of forces, moments, displacements and rotations. For these values to be obtained with reasonable accuracy, either single or double precision variables need to be used with a reasonable number of significant figures. This usually offsets the amount of memory(RAM) allocated for storage of such variables. As this amount is limited by the virtual machine or operating system running the program, it needs to be appropriately utilized, especially for storing large pieces of data such as matrices. Table 4.1 summarizes the space complexity of some of the variables that use up large portions of the allocated RAM. The third column illustrates how the expressions in column 2 are used for a 10x10x10 portal frame with $11*11*11*6=7986$ degrees of freedom, 3410 members, $121*6=726$ restrained DOFs (*cDOFs*) and $7986-726=7260$ unrestrained DOFs (*uDOFs*).

Matrix	Memory Used (Bytes)*	EXAMPLE
Structure Stiffness Matrix	$8 * DOF * DOF$	$8 * 7986 * 7986 = 486MB^{**}$
Reduced Stiffness Matrix	$8 * uDOF * uDOF$	$8 * 7260 * 7260 = 402MB$
Reaction Stiffness Matrix	$8 * cDOF * uDOF$	$8 * 7260 * 726 = 40.2MB$
Element Stiffness Matrix	$8 * 12 * 12 * n$	$8 * 12 * 12 * 3410 = 3.74MB$
Element Rotation Matrix	$8 * 12 * 12 * n$	$8 * 12 * 12 * 3410 = 3.74MB$
TOTAL		935.68MB
*1 Real value with double precision= 8 Bytes		
**1MB=1024*1024 Bytes		

Table 4.1: Memory Consumption by Matrices

4.2.2 Improvements to the Stiffness Solver

As shown in the previous section, manipulation of matrices involved in structural analysis hold up a lot of memory and take exponential amounts of time to complete processing. However, a large percentage of this problem can be reduced by studying each matrix and applying the appropriate mechanisms to minimize both space and time utilization.

For example, the primarily important structural stiffness matrix has common properties for most types of topologies. This can be taken advantage of, especially if a fast structural analysis program is required.

First, the SSM can be classified as very sparse, that is, in most cases, more than 95% of its entries are zeros. This implies that vast amounts of memory and time is wasted manipulating zeros. Hence, a more effective means of data storage is needed. Past efforts in this topic have yielded several efficient formats for the storage of sparse matrices. These include compressed matrix storage formats like Compressed Row Storage(CRS), Compressed Column Storage(CCS), Coordinate Column/Row Storage(COOC/R) and their variants. Hence, one of these formats could be selected as a replacement to standard

matrices. In this study, a relatively new storage format called the sparse array[14] has been adopted to store large matrices, preferred for its almost excessive versatility and efficiency in manipulating sparse matrices.

Second, the SSM(and its reduced form after boundary conditions have been applied) is Symmetric Positive Definite (SPD). This allows for iterative methods such as the Preconditioned Conjugate Gradient(PCG) method to be applied directly for solving the linear equation system with the nodal displacements as unknowns. Because the SSM is diagonal in shape, diagonal preconditioning has been selected for this algorithm.

The same schemes discussed in the previous paragraphs have been used for other matrix based computations present in the study.

4.2.3 Computation of Design Actions

Once structural analysis has been completed, the next step is to determine the magnitudes of design actions to be used for carrying out sizing and reinforcement computations. For general load cases, determination of these actions requires locating all maxima and minima internal action magnitudes that occur on the member at one or several locations. This can be done by calculating the member end actions for each member, and determining internal action coordinates through section-based equilibrium at predefined intervals along the member. Then, gradients/differentials could be determined by using:

$$\nabla_i = \frac{(F_{i+1} - F_i)}{\Delta x} \quad (4.1)$$

where ∇_i is the gradient, F_i and F_{i+1} represent consecutive internal action coordinates, and Δx is the length interval.

By detecting a change of signs in the gradient, one can locate all maximum and/or minimum point locations on the member. This step needs to be executed 6 times for all 6 types of internal actions available, namely: axial force A_{X-X} , major shear V_{Y-Y} , minor shear V_{Z-Z} , torsional moments T_{X-X} , major bending moment M_{Z-Z} and minor bending moment M_{Y-Y} .

A relatively special case for determining design actions would be axial force and bending moment combinations. Since the location of the critical A_{X-X} - M_{Z-Z} - M_{Y-Y} is unknown, the program is provided with guided search tools that allow it to select the most critical combination by applying these protocols:

1. If M_{Y-Y} or M_{Z-Z} is too small, then a combination of M_{Y-Y} or M_{Z-Z} with A_{X-X} can be used to check for the uniaxial interaction resistance of the structural element.
2. If A_{X-X} is too small, then bending moments M_{Y-Y} and M_{Z-Z} are checked for bending moment resistance only.

3. If both M_{Y-Y} and M_{Z-Z} are too small in magnitude, then the element is treated as an axial member and is treated as such in the checking process.
4. If none of the conditions apply, a guided search is done to find the critical A_{X-X} - M_{Z-Z} - M_{Y-Y} section. The guided search algorithm starts from the largest combinations available and checks if other more critical combinations exist by randomly taking a section and constructing a biaxial diagram for the formed eccentricity.

4.3 Design and Constraint Algorithms

4.3.1 Assumptions

The next few subsections briefly explain how design procedures according to Eurocode-2 were implemented in the program, and are divided based on the actions considered. The design algorithms were constructed so as to handle standard concrete frame elements, in this case, beams and columns. Additionally, sizing constraints imposed by Eurocodes 2-1 and 2-2 have also been dealt with.

Design actions to be checked for, are obtained from searching for the maximum and minimum values of each internal action diagram (See previous section). These values are then used for differentiating regions with varying quantity or positioning in reinforcement.

4.3.2 Design Checks for Bending(Alg. 4.14, 4.15)

Based on the procedures described in [3], sections subjected to combined axial and bending moments need to be designed for the combined stress interactions developed. While other approximate methods exist for such designs, in this work, the direct procedure of constructing interaction diagrams has been used to check the available capacity for the sections under stress. Based on the existent conditions, interaction checks are made either in individual directions, initiating uniaxial interaction functions, or in cases where moments in both directions are comparable, initiating biaxial check functions. This classification is based on the criteria:

$$\frac{e_y}{h} / \frac{e_z}{b} \leq 0.2 \quad (4.2)$$

or its inverse. Any member that doesn't satisfy this condition is treated to behave in a biaxial manner. This case can be handled by constructing biaxial interaction diagrams for the current orientation θ that the design moment in the major direction makes with the minor one. A separate program has not been made for doing uniaxial interaction checks, as applying specific orientations of $\theta = 0^\circ$ or $\theta = 90^\circ$ in the biaxial interaction routines, for the major and minor directions respectively, yields the desired result.

Condition for Penalty	Penalty Magnitude
$\epsilon_{biaxial}=0.0$	1.0
$0.0<\epsilon_{biaxial}\leq 1.0$	$1-\epsilon_{biaxial}$
$\epsilon_{biaxial}>1.0$	p_{max}

Table 4.2: Penalty Values for Biaxial Bending

Biaxial interaction diagrams have been constructed by directly manipulating the neutral axis depth, and collecting the resulting internal moment and axial loading capacity coordinates. To speed up relevant computations, stress magnitude in concrete is assumed to have a rectangular distribution which spans 80% of the length from the top of the section to the current neutral axis.

Once the coordinates for the interaction diagrams have been computed, additional work needs to be done to check for capacity. First, the algorithm needs to check if the internal bending moment and axial load combinations reside within the interaction boundary. Second, it is also required to check how far the loading coordinate is from the boundary. The farther it is from the boundary, the more the reserve capacity the section has, making the assigned section conservative. The fitness and penalty for the member is also computed based on this efficiency value.

If the internal action coordinates reside outside of the interaction boundary, then the assigned section is inadequate. Penalty is thus applied based on this deviation. The algorithm for the full procedure is as given in Algorithm 4.15.

The function *insideBoundary()* takes two arguments: the design action coordinates and the interaction coordinate list, and checks in which sub-region the design point lies. If it doesn't reside in any of the regions, then by definition, the point should be outside of the interaction boundary. If it does reside in one of the regions, the index for that region gets returned. The function *computeEfficiency()* takes the index of the region returned by *insideBoundary()* and constructs a relative scale through which the efficiency can be determined. (See Figure 4.1.)

Penalty is applied to the cost function whenever a structural member is found to have failed this biaxial design check. The magnitude of the penalty is a constant value for all members. Penalty values are also applied based on how efficient the section is. That is, the less efficient the section is, the more the magnitude of penalty would be.

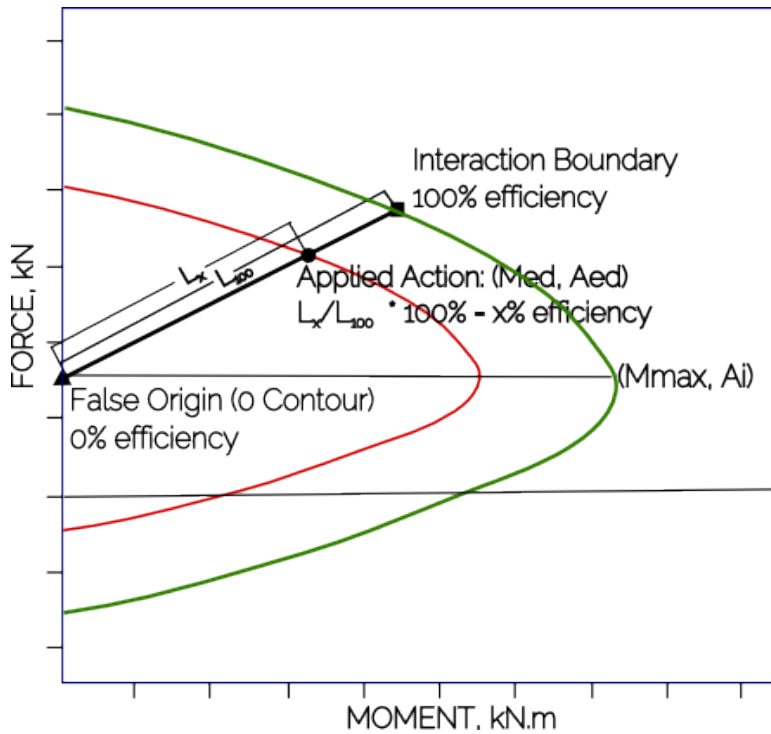


Figure 4.1: Efficiency Computation for Biaxial Bending Interactions

4.3.3 Design Checks for Shear(Alg. 4.18)

The design for pure shear in both the major and minor directions follows the procedure outlined on [4, 22], which involves computing the resistance provided by concrete through strut action. Because the shear link spacings for each member have already been randomly assigned by the genetic algorithm, the procedure is supposed to check the adequacy of these spacing values. In order to do this, the spacing of shear links required to resist the design shear needs to be computed. The design shear forces can be found from the shear force diagram of that member and the values are taken as the magnitudes at the intersection between elastic centroidal axis of the member and a 45° line emanating from the supports. It should be noted that this method of finding design shear forces is only applicable when load cases are typical. For atypical cases, the gradient method outlined in section 4.2.3 could be used.

Finally, the penalty factor for shear capacity is calculated for both minor and major directions by using the required and the provided shear link spacing values.

4.3.4 Design Checks for Torsion (Alg. 4.19, 4.20)

The designer's guide for Eurocode-2[3] subdivides torsional action into two categories: equilibrium torsion, which exists to provide stability to a structural element (such as in girders and curved elements), and compatibility torsion, usually encountered to satisfy compatibility conditions for deformation. Examples include torsional action developed

Member	Design Action	Maximum	Minimum
Columns	Flexure	$\left\{ \begin{array}{c} 0.04 * b * h - A_s \\ or \\ 0.08 * b * h - A_s^+ \end{array} \right\}$	$Max \left\{ \begin{array}{c} 0.1N_{ED}/(f_{yk}/1.15) \\ (0.002 * 0.04 * b * h - A_s) \end{array} \right\}$
	Shear	$S_{max} = 0.75 * d$	–
Beams	Flexure	$0.04 * b * h - A_s$	$0.0013 * b * d$
	Shear	$S_{max} = 0.75 * d$	$S_{min} = (A_{sv} * f_{yk}) / (0.08 * b * \sqrt{f_{ck}})$
+Near Lapped Regions			

Table 4.3: Maximum and Minimum Reinforcement Amounts for Structural Members

in edge beams attached to slabs. For regular structural frames, the dominant action encountered would be compatibility torsion.

The design actions for torsional capacity checks are obtained by taking the maximum torsional moment value from the torsional moment diagram of the member. The procedure for checking torsional capacity is similar as that for pure shear. Additionally, structural members subjected to both shear and torsion require a simple interaction check, i.e.

$$\frac{V_{sd,i}}{V_{RD,max}} + \frac{T_{sd,i}}{T_{RD,max}} \leq 1 \quad (4.3)$$

The computation of efficiency and boundary checks follow the same mechanisms as discussed in biaxial interaction checks. The effect of torsional moments on a member needs to be reflected on both the longitudinal and shear reinforcements. Both these reinforcements can carry the inclined tensile stresses developed by acting as tension struts where as the concrete serves as a compressive member within the truss. Failure occurs when longitudinal reinforcements start yielding and the concrete crushes along the crack axes. Once the torsional tensile stress exceeds the cracking stress magnitude, tensile reinforcement in the form of shear links(stirrups) needs to be provided to resist the full torsional moment magnitude[22].

4.3.5 Sizing Constraints

In the design of concrete structures, Eurocode-2 dictates the following additional requirements to be fulfilled:

- 1. Maximum and Minimum Reinforcement Quantities:** For the purpose of resisting temperature stresses and limiting crack width, Eurocode-2-1[5](Section 9.2) recommends a small amount of reinforcement be placed in members that do not require strength reinforcement. Maximum reinforcement limits are provided such that the ease of constructibility in placing reinforcement bars and pouring concrete is ensured.

2. Breadth-to-Height Ratio: Since the algorithm chooses section sizes randomly, certain limits need to be placed on these sizes, so as to avoid deep/narrow or thin/wide members. In this study, a b/h ratio of 0.5 has been taken as a limiting criteria.

3. Minimum Width: To provide ample resistance to fire (Eurocode 2-2[6], section 5.6.3, Table 5.6), all beams shall have a minimum width of 200mm by assuming all calculations to be done for a standard fire resistance duration of 60 minutes and an axis distance of 12mm. Columns shall have a minimum width of 200mm with a concrete cover of 25mm, again by assuming a 60 minute fire resistance duration, and with the column being exposed on more than one side. (Eurocode 2-2[6], section 5.3.2, Table 5.2a)

4.4 Optimization Formulation

Let F_i be a 3D frame composed of H horizontal members and V vertical members and having l regions of critical internal action. A general formulation for its cost optimization would be defined as:

Decision Variables: $\{A_{sv,k}A_{s,l,k}, b_k, h_k\}$ where $k \in (H \cup V)$, $l \in \{X_{f,k}\}$

Minimize $O = \sum_{r=k} \{\sum C_{conc} + C_{steel} + C_{formwork}\}$

Subject to:

$$\begin{cases} M_{ED,k} \leq M_{rd,k} & k \in H \\ V_{ED,k} \leq V_{rd,k} & k \in H \\ \frac{N_k}{N_{u,k}} + \frac{M_k}{M_{yu,k}} + \frac{M_{zk}}{M_{zu,k}} \leq 1 & k \in (H \cup V) \\ A_{sk} \geq A_{sk,min} & k \in (H \cup V) \\ A_{sk} \leq A_{sk,max} & k \in (H \cup V) \\ \frac{V_{ED,k}}{V_{RD,max,k}} + \frac{T_{ED,k}}{T_{RD,max,k}} \leq 1 & k \in (H \cup V) \\ T_{ED,k} \leq T_{rd,s,k} & k \in H \end{cases} \quad (4.4)$$

Where individual costs for volume of concrete and reinforcing steel as well as the area for form-work and scaffolding for beams could be determined, respectively, as:

$$C_{conc,k} = c_{conc} \gamma_{conc} b_k h_k l_k \quad (4.5)$$

$$C_{steel,k} = c_{steel} * \left(\sum \gamma_{steel} A_{s,l,k} l_{l,k} + \sum \gamma_{steel} A_{sv,k} l_{l,k} \right) \quad (4.6)$$

$$C_{formwork,k} = \begin{cases} c_{form} * (2 * h_k + 2 * b_k) * l_k & k \in V \\ c_{scaffold} * (b_k * l_k) + c_{form} * (2 * h_k + b_k) * l_k & k \in H \end{cases} \quad (4.7)$$

Because genetic algorithms are to be used for the structural optimization, this standard formulation needs to be reconfigured to achieve fitness evaluation expressions that include both the effect of the objective function as well as the constraints.

4.5 Genetic Modeling

Genetic algorithms operate on collections of randomly generated strings of data, portions of which represent the design variables. These strings, called chromosomes, should therefore be decoded in order for the data stored to be utilized. In this study, a single chromosome is used to represent all geometric and reinforcement data contained within a single structure. This includes the sizes (widths and heights) of each member in the

structure, and flexural and shear reinforcements for several moment and shear critical regions of all members. Every index in a chromosome thus, represents a real number value for one of this variables, and its corresponding value is obtained by doing a simple table lookup. Once all these indices have been decoded and their corresponding values assigned to each respective member, the resulting structure formed can be analyzed and evaluated for all constraints listed in section 4.3.

4.5.1 Member Sizing

Sizes for structural member are encoded on a simple discrete scale as shown in the table below. Each integer represents a single width/height value in the individual chromosome. One observation that can be made from table 4.4 is that both width and height variables could assume any randomly generated key. Thus, it is possible to obtain width-height combinations that would result in wide, deep or narrow sections. This can be avoided by utilizing the sizing constraints discussed in section 4.3.5. These constraints impose large magnitudes of penalty on individual chromosomes having members that exhibit size misconfiguration. Thus, after a few iterations in the simulation, these individuals will be eliminated from the population as a result of their fitness value.

Key	Value(mm)	Key	Value(mm)	Key	Value(mm)
0	200	5	325	10	450
1	225	6	350	11	475
2	250	7	375	12	500
3	275	8	400	13	525
4	300	9	425	14	550

Table 4.4: Value Encoding for the Sizing of Members

4.5.2 Reinforcement Encoding

Each member has two basic types of flexural reinforcements assigned to it: continuous reinforcements located at each corner of the member, and additional reinforcements for regions of critical negative and positive bending. Continuous reinforcements have a simple indexing scheme based on all available diameters of reinforcing bars.

Top and Bottom Reinforcements			
Key	No.of Bars	ϕ , <i>mm</i>	Area, mm^2
0	2	12	226.1946711
1	2	14	307.8760801
2	2	16	402.1238597
3	2	20	628.3185307
4	2	24	904.7786842
5	2	32	1413.716694

Table 4.5: Continuous Reinforcements

Additional reinforcements are provided based on available diameters of reinforcement and number of bars. Four values are used in the encoding scheme: two indices for positive reinforcements in the Y- and Z- directions, and the remaining two assigned for negative reinforcements. Areas of reinforcements for positive and negative bending regions in a section could then be computed by adding up the reinforcement areas for the continuous and additional reinforcements. A maximum number of 3 bars has been encoded for all diameter types. If more than three bars is desired, the source code could be modified easily to accomodate this.

Additional Negative and Positive Reinforcements							
Key	No.of Bars	ϕ , <i>mm</i>	Area, mm^2	Key	No.of Bars	ϕ , <i>mm</i>	Area, mm^2
0	1	12	113.0973355	9	1	20	314.1592654
1	2	12	226.1946711	10	2	20	628.3185307
2	3	12	339.2920066	11	3	20	942.4777961
3	1	14	153.9380400	12	1	24	452.3893421
4	2	14	307.8760801	13	2	24	904.7786842
5	3	14	461.8141201	14	3	24	1357.168026
6	1	16	201.0619298	15	1	32	706.8583471
7	2	16	402.1238597	16	2	32	1413.716694
8	3	16	603.1857895	17	3	32	2120.575041

Table 4.6: Extra Reinforcement Encoding

Shear reinforcements for both beams and columns is provided in the form on shear links. For convenience, 8mm bars are used with varying spacings for each member. Indices from chromosomes represent discrete spacing values. As varied spacing of links is more critical in beams than in columns, two potential spacing values (excluding the minimum and maximum spacing) are used.

Shear Link Spacing			
Key	Value(mm)	Key	Value(mm)
0	100	6	250
1	120	7	270
2	150	8	300
3	170	9	320
4	200	10	350
5	220		

Table 4.7: Shear Link Spacing Indices

4.6 Genetic Optimization

4.6.1 Cost Estimation and Penalty (Alg. 4.26)

The performance of a structural design is estimated based on two criteria: its structural performance under loading and its cost. The structural performance could be measured by assessing flexural and shear capacities of each member based on the current loading conditions. This would also include determining permissible envelopes for internal actions. Details of determining the structural performance are given in section 4.3. Cost values could be estimated by taking appropriate cost indices for concrete(c_{conc}), reinforcing steel(c_{steel}), formwork and scaffolds(c_{form} , $c_{scaffold}$) and computing the total reinforcement and concrete volumes. The total cost could be determined by using equation 4.4. In order to evaluate a design from both these aspects, a combined expression for incorporating these values is needed. This expression is called the fitness function, and its evaluated value is the primary criteria through which the evolutionary selection process is carried out with.

4.6.2 The Fitness Function (Alg. 4.27)

The fitness function evaluates the total performance of the structure and lets the genetic simulation algorithms use its result as a means to identify whether that particular design is a good fit or not. The general expression is:

$$F = (1 + p) * C \quad (4.8)$$

where F is the fitness value, p is the penalty and C is the total cost of the structure. It can be seen from equation 4.8 that, low values of F indicate that:

1. The penalty value p is small, or
2. The total cost C is small, or

3. Both (1) and (2) are true, in which case, the design is optimal.

Low p values indicate that the structure is efficient in terms of structural capacity (but not necessarily cheap), while low C values indicate that the design is cheap (but doesn't necessarily satisfy strength/capacity/sizing constraints). Thus, the fitness function acts as a weighted visualizer of the design's cost and performance feasibility, whereas the penalty and cost parameters only illuminate part of this evaluation. Table 4.8 illustrates different values for these parameters and how to assess their value.

Case	Cost	Penalty	Fitness	Remark	Conclusion
1	10000	5.0	60000.0	High Pen., Med. Cost, High Fitness	Not Optimal
2	10000	1.0	20000.0	Low Pen., Med. Cost, Low Fitness	Near Optimal
3	2000	11.0	24000.0	High Pen., Low Cost, High Fitness	Not Optimal
4	30000	1.0	60000.0	Low Pen., High Cost, High Fitness	Not Optimal

Table 4.8: Examples of Fitness Evaluation

4.6.3 Simulation(Alg. 4.21)

The simulation process of a simple genetic algorithm is described in algorithm 4.21. The routine takes a population size and the number of iterations as input. Based on this, it randomly creates a population of individuals with value encoded strings. For each individual, this encoded information is decrypted and entered into the structure's database. Based on the decoded information, the structure is analyzed, and evaluated for its cost and its structural performance. Finally, the fitness function is evaluated for it. Once all individuals in the population have gone through this process, the genetic algorithm takes over to apply selection, mutation, and crossover on the individuals. This procedure is repeated for the prescribed number of iterations. Convergence is achieved when the variance between successive iterations vanishes or when a user-specified criteria is satisfied.

4.7 HELIX: A Structural Optimization Platform

The following sections illustrate the main features of the developed program, named HELIX. HELIX was written in Java 8 and at the time of this writing is about 138MB in size (including external libraries and graphics resources). It is a single-threaded application that can run on any x86/x64 based Intel or AMD processor. HELIX can be used as a structural analysis software as well as for structural optimization problems.

4.7.1 Why Java?

Java is an object-oriented programming language used universally in a multitude of sectors from web-scripting to operating system development, artificial intelligence research, infrastructure control systems and consumer application production. Currently, Java runs on more than 3.5 billion devices including mainframes, smart phones, computers, cars, home appliances, communication platforms and several other electronic equipments.

Unlike other programming languages which run on the environment of the operating system they are installed on, Java runs exclusively on a virtual machine with preallocated resources known as the Java Virtual Machine, which is instantiated alongside the operating system when a program runs. This feature allows a program written in Java to compile/run on any computer/OS capable of supporting the JVM. Another feature of the JVM would be customizable resource allocation(i.e. total amount of RAM and bandwidth allocated by the OS can be modified by the user/programmer). In addition, Java comes with a plethora of external and stock libraries for problems ranging from UI design to numerical computations. Perhaps the most alluring feature of Java, in addition to the aforementioned features, would be its object-oriented paradigm. This allows for either a top-down or a bottom-up design to be feasible.

This research work has chosen Java for its almost excessive support for matrix manipulations, for its ease of integration with other applications, its speed and customizability in using hardware resources, and for its unparalleled integration with XML for writing Graphic User Interfaces(GUIs).

4.7.2 Main Interface and Grid Definition

The primary window is the one that is encountered when the program opens. At the start of the program, the ribbon shown at the left is the grid definition menu. Depending on the topology, empty two and three dimensional grids, can be defined here. In addition, predefined portal frames could also be drawn on the canvas.

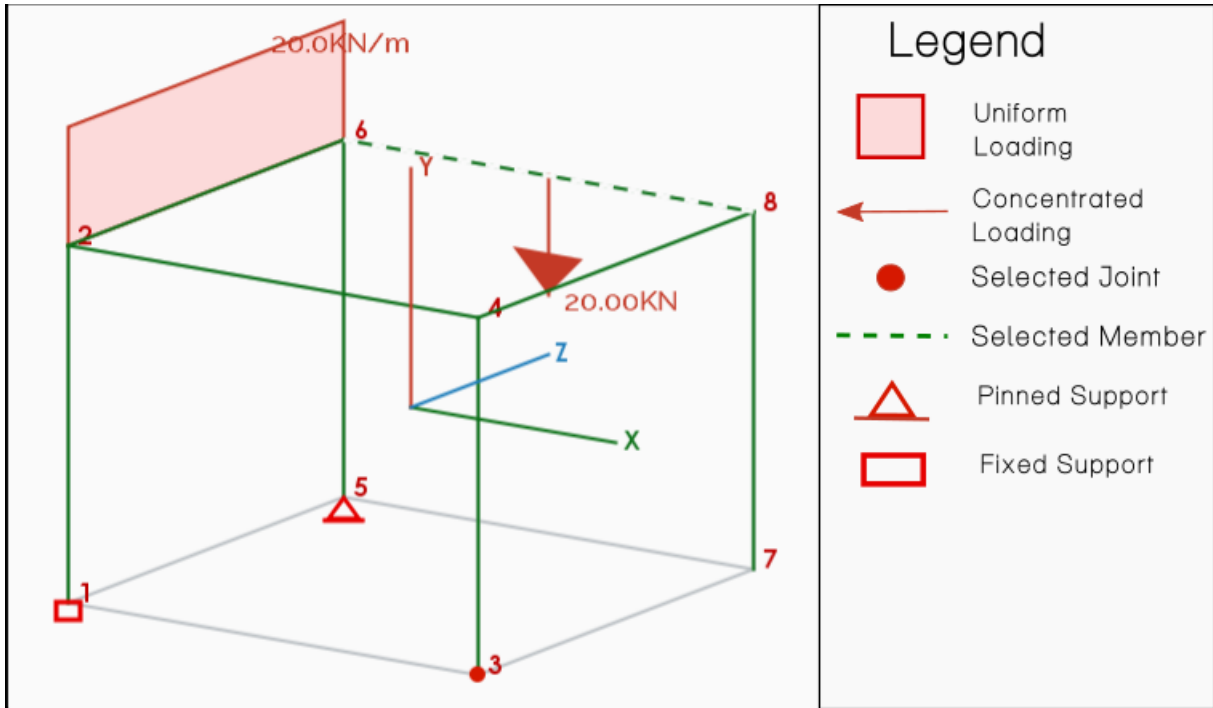


Figure 4.2: Main Canvas Components

4.7.3 Ribbon Functions

Through the ribbon, one could access the majority of functionalities featured in the program. Table 4.9 summarizes these functions.











Ribbon	Functions
 Draw Member	Sets the display canvas to drawing mode. In this mode, members could be quickly drawn by connecting points on the grid using the left mouse button.
 Select Joints	Sets the display canvas to joint selection mode. Joints can then be selected individually by placing the cursor near a labeled joint and clicking the left mouse button once. Multiple joints could also be selected by clicking the left mouse button and dragging through the area of interest.
 Select Members	Sets the display canvas to member selection mode. A member can be easily selected by moving the cursor near it, and single clicking using the left mouse button. Multiple members could also be selected by cross- dragging along the members using the left mouse button.
 Material Def.	Displays the material definition menu. Constitutive property data for concrete and reinforcing steel is to be entered here.
 Restraints	Displays the joint restraint options menu. Selected joints can be assigned appropriate restraints for any degree of freedom using this menu.
 Loads	Displays the load definition menu. Load data, such as direction of application, type(distributed or concentrated) magnitude and location is to be specified here.
 Section Def.	Displays the section definition menu. Cross-sectional data such as width and height of members
 Analyze	Performs structural analysis, if all relevant data has been entered, and topology has been defined appropriately.
 Design Member	Computes reinforcement quantities based on previously assigned section information and analysis results.
 Optimization	Opens the genetic optimization menu. In this dialog, one can enter simulation parameters such as iteration number and population size. The simulation is also to be started from this menu.

Table 4.9: Main Ribbon Functions

4.7.4 Sub-menus

There are six sub-menus that could be accessed from the main ribbon. These are:

1. **Grid Definition** - the user can enter parameters for defining an empty rectangular grid with regular spacing in this sub-menu. These include basic dimensions such as story height and bay width in the horizontal directions. The number of grid lines in each direction can also be specified. If two dimensional frames/grids are required, then one could set the number of grid lines in the unwanted direction to zero. If a regular portal frame is what's required, then manual drawing of all members can be avoided by selecting the "Generate Frame Elements" check-box. This option generates a portal frame aligned with the grid, and instantiates all necessary topological data. This is useful especially if large portal frames are to be drawn quickly.
2. **Material Definition**- Concrete and reinforcing steel material properties are to be entered here. This data is later utilized in forming the element and structure stiffness matrices. If these values are not known, or haven't been decided upon, then by checking the "Defaults" option, predetermined data could be set.
3. **Loads**- Before load data can be entered, members to which the load is to be applied need to be selected. One can select members by clicking on the "Select Members" ribbon. Once selection is complete, load data could be entered. Through the sub-menu, concentrated force and moment magnitudes can be applied by entering the magnitude and relative position of the application point. Distributed load data can be input by selecting the "Distributed" check-box.
4. **Restraints**- Joints can be restrained against the six possible degrees of freedom. Similar to the load sub-menu, the joints sub-menu needs at least one selected joint as input.
5. **Optimization**- Through the optimization sub-menu, parameters for simulating a genetic selection scheme can be specified. This includes assigning mutation, crossover probabilities and population size. Once these values are entered, the "Accept" button can be pressed to start the iteration. The "Optimize as Building" option allows for the structure to be treated as a regular building with uniaxial actions and reinforcement profiles. If selected, secondary actions in the minor direction would also be ignored in the design check procedure.

The image displays three sub-menus from a software application, arranged horizontally. Each sub-menu has a title at the top and various input fields and controls below.

- Grid Definition...:** Contains five input fields for 'Number of Stories', 'Story Height, m', 'Number of Bays, X', 'Bay Width, X, m', and 'Number of Bays, Z'. Below these is another input field for 'Bay Width, Z, m' and a checkbox labeled 'Generate Frame Elements'. At the bottom are 'Generate' and 'Cancel' buttons.
- Materials...:** Starts with a checked checkbox 'Defaults'. It is divided into two sections: 'Concrete' and 'Steel'. The 'Concrete' section has input fields for 'Conc. Char. Strength(fck), MPa' (value: 20) and 'Young's Modulus, Ec, GPa' (value: 29). The 'Steel' section has input fields for 'Steel Yield Strength(fyk), MPa' (value: 300) and 'Young's Modulus, Es, GPa' (value: 200). At the bottom are 'Accept' and 'Cancel' buttons.
- Loads...:** Starts with an unchecked checkbox 'DISTRIBUTED'. It contains six pairs of input fields: 'Force, X, KN' and 'Rel Position, X'; 'Force, Y, KN' and 'Rel. Position, Y'; 'Force, Z, KN' and 'Rel. Position, Z'; 'Moment, X, KN.m.' and 'Rel Position, X'; 'Moment, Y, KN.m.' and 'Rel. Position, Y'; and 'Moment, Z, KN.m.' and 'Rel. Position, Z'. At the bottom are 'Accept' and 'Cancel' buttons.

Figure 4.3: Sub-Menus 1

The image shows two side-by-side sub-menus. The left sub-menu is titled "Optimization" and contains several input fields: "Population Size", "Number of Iterations", "Mutation Probability", "Crossover Probability", "Biaxial Weight, Kw", and "Shear Weight, Kv". Each field has a horizontal line below it. At the bottom of this sub-menu is a checkbox labeled "OPTIMIZE AS BUILDING" and two buttons: "Simulate" and "Cancel". The right sub-menu is titled "Restraints" and is divided into two sections: "Displacements" and "Rotations". Under "Displacements", there are three checkboxes labeled "x", "y", and "z". Under "Rotations", there are three checkboxes labeled "x", "y", and "z". At the bottom of the "Restraints" sub-menu are two buttons: "Accept" and "Cancel".

Optimization

Population Size

Number of Iterations

Mutation Probability

Crossover Probability

Biaxial Weight, Kw

Shear Weight, Kv

OPTIMIZE AS BUILDING

Simulate Cancel

Restraints

Displacements

x

y

z

Rotations

x

y

z

Accept Cancel

Figure 4.4: Sub-Menus 2

4.7.5 Input and Output Data

Based on the provided input, three tasks could be accomplished using the software: Structural analysis, structural design and optimization. In order for these procedures to be carried out, however, specific groups of information need to be given beforehand. Table 4.10 summarizes all required inputs and generated outputs of the software.

Data	Analysis	Design	Optimization
Input			
Material Properties	X	X	X
Section Definition	X	X	
Initial Reinforcement Quantity		X	
Optimization Parameters	X	X	X
Material/Equipment Cost			X
Output			
Support Reactions	X	X	X
Internal Actions	X	X	X
Design Section			X
Design Reinforcement		X	X
Capacity Information			X
Cost Information		X	X

Table 4.10: Input and Output Data for Analysis, Design and Optimization

4.7.6 Evolutionary Simulation

Genetic simulation can be initiated by supplying the necessary optimization information. After simulation has been initiated, depending on the size of the problem, a progress window might appear showing all necessary simulation and performance information in real-time. The required time depends upon several factors, such as, the number of iterations, the size of the structure(topologically) and the size of the population. When the simulation ends, plots showing the progress of convergence and the descents of penalty and fitness values could be viewed. The plots could also be saved as images by accessing the right-click properties menu on the graph.

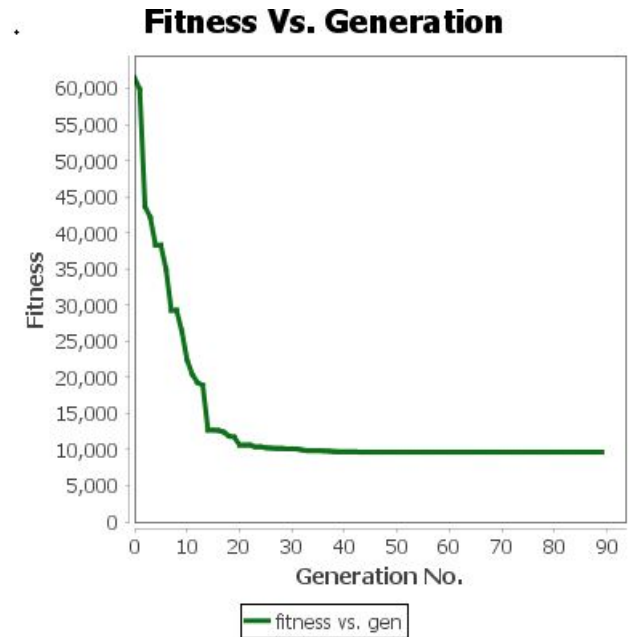


Figure 4.5: Part of the Plot Dialog

4.7.7 The Result and Section Dialogs

These dialog boxes are displayed, if and only if, a successful design, genetic simulation or structural analysis has been performed. Through these dialogs output data such as internal action magnitudes, design reinforcement areas, sizing information and structural efficiency can be viewed. Additionally, all information can also be exported in a tabular form by using .TSV file format, which can be readily opened by any spreadsheet application like Microsoft Excel.

4.8 Algorithm Listings

In this section, algorithms for the topics discussed in chapter 4 have been concisely presented. While the algorithms were made written in the hope of having a clear and concise description, some details haven't been included for the sake of brevity. The reader is advised to read the preceding sections before going through this section.

4.8.1 Structural Analysis

Algorithm 4.1 Summary of the Stiffness Method

Function Analyze(S)

Material Properties : $\{E_i\}$

Joint Vector : J_i

Member Vector : M_i

Load Vector : L_i

Structure : S

Total Degrees of Freedom DOF : $n = \text{Length}(J_i) * 6$

Restrained DOF: $nr = \text{RestrainedDOF}(J_i)$

MemberParameters($M_i, \{E_i\}$)

AssembleStiffness(S)

ApplyBoundaryConditions(S)

AssembleReactionStiffness(S)

ComputeDisplacements(S)

ComputeReactions(S)

MemberEndActions(S)

for each action R in $\{A_{X-X}, V_{Y-Y}, V_{Z-Z}, T_{X-X}, M_{Z-Z}, M_{Y-Y}\}$ do

 DesignActions(R, S)

end for

End Function

Algorithm 4.2 Computation of Member Parameters

```
function MemberParameters( $M_i$  ,  $\{E_i\}$ )  
  
  for each member M in  $M_i$  do  
    h=height(m)  
    b=breadth(m)  
    l=length(m)  
     $l_x = x_{end} - x_{start}$  ,  $l_y = y_{end} - y_{start}$  ,  $l_z = z_{end} - z_{start}$   
    Cross-sectional Area:  $A = \text{height}(m) * \text{breadth}(m)$   
    Moment of Inertia:  
       $I_{zz} = \frac{b * h^3}{12}$   
       $I_{yy} = \frac{h * b^3}{12}$   
       $I_{xx} = (1/3) - (0.21 * (\frac{b}{h}) * (1 - (\frac{b^4}{12 * h^4}))) * h * b^3$   
    Direction Cosines:  
       $c_x = l_x / l$  ,  $c_y = l_y / l$  ,  $c_z = l_z / l$   
    Rotation Matrix:  
       $R = \text{RotationMatrix}(c_x, c_y, c_z)$   
    Local Stiffness Matrix:  
       $K_l = \text{LocalStiffness}(M, E, J)$   
    Global Stiffness Matrix:  
       $K_g = \text{GlobalStiffness}(M)$   
    AML(M)  
  end for  
end function
```

Algorithm 4.3 Computation of Local Stiffness Matrix

```

function LocalStiffness(Member M ,Constitutive Prop. {Ei})
k1 =  $\frac{A*E}{L}$ , k2 =  $\frac{12*E*I_z}{L^3}$ , k3 =  $\frac{12*E*I_y}{L^3}$ , k4 =  $\frac{6*E*I_z}{L^2}$ , k5 =  $\frac{6*E*I_y}{L^2}$ , k6 =
 $\frac{4*E*I_z}{L}$ , k7 =  $\frac{4*E*I_y}{L}$ , k8 =  $\frac{2*E*I_z}{L}$ , k9 =  $\frac{2*E*I_y}{L}$ , k10 =  $\frac{G*I_x}{L}$ .
Sparse Array: Kl=

```

$$\left(\begin{array}{cccccccccccc} k_1 & 0 & 0 & 0 & 0 & 0 & -k_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 & 0 & k_4 & 0 & -k_2 & 0 & 0 & 0 & k_4 \\ 0 & 0 & k_3 & 0 & -k_5 & 0 & 0 & 0 & -k_3 & 0 & -k_5 & 0 \\ 0 & 0 & 0 & k_{10} & 0 & 0 & 0 & 0 & 0 & -k_{10} & 0 & 0 \\ 0 & 0 & -k_5 & 0 & k_7 & 0 & 0 & 0 & k_5 & 0 & k_9 & 0 \\ 0 & k_4 & 0 & 0 & 0 & k_6 & 0 & -k_4 & 0 & 0 & 0 & k_8 \\ -k_1 & 0 & 0 & 0 & 0 & 0 & k_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -k_2 & 0 & 0 & 0 & -k_4 & 0 & k_2 & 0 & 0 & 0 & -k_4 \\ 0 & 0 & -k_3 & 0 & k_5 & 0 & 0 & 0 & k_3 & 0 & k_5 & 0 \\ 0 & 0 & 0 & -k_{10} & 0 & 0 & 0 & 0 & 0 & k_{10} & 0 & 0 \\ 0 & 0 & -k_5 & 0 & k_9 & 0 & 0 & 0 & k_5 & 0 & k_7 & 0 \\ 0 & k_4 & 0 & 0 & 0 & k_8 & 0 & -k_4 & 0 & 0 & 0 & k_6 \end{array} \right)$$

```

end function

```

Algorithm 4.4 Member Rotation Matrix

```

function RotationMatrix(cx, cy, cz)
Direction Cosines: cx, cy, cz
Member Roll Angle: α

```

$$c_{xz} = \sqrt{c_x^2 + c_z^2}, \quad \cos A = \cos(\alpha), \quad \sin A = \sin(\alpha)$$

$$r_1 = \frac{(-c_x*c_y*\cos A - c_z*\sin A)}{c_{xz}}, \quad r_2 = c_{xz} * \cos A$$

$$r_3 = \frac{(-c_y*c_z*\cos A + c_x*\sin A)}{c_{xz}}, \quad r_4 = \frac{(c_x*c_y*\sin A - c_z*\cos A)}{c_{xz}}$$

$$r_5 = -c_{xz} * \sin A, \quad r_6 = \frac{(c_y*c_z*\sin A - c_x*\cos A)}{c_{xz}}$$

```

Rotation Matrix: R=

```

$$\left(\begin{array}{cccccccccccc} c_x & c_y & c_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_1 & r_2 & r_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_4 & r_5 & r_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_x & c_y & c_z & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_1 & r_2 & r_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_4 & r_5 & r_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & c_x & c_y & c_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r_1 & r_2 & r_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r_4 & r_5 & r_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_x & c_y & c_z \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_1 & r_2 & r_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_4 & r_5 & r_6 \end{array} \right)$$

```

end function

```

Algorithm 4.5 Computation of Member Global Stiffness Matrix

```
function GlobalStiffness(Member  $M$ )  
  
    Local Stiffness Matrix:  $K_l$   
    Rotation Matrix:  $R$   
    Global Stiffness Matrix:  $K_g = R^T * K_l * R$   
  
end function
```

Algorithm 4.6 Computation of Member End Actions

```
function MemberEndActions(Member  $M$ )  
  
    Member Displacement Vector:  $Disp_m =$   
     $\{dx_{st}, dy_{st}, dz_{st}, rx_{st}, ry_{st}, rz_{st}, dx_{end}, dy_{end}, dz_{end}, rx_{end}, ry_{end}, rz_{end}\}^T$   
    Member End Actions:  $F_{end} = A_{ml} + K_l * R * Disp_m$   
  
end function
```

Algorithm 4.7 Computation of Equivalent End Actions

```
function AML(Member  $M$ )  
  
    Loading Vector:  $\{L\}$   
    For each loading  $L$  in  $\{L\}$   
        Fixed End Moments :  
            Start Node:  $R_{sx}, R_{sy}, R_{sz}, M_{sx}, M_{sy}, M_{sz}$   
            End Node:  $R_{ex}, R_{ey}, R_{ez}, M_{ex}, M_{ey}, M_{ez}$   
            Local actions:  $A_{ml,local} =$   
  
             $\{R_{sx}, R_{sy}, R_{sz}, M_{sx}, M_{sy}, M_{sz}, R_{ex}, R_{ey}, R_{ez}, M_{ex}, M_{ey}, M_{ez}\}$   
            Global actions:  $A_{ml,global} = R^T * A_{ml,local}$   
             $A_{ml,total} += A_{ml,local}$   
        end for  
  
end function
```

Algorithm 4.8 Stiffness Matrix Assembly

```
function AssembleStiffness(Structure3 S)
Total Structure DOF : tDOF
Structure Stiffness Matrix:  $K[tDOF][tDOF]$ 

  for each Member in MemberVector:  $S \rightarrow \{M\}$ 
    Start Joint DOF indices:  $sDoF = dx_s, dy_s, dz_s, rx_s, ry_s, rz_s$ 
    End Joint DOF indices:    $eDoF = dx_e, dy_e, dz_e, rx_e, ry_e, rz_e$ 
    Global Stiffness Matrix:  $K_g[12][12]$ 
     $localDOF_{index,row} = 0$ 
    indexCombinations={  $\{sDOF_{index}, sDOF_{index}\}, \{sDOF_{index}, eDOF_{index}\},$   

                        $\{eDOF_{index}, sDOF_{index}\}, \{eDOF_{index}, eDOF_{index}\}$  }
     $localDOF_{index,row} = 0$ 
    for each indexCombination in indexCombinations do
       $\{index_{row}, index_{column}\} \leftarrow indexCombination$ 
      for each  $index_{row}$  in  $sDoF$  do
         $localDOF_{index,column} = 0$ 
        for each  $index_{column}$  in  $eDoF$  do
           $K[index_{row}][index_{column}] = K[index_{row}][index_{column}] +$   

             $K_g[localDOF_{index,row}][localDOF_{index,column}]$ 
           $localDOF_{index,column}++$ 
        end for
         $localDOF_{index,row}++$ 
      end for
    end for
  end for
end function
```

Algorithm 4.9 Application of Boundary Conditions

```
function ApplyBoundaryCondition(Structure S)
  Array: unrestrained DOF Index :  $uDOF[]$ 

  Reduced Stiffness Matrix:  $K_r$ 
  Reduced Load Vector:  $F_r$ 
   $r = 0, c = 0$ 
  for each  $uDOF_{row}$  in  $uDOF[]$  do
     $c = 0$ 
    for each  $uDOF_{column}$  in  $uDOF[]$  do
       $K_r[r][c] = K[uDOF_{row}][uDOF_{column}]$ 
       $c++$ 
    end for
     $F_r[r] = loadVector[uDOF_{row}]$ 
     $r++$ 
  end for

end function
```

Algorithm 4.10 Reaction Stiffness Matrix Assembly

```
function AssembleReactionStiffness(Structure S)
  Array: restrained DOF Index :  $cDOF[]$ 
  Array: unrestrained DOF Index :  $uDOF[]$ 

  Reaction Stiffness Matrix:  $K_{rxn}$ 
  Reaction Load Vector:  $F_{rxn}$ 
   $r = 0, c = 0$ 
  for each  $cDOF_{row}$  in  $cDOF[]$  do
     $c = 0$ 
    for each  $uDOF_{column}$  in  $uDOF[]$  do
       $K_{rxn}[r][c] = K[cDOF_{row}][uDOF_{column}]$ 
       $c++$ 
    end for
     $F_{rxn}[r] = loadVector[cDOF_{row}]$ 
     $r++$ 
  end for

end function
```

Algorithm 4.11 Computation of Displacements

```
function ComputeDisplacements(Structure S)

    Reduced Load Vector :  $F_r$ 
    Master/Structure Stiffness Matrix:  $K$ 
    Displacement Vector:  $Disp$ 
    Diagonal Preconditioner :  $Diag()$ 
    Conjugate Gradient Solver : Solver(out  $Disp$ )
    Solver->preconditioner( $Diag()$ )
    Solver->ConjugateGradientSolve( $F_r$ ,  $K$ ,  $Disp$ )
    return  $Disp$ 

end function
```

Algorithm 4.12 Computation of Reactions

```
function ComputeReactions(Structure S)

    Reaction Load Vector :  $F_r$ 
    Reaction Vector :  $F_{rxn}$ 
    Reaction Stiffness Matrix:  $K_{rxn}$ 
    Displacement Vector:  $Disp$ 
     $F_{rxn} = F_r + K_{rxn} * Disp$ 
    return  $F_{rxn}$ 

end function
```

Algorithm 4.13 The Conjugate Gradient Method

```
function ConjugateGradientSolve( $b$ ,  $A$ ,  $x$ )

    Generate  $x_0$ 
     $r_0 \leftarrow b - Ax_0$ 
    Steepest Descent:  $s_0 \leftarrow r_0$ 
    for k from 0 to RowLength( $A$ ) do
         $\alpha_k \leftarrow s_k^T r_k / s_k^T A s_k$ 
         $x_{k+1} \leftarrow x_k + \alpha_k s_k$ 
         $r_{k+1} \leftarrow b - Ax_{k+1}$ 
        if  $|r_{k+1}| \leq \epsilon$  exit loop
         $\beta_k \leftarrow -r_{k+1}^T A s_k / s_k^T A s_k$ 
         $s_{k+1} \leftarrow r_{k+1} + \beta_k s_k$ 
    end for
    return  $x$ 

end function
```

4.8.2 Structural Constraints

Algorithm 4.14 Algorithm for Determining Biaxial Interaction Boundary Coordinates of Members

```

function BiaxialBoundary(Member M)
  Biaxial Interaction Coordinates:  $\{P_i\}=\{(M_1, N_1), (M_2, N_2), \dots\}$ 
  Section Profile: Dimensions [breadth: b], [height: h]
    Corner reinforcement:  $\{r_1, r_2, r_3, r_4\}$ 
    Extra reinforcement, Y:  $\{r_{y1}, r_{y2}, r_{y3}, r_{y4}\}$ 
    Extra reinforcement, Z:  $\{r_{z1}, r_{z2}, r_{z3}, r_{z4}\}$ 
  Design Actions:  $M_{yED}, M_{zED}, N_{ED}$ 
  Neutral Axis Orientation:  $\theta = \text{Arctan}(M_{yED}/M_{zED})$ 
  Maximum Neutral Axis Location:  $x_{max} = Y_{min} + (Y_{max} - Y_{min}) * 1.25$ 
  Neutral Axis Depth:  $x$ 
  While  $x < x_{max}$  do

    Concrete Compression Area Coordinates:  $C_{xy}=\{(X_1, Y_1), (X_2, Y_2), \dots\}$ 
    Centroid(X,Y)= ComputeCentroid( $C_{xy}$ )
    Area= ComputeArea( $C_{xy}$ )
    Concrete Compressive Force,  $F_c = \text{Area} * fcd/1000.0$  kN
    Concrete Compressive Moment,  $M_c =$ 
     $F_c * (\text{CenterY} - \text{CentroidY})/1000.0$  kN.m
    For every reinforcement bar R $\{\phi, X, Y\}$ , in
       $\{r_1, r_2, r_3, r_4, r_{y1}, r_{y2}, r_{y3}, r_{y4}, r_{z1}, r_{z2}, r_{z3}, r_{z4}\}$  do
        If (RY>x) then
           $\epsilon_{bottom} = ((L_{total} - L_{NA})/L_{NA}) * \epsilon_{cm}$ 
           $\epsilon_R = (R_Y - x)/(L_{total} - L_{NA}) * \epsilon_{bottom}$ 
        Otherwise
           $\epsilon_R = (x - R_Y)/L_{NA} * \epsilon_{cm}$ 
        End If
        If ( $\epsilon_R > \epsilon_{yd}$ ) then  $\epsilon_R = \epsilon_{yd}$ 
        Steel Stress,  $\sigma_{sR} = \epsilon_R * E_s$ 
        Steel Force,  $F_{sR} = \sigma_{sR} * A_R$  where  $A_R = \phi_R^2 * \pi/4$ 
        Steel Moment,  $M_{sR} = F_{sR} * (R_Y - \text{centerY})/1000.0$ 
      End For
     $N_i = F_{total} = \sum F_{sRi} + F_c$ 
     $M_i = M_{total} = \sum M_{sRi} + M_c$ 
     $P_i \leftarrow \{N_i, M_i\}$ 
  End While
  Return  $\{P_i\}$ 
End Function

```

Algorithm 4.15 Efficiency Computation for Biaxial Capacity

```
Function BiaxialEfficiency()  
Efficiency : e  
Interaction Coordinates  $\{P_i\}$ : BiaxialBoundary $\{(M_1, N_1), (M_2, N_2), \dots\}$   
Design Actions:  $M_{yED}, M_{zED}, N_{ED}$   
Resultant Moment:  $M_{rED} = \sqrt{(M_{zED})^2 + (M_{yED})^2}$   
Design Coordinates:  $P_{ED} = (M_{rED}, N_{ED})$   
i=insideBoundary( $P_{ED}, \{P_i\}$ )  
If  $i \neq null$  then  
    e=computeEfficiency( $P_{ED}, i$ )  
    Return e  
Else  
    Return e=-1  
End If  
Biaxial Penalty:  $C_{biax} = 1 - e$   
Return  $C_{biax}$ 
```

Algorithm 4.16 Check for a Point Residing in a Closed Boundary

```
Function InsideBoundary( $P_r, R$ )  
    Boundary Geometry:  $R = \{(X_0, Y_0), (X_1, Y_1), \dots\}$   
    Point:  $P_r = (X_r, Y_r)$   
    Contour Origin:  $P_C = (X_C, Y_C)$   
    Residence Region Index :  $m$   
    Inside Boundary: Check=false  
    while  $i < \text{Length}(R)$  and Check=false do  
         $Triangle_i = \{P_C, R[i], R[i + 1]\}$   
        if(insideTriangle( $P_r, Triangle_i$ ))  
            Check=true  
             $m = i$   
        end if  
    end while  
Return
```

Algorithm 4.17 Check for a Point Residing in a Triangle

```
Function InsideTriangle(Point  $P_r$ , Region  $Triangle$ )

Triangle Coordinates:  $T = \{(X_0, Y_0), (X_1, Y_1), (X_2, Y_2)\}$ 
Point:  $P_r = (X_r, Y_r)$ 
Region 1:  $R_1 = \{(X_r, Y_r), (X_1, Y_1), (X_2, Y_2)\}$ 
Region 2:  $R_2 = \{(X_0, Y_0), (X_r, Y_r), (X_2, Y_2)\}$ 
Region 3:  $R_3 = \{(X_0, Y_0), (X_1, Y_1), (X_r, Y_r)\}$ 
Triangle Area:  $A_T = |((x_0 * (y_1 - y_2)) + x_1 * (y_2 - y_0) + x_2 * (y_0 - y_1))|$ 
Region 1 Area:  $A_1 = |((x_r * (y_1 - y_2)) + x_1 * (y_2 - y_r) + x_2 * (y_r - y_1))|$ 
Region 2 Area:  $A_2 = |((x_0 * (y_r - y_2)) + x_r * (y_2 - y_0) + x_2 * (y_0 - y_r))|$ 
Region 3 Area:  $A_3 = |((x_0 * (y_1 - y_r)) + x_1 * (y_r - y_0) + x_r * (y_0 - y_1))|$ 
If  $(|A_T - (A_1 + A_2 + A_3)| < 0.000001)$ 
    Return true
Else if
    Return false

End function
```

Algorithm 4.18 Shear Resistance Penalty for Reinforced Concrete Beam Elements

```
Function shearCapacity()
for  $i = Y, Z$ 
Material Properties:  $E_c, E_s, f_{ck}, f_{yd}$ 
Design Properties:  $H, B, S, A_{s1}, hp, d$ 
Design Shear:  $V_{ed,i}$ 
Spacing Provided:  $S_i$ 
Penalty:  $C_{shear,i}$ 
Shear Resistance:  $V_{rd,max,i} = 0.124 * b_w d * (1 - f_{ck}/250) * f_{ck}$ 
if  $(V_{rd,max} \geq V_{ed})$ 
     $\theta = 22^\circ$ 
else
     $\theta = 0.5 * \sin^{-1}\left(\frac{V_{ed,i}}{0.18 * b_w d * f_{ck} * (1 - f_{ck}/250)}\right)$ 
end if
Shear Spacing:  $S_{reqd,i} = 0.78 * d * f_{yk} * \cot(\theta) * A_{sw}/V_{ed,i}$ 
 $C_{shear,i} = \text{Penalty}(S_i, S_{reqd,i})$ 
Return  $C_{shear,i}$ 
End Function
```

Algorithm 4.19 Torsion and Shear-Torsion Interaction Penalties

```
Function torsionShearCapacity()
Material Properties:  $E_c, E_s, f_{ck}, f_{yd}$ 
Design Properties:  $H, B, S, A_{s1}, hp, d$ 
Design Torsional Moment:  $T_{ED}$ 
Cross-Section Area:  $B*H$ 
Cross-Section perimeter:  $B*2+H*2$ 
Equivalent Hollow Section Thickness  $t = \frac{(B*H)}{(B*2+H*2)}$ 
Equivalent Hollow Section Area  $A_k = (B - t) * (H - t)$ 
Efficiency Factor  $\nu = (1 - f_{ck}/250) * 0.7$ 
Truss Inclination Angle  $\theta=22^\circ$ 
 $T_{RD,max} = 2 * \nu * f_{cd} * A_k * t * \sin(\theta) * \cos(\theta) / 1000000.0$ 
Torsion Spacing:  $S_t = \frac{2*A_k*0.87*f_{yk}*cot(\theta)*A_{sw}}{T_{ED}}$ 
Shear Spacing:  $S_{reqd,i} = 0.78 * d * f_{yk} * cot(\theta) * A_{sw} / V_{ed,i}$ 
Total Spacing :  $S_{reqd,total} = \frac{A_{sw}}{(A_{sw}/S_t) + (A_{sw}/S_{reqd,i})}$ 
Shear-Torsion Interaction  $C_{s-t} = \text{shearTorsionInteraction}(T_{rd,max}, T_{ED}, V_{rd,max}, V_{ED})$ 
Shear-Torsion combined penalty: Penalty( $S_i, S_{reqd,total}$ )
Return  $C_{torsion}, C_{s-t}$ 
End Function
```

Algorithm 4.20 Combined Interaction Capacity for Shear and Torsional Actions

```
Function shearTorsionInteraction()
Material Properties:  $E_c, E_s, f_{ck}, f_{yd}$ 
Design Properties:  $H, B, S, A_{s1}, hp, d$ 
Design Torsional Moment:  $T_{ED}$ 
Design Shear:  $V_{sd,Y}, V_{sd,Z}$ 
interaction Coordinates:  $(T_{RD,max}, 0), (0, V_{RD,max})$ 
design Coordinates:
if insideBoundary( $(T_{ED}, V_{EDi}), (T_{RD,max}, 0), (0, V_{RD,max})$ ) then
    e=computeEfficiency( $(T_{ED}, V_{EDi}), (T_{RD,max}, 0), (0, V_{RD,max})$ )
Else
    e=-1
End if
 $C_{shear-torsion} = 1 - e$ 
Return  $C_{shear-torsion}$ 
End Function
```

4.8.3 Genetic Simulation

Algorithm 4.21 A Simple Genetic Algorithm

```
function GASimulation()
Objective function and constraints:  $O(), \phi(), \psi()$ 
Fitness function ->>  $Fitness(x) \propto O(), \phi(), \psi()$ 
Probabilities of crossover:  $P_c$  and mutation:  $P_m$ 
Elitism Percentage:  $e$  , Population Size:  $n$ 
Genetic operators:
{tournamentSelection(Population P),
 mutation(individual i),
 crossover(individual  $i_1$ , individual  $i_2$ )}
convergence criteria: Conv
//-----STAGE 1: INITIALIZATION-----//
Initialize population  $P(n)$ 
for each individual i in P do

    fitness(i)

end do
//-----//
//-----STAGE 2: SELECTION+EVOLUTION-----//
while (Conv = false)

    while ( $n_{new} < n$ )
        individual  $P_{new}[]$  = tournamentSelection( $P$ )
        Select 2 individuals from  $P_{new}$ :
            return individuals  $i_1, i_2$ 
        end select
         $i_{new}$  = crossover( $i_1, i_2$ )
    end while
    mutation( $P_{new}(random(i))$ )
    for each individual i in  $P$ 
         $f$  = fitness(decode(Structure S, Individual i))
    end for
    Sort( $P_{new}$ , parameter= $f$ )
    Select the top  $e$  percent of fit individuals

end while
//-----//
end function
```

Algorithm 4.22 Uniform Crossover

```
function Crossover(individual  $i_1$ , individual  $i_2$ )
  Crossover Probability:  $P_c$ 
  Offspring:  $i_3$ 
  Gene Sequences:  $G_1[]$  ,  $G_2[]$  ,  $G_3[]$ 
  Gene Length:  $g=\text{Length}(G_1[])$ 

  for i from 1 to  $g$  do
    random probability:  $r=\text{random}(0,1)$ 
    if ( $r \leq P_c$ ) then
       $G_3[i] = G_1[i]$ 
    else
       $G_3[i] = G_2[i]$ 
    end if
  end for

End function
```

Algorithm 4.23 Mutation

```
function Mutation(individual  $i_1$ )
  Mutation Probability:  $P_m$ 
  Gene Sequence:  $G_1[]$ 
  Gene Length:  $g=\text{Length}(G_1[])$ 

  for i from 1 to  $g$  do
    random probability:  $r=\text{random}(0,1)$ 
    if ( $r \leq P_m$ ) then
      //generate a mutated gene
      mutation:  $m = \text{random}(0,1) * G_1[i]$ 
       $G_1[i] = m$ 
    else
      // Do Nothing
    end if
  end for

End function
```

Algorithm 4.24 Tournament Selection of Best Individuals

```
function tournamentSelection(Population  $P$ )

    Tournament Size :  $n$ 
    Tournament Population:  $T$ 
    for  $i$  from 0 to  $n$  do

        Random Individual Index:  $r = \text{random}(0,1) * \text{Length}(P)$ 
         $T[i] = P[r]$ 
    end for
    Return Best Individual  $i_{fit}$ : fittest[ $T$ ]

End function
```

Algorithm 4.25 Gene Decoding

```
function decode(Structure  $S$ , Individual  $i$ )

    Number of Members:  $n_{members}$ 
    Breadth Data: BIndex[] = {250, 300, 325, 350, ...}
    Height Data: HIndex[] = {250, 300, 325, 350, ...}
    Flexural Reinforcement Data: RIndex[] = {{ $n_{T0}\phi d_{T0} - n_{B0}\phi d_{B0}$ }, { $n_{T1}\phi d_{T1} - n_{B1}\phi d_{B1}$ }, ...}
    Shear Spacing Data: SpIndex[] = {100, 110, 120, 130, ...}
    Member Vector:  $S \rightarrow M$ 
    gene index:  $g=0$ 
    for  $i$  from 0 to  $n_{members}$  do
        Member:  $m = M[i]$ 
        Breadth:  $m \leftarrow b = \text{BIndex}[g]$ 
         $g++$ 
        Height:  $m \leftarrow h = \text{HIndex}[g]$ 
         $g++$ 
        Flexural Reinforcement :  $m \leftarrow R = \text{RIndex}[g]$ 
         $g++$ 
        Shear Reinforcement :  $m \leftarrow$ 
         $Sp = \text{SpIndex}[g]$ 
         $g++$ 
    end for

End function
```

Algorithm 4.26 The Penalty Function

Function Penalty($X_{allowable}$, X_{design})

Allowable Value: $X_{allowable}$

Design Value: X_{design}

Penalty: C

Maximum Penalty: C_{max}

$C = (\frac{X_{design}}{X_{allowable}}) - 1$

if ($C < 0$)

$C = 0$

else if ($C > 1$)

$C = C_{max}$

end if

Return C

Algorithm 4.27 Individual Fitness Computation

Function Fitness()

Individual : i

Structure: S

Material Properties: E , E_s , f_y , f_c

Geometric Properties:

For each member M in $\{M\}$ do

If Beam?(M)

$M \leftarrow \{B, H, A_{st}, A_{sb}, A_{sn1}, A_{sn2}, A_{sp}, A_{sv1}, A_{sv2}\}$

Else

$M \leftarrow \{B, H, A_{s1}, A_{s2}, A_{sv}\}$

End if

end For

Analyze(S)

Cost: $C = \text{ComputeCost}(S)$

Penalty: $p = \text{TotalPenalty}(S)$

Penalized Fitness Function: $F = (1 + p) * C$

$S \leftarrow F$

End function

Algorithm 4.28 Total Penalty of a Structure

```
Function TotalPenalty(S)
TotalPenalty:  $p$ 
For each member  $m$  in  $S \rightarrow \{m\}$ 

     $p_{torsion} = \text{torsionShearCapacity}()$ 
     $p_{shear} = \text{shearCapacity}()$ 
     $p_{biaxial} = \text{biaxialEfficiency}()$ 
     $p_{sizing} = \text{sizingConstraints}()$ 
    MemberPenalty:  $p_{mem} = p_{torsion} + p_{shear} + p_{biaxial} + p_{sizing}$ 
     $p += p_{mem}$ 

end For
return  $p$ 
End Function
```

Algorithm 4.29 Total Cost of a Structure

```
Function ComputeCost(S)

Total Cost:  $C_{total} = 0.0$ 
Material Unit Weights:  $\gamma_{conc}, \gamma_{steel}$ 
Material Unit Costs:  $c_{conc}, c_{steel}, c_{form}, c_{scaffold}$ 
For each member in  $S \rightarrow m$ 

     $C_{conc} = c_{conc} * \gamma_{conc} * b * h * l$ 
     $C_{steel} = c_{steel} * (\sum \gamma_{steel} A_s l + \sum \gamma_{steel} A_{sv} l)$ 
    if ( $m$  is vertical) then
         $C_{formwork} = c_{form} * (2 * h + 2 * b) * l$ 
    else
         $C_{formwork} = c_{scaffold} * (b * l) + c_{form} * (2 * h + b) * l$ 
    end if
     $C_{total} = C_{total} + C_{formwork} + C_{conc} + C_{steel}$ 

End For
End Function
```

4.8.4 Flow Charts

This sub-section enlists flow charts to be used in conjunction with the algorithms in the previous sub-sections. Aside from auxiliary methods, all major algorithms discussed in this chapter have been included in the flow charts for simple convenience of cross-reference with sections 4.8.1, 4.8.2, 4.8.3.

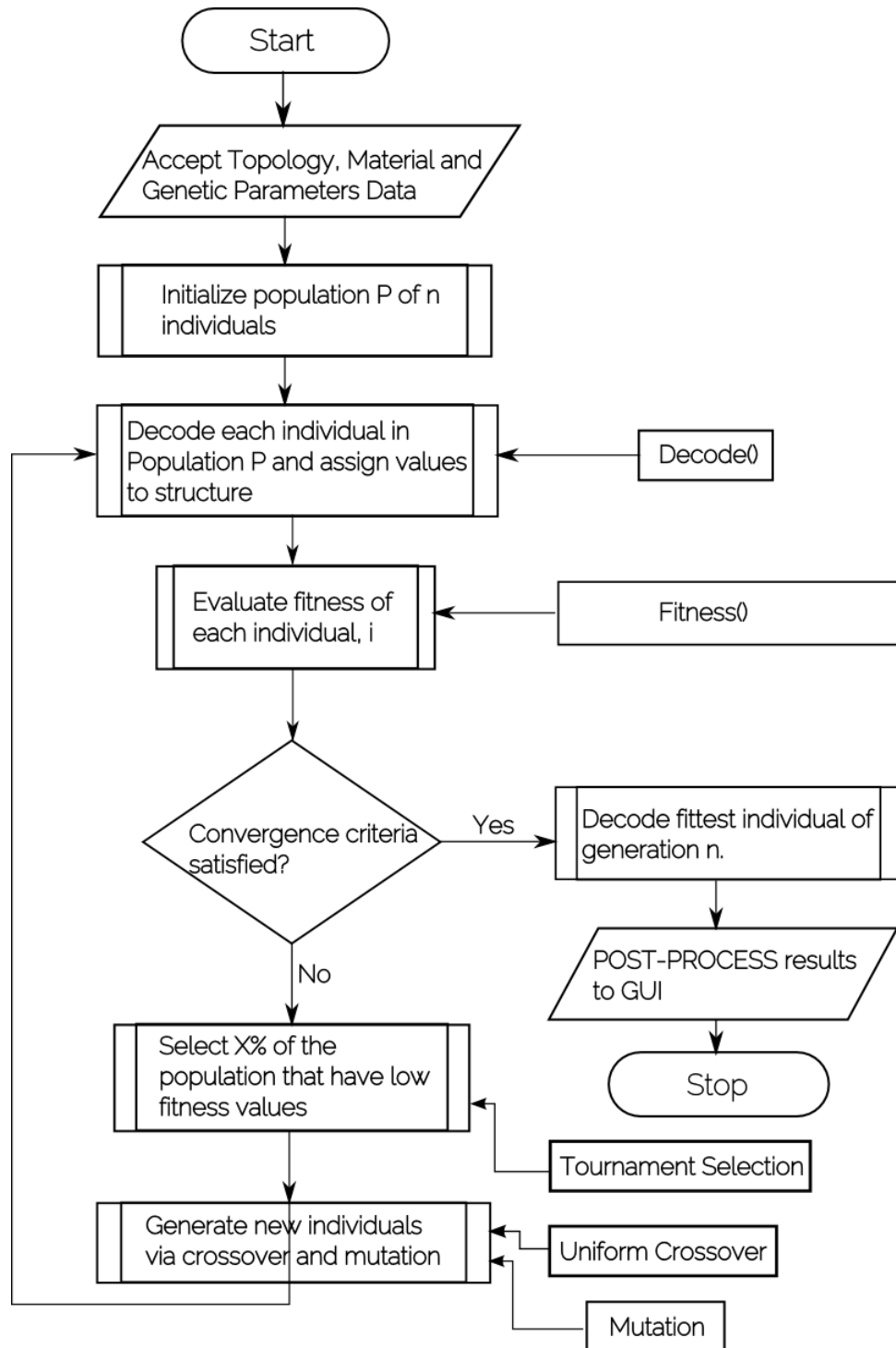


Figure 4.6: Main Program Flow Chart

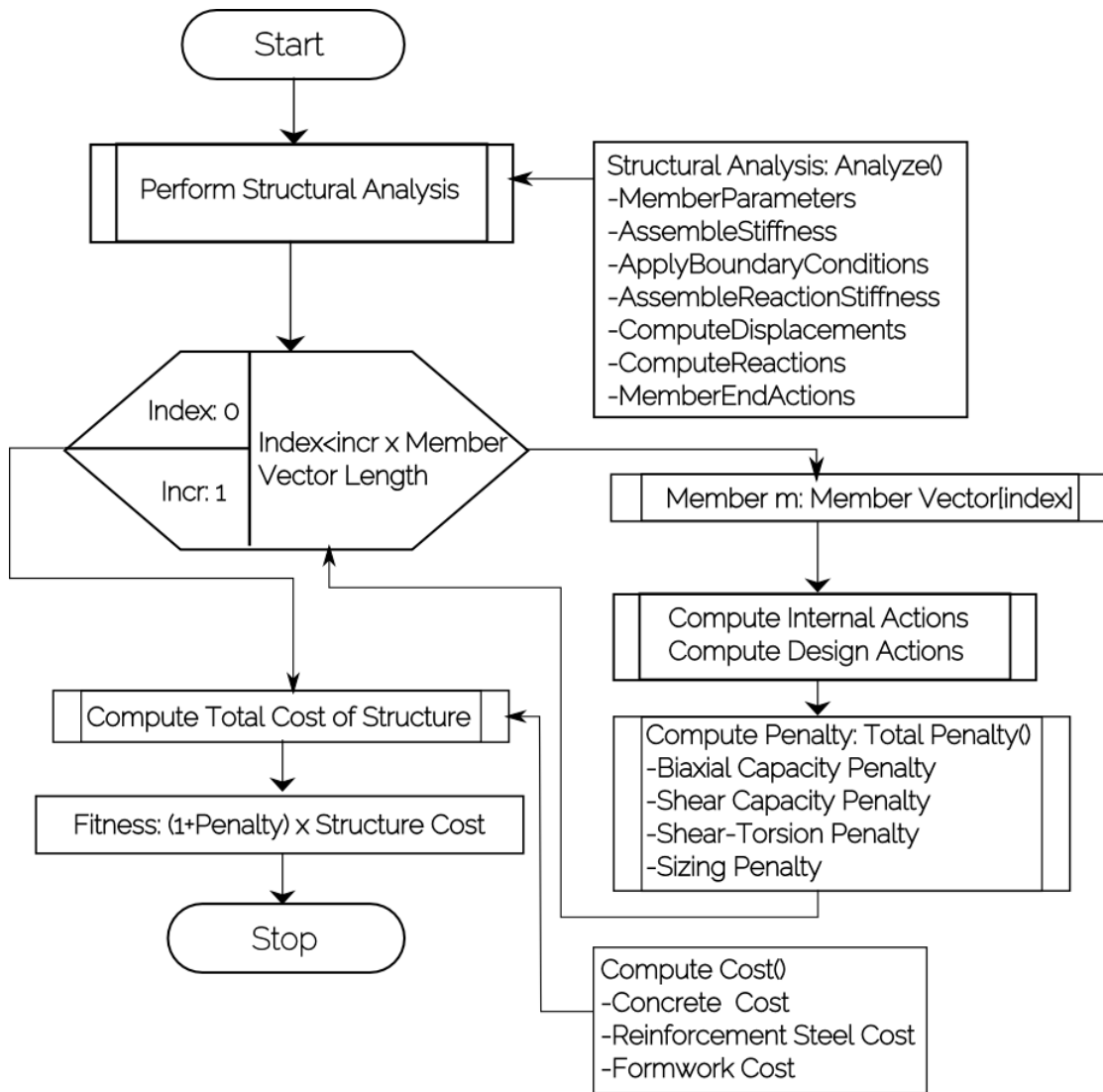


Figure 4.7: Fitness Computation Flow Chart

Chapter 5

Results and Discussion

5.1 Introduction

In this chapter, the algorithms discussed previously are put to use on reference and example problems and the results are presented and analyzed. Additionally, the chapter also discusses about observations made from varying genetic operator parameters and their effect on performance, convergence and accuracy of solution. The results of the program are also compared for accuracy, with other research attempts and with traditional design paradigms.

5.2 Models for Design Results Verification

To verify the results obtained from the genetic based simulation, experiments that can compare these results to values obtained from other methods need to be designed. These experiments can take one or more of the following four approaches.

1. To compare the results with designed and built building frames.
2. To evaluate the complete search space of the problem. This approach is mainly size dependent, and can only be used to get results for small problems since size of the search space is bound to increase exponentially, causing computational explosion. However, it is a foolproof method to get the global optimum for any optimization problem.
3. To compare results with benchmarks developed in past research efforts.
4. To evaluate the outputs obtained from other structural design software such as SAP, ETABS or SAFE.

5.3 Verification Environment

All tests in this chapter were done on a single computer with specifications given in table 5.1.

Feature	Value
RAM	8GB
No. of Cores	4
No. of Threads	4
Hyper-threading Support	Yes
Processor Frequency	3.2 GHz
Processor Architecture	x64
OS Architecture	x64

Table 5.1: Benchmark Computer Specifications

5.4 MODEL 1: Validation using a Benchmark

5.4.1 Problem Description

The work on optimizing building frames done by Vidosa et. al. [25] has been used to compare the magnitude of results obtained by the procedures developed above. The benchmark consists of a 5-story 2-bay frame and the procedure used for optimization was simulated annealing. For the sake of comparison, all material properties and cost indices were taken identical to this study. The values are given in tables 5.2 and 5.3

Material Properties	Value
Cylindrical Strength of Concrete, f_{ck} , MPa	35
Young's Modulus of Concrete, E_c , GPa	34
Shear Modulus of Concrete, G_c , GPa	13.07
Poisson's Ratio, ν_c	0.3
Characteristic Yield Strength of Steel, f_{yk} , MPa	500
Young's Modulus of Steel, E_s , GPa	200
Shear Modulus of Steel, G_s , GPa	80
Poisson's Ratio, ν_s	0.25

Table 5.2: Material Parameters

The problem used a combined vertical load of 35kN/m at each floor level (including self-weight), and a wind load magnitude of 4.5kN/m applied horizontally on columns of the left side(paper-wise). The simulation checked for ultimate limit states for flexure, shear and instability as well as for service conditions satisfying deflection based on the Spanish code standards. The optimization was treated as a discrete optimization problem with predefined reinforcement details and section sizing. A cost of €4458.08 has been

Cost	Units	Cost(€)/Unit
B-500S (S-500) Reinforcing Steel *	<i>kg</i>	1.3
HA-45 (C-35/45) Concrete*	m^3	112.13
Formwork for Beams	m^2	25.05
Formwork for Columns	m^2	22.75
Scaffolding for Beams	m^2	38.89
*Designations are based on the Spanish Design Codes		

Table 5.3: Cost Parameters

reported after using simulated annealing. The simulation converged in 105000 iterations and took 97 minutes to complete on a 3.2GHz processor. For comparison purposes, the total cost was re-evaluated using the cost function defined in equation 4.4. A value of €4887.52 was obtained using the details described in the research paper (See Appendix A).

5.4.2 Design Space

The total number of decision variables could be quantified by as follows.

- ▷ Two variables are needed to describe the cross-sectional geometry of each member, namely the width and height.
- ▷ Base flexural reinforcement(section 4.5) is provided throughout the member. This variable assumes any of the available reinforcement diameters for the current simulation.
- ▷ Extra reinforcements(section 4.6) is provided at moment critical regions. If the structure is a building, then beams have 3 such critical regions while columns are assumed to have one region represented by the maximum moment magnitude. Additionally, extra reinforcements are to be provided for the major as well as the minor bending directions. 4 variables are needed for extra reinforcements of a section.
- ▷ Shear reinforcements(section 4.7) for beams are provided at three different locations: the left support region, the right support region and the mid-span region. Hence, two variables¹ are needed. Shear reinforcement for columns is provided using a single spacing value and requires only one variable.
- ▷ Therefore, for a beam element, a total of $2+1+4*3+2=17$ variables are required to represent it. For a column element, $2+1+4+1=8$ variables are needed. Thus, a frame would need $17 * N_{beam} + 8 * N_{column}$ variables to represent it.

¹The mid-span region is provided with the minimum reinforcement spacing, which is calculated from $0.08f_{ck}^{0.5} * b_w / f_{yk}$. Thus, it doesn't need a variable to represent it.

For the frame in figure 5.1, the total number of decision variables required are thus $17 \cdot 10 + 8 \cdot 15 = 290$. By calculating the combinatorial possibilities of each possible solution, a search space of about 10^{150} solutions is obtained.

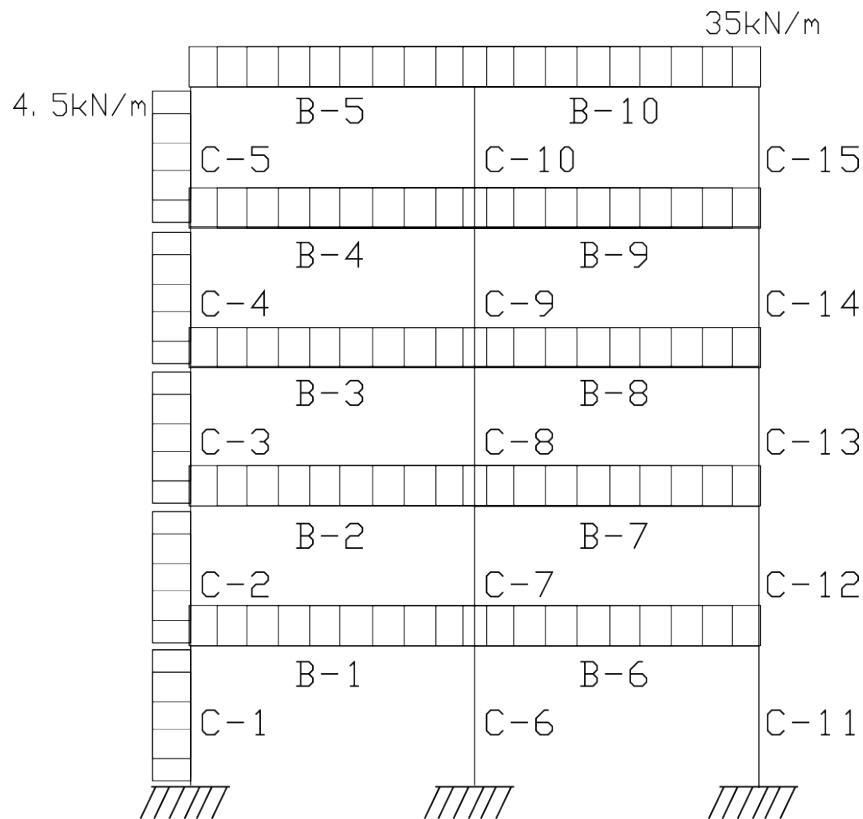


Figure 5.1: Benchmark 2D Frame

5.4.3 Genetic Simulation

For comparison purposes, the frame shown in figure 5.1 has been optimized by using the genetic algorithm formulation made earlier. To get unbiased results, some simulation features have been modified in the following ways.

- ▷ While structural analysis is carried out by assuming 3D members, the design of these members, however, proceeds in a two dimensional manner. Thus, both columns and beams will be treated as uniaxial members. Additionally, shear forces in the minor directions have been ignored.
- ▷ Flexural reinforcements to be generated by the simulator have also been restricted to utilize uniaxial profiles only.

5.4.4 Comparison

The genetic simulation was done for more than 40 runs using varied genetic parameter combinations. Population sizes varying from 40 up to 300 and mutation probabilities

ranging from 1% up to 6% were used in the test runs. The best result was found using a mutation probability of 3.2%, a crossover probability of 50% and a population size of 250, while convergence was achieved in 200 iterations. Each simulation took about 44 seconds to complete. The best individual had a fitness value of 18439.86 with a cost of €4644.8 (as compared to €4887.52 for SA)² and a total penalty value of 2.97. These results are close to those obtained in the research by Paya with an improvement of about 4.97%. Tables 5.4 and 5.5 give the complete design details obtained from the software. It can be observed that the simulation program achieved these results within 0.55%³ of the time taken for the SA simulation to complete. Furthermore, the GA simulation searched through a total of 250*200=50,000 possible designs to arrive at a comparable solution with the one obtained from SA methods(which searched through a total of 105,000 solutions) reducing the extent of the navigated search space by 47.6%.

ID	Breadth	Height	Basic R.	Extra R.	Shear R.	ϵ_{BIAX} *	Penalty
C-1	225	450	4 d 20	2 d 12	d8/ 200.0	99.53%	0
C-2	250	300	4 d 12	1 d 12	d8/ 200.0	99.75%	0
C-3	225	350	4 d 14	1 d 12	d8/ 200.0	96.89%	0.03
C-4	250	275	4 d 12	2 d 16	d8/ 200.0	94.90%	0.05
C-5	225	350	4 d 20	1 d 12	d8/ 200.0	94.92%	0.05
C-6	275	550	4 d 12	1 d 12	d8/ 200.0	98.66%	0.01
C-7	225	550	4 d 12	2 d 12	d8/ 200.0	95.82%	0.04
C-8	250	450	4 d 12	2 d 12	d8/ 200.0	93.66%	0.06
C-9	225	325	4 d 12	3 d 32	d8/ 200.0	99.79%	0
C-10	250	375	4 d 12	1 d 14	d8/ 200.0	99.17%	0.01
C-11	250	350	4 d 12	1 d 12	d8/ 200.0	91.58%	0.08
C-12	225	275	4 d 12	1 d 12	d8/ 200.0	99.18%	0.01
C-13	225	225	4 d 12	3 d 16	d8/ 200.0	95.32%	0.05
C-14	225	350	4 d 16	2 d 16	d8/ 200.0	94.44%	0.06
C-15	225	525	4 d 32	1 d 12	d8/ 200.0	91.81%	0.08

*See section 4.3

Table 5.4: Optimally Designed Columns for the GA simulation

²ETB113,268.09 as compared to ETB119,187.06, with €1.00=ETB24.386, received on March 20,2017

³Simulations from both research efforts used 3.2GHz Intel processors without multi- or hyper-threading, and since computational speed is independent of RAM, this percentage comparison can be made without bias.

ID	Breadth	Height	Base	Extra Y.	Shear L.	Shear S.	Shear R.	ϵ_{BIAX}	ϵ_{VY}	Penalty
B-1	225	275	4 d 24	1 d 12(TL), 2 d 14(TR), 1 d 12(B)	d8/ 130	d8/ 200	d8/ 110	95.07%	84.62%	0.25
B-2	225	250	4 d 24	1 d 12(TL), 2 d 12(TR), 1 d 12(B)	d8/ 130	d8/ 200	d8/ 130	97.63%	100.00%	0.07
B-3	250	300	4 d 20	1 d 12(TL), 1 d 12(TR), 1 d 12(B)	d8/ 140	d8/ 200	d8/ 150	97.09%	100.00%	0.13
B-4	225	375	4 d 16	1 d 12(TL), 1 d 12(TR), 1 d 12(B)	d8/ 130	d8/ 200	d8/ 100	98.59%	76.92%	0.2
B-5	225	300	4 d 16	2 d 12(TL), 1 d 12(TR), 2 d 12(B)	d8/ 120	d8/ 200	d8/ 120	90.44%	92.31%	0.39
B-6	225	300	4 d 24	1 d 12(TL), 3 d 20(TR), 1 d 12(B)	d8/ 120	d8/ 200	d8/ 130	96.30%	100.00%	0.16
B-7	275	400	4 d 24	1 d 12(TL), 2 d 24(TR), 1 d 12(B)	d8/ 170	d8/ 200	d8/ 150	92.28%	88.24%	0.31
B-8	225	425	4 d 16	1 d 20(TL), 3 d 20(TR), 2 d 14(B)	d8/ 130	d8/ 200	d8/ 110	97.53%	84.62%	0.18
B-9	300	350	4 d 20	1 d 12(TL), 1 d 24(TR), 1 d 12(B)	d8/ 170	d8/ 200	d8/ 190	99.04%	100.00%	0.1
B-10	275	375	4 d 24	1 d 14(TL), 1 d 12(TR), 1 d 12(B)	d8/ 160	d8/ 200	d8/ 150	82.56%	88.24%	0.64

Table 5.5: Optimally Designed Beams for the GA simulation

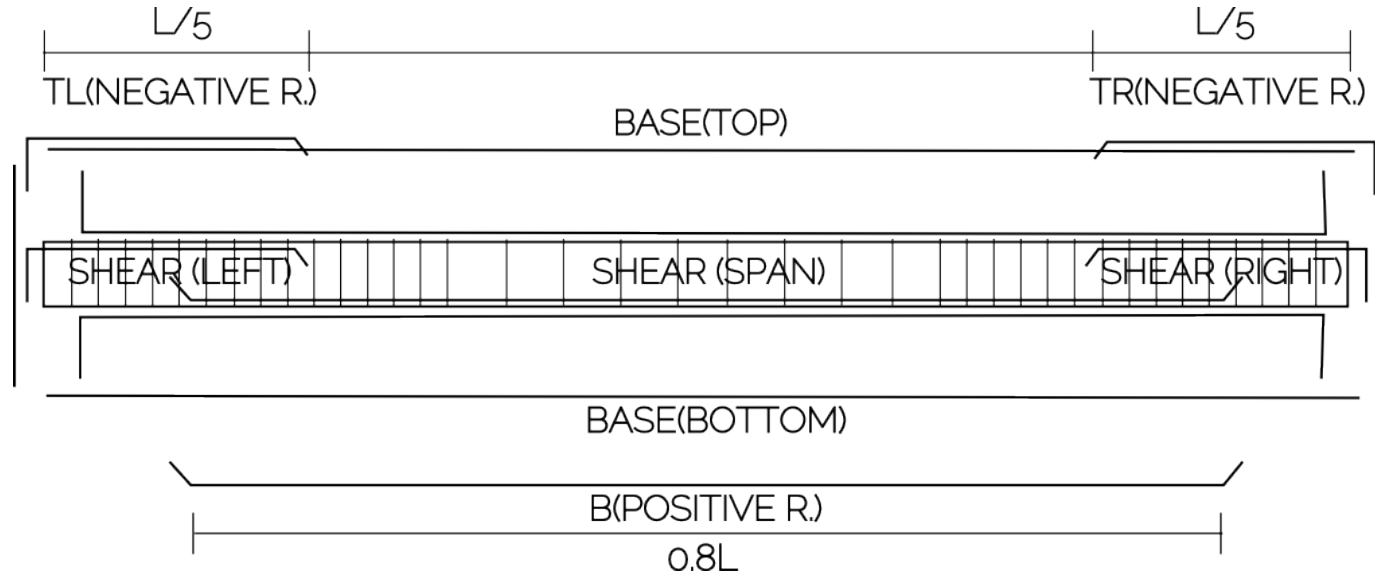


Figure 5.2: Reinforcement Configuration for Beams

5.5 MODEL 2: Validation using ETABS

5.5.1 Problem Description

The second experiment involved comparing the results obtained from HELIX to that designed by an industry standard structural design package. For this purpose, CSI ETABS 2015 x64(version 15.2.2) was used. The problem selected for comparison was a 2-story 1-bay 3D portal frame. All longitudinal dimensions are 3m in length. All model, material data were assumed according to table 5.3 and 5.2.

Since this is a reinforced concrete structural design, ETABS needs predefined concrete sections assigned to each member. However, if these section assignments were done by a human, a great deal of bias would be introduced (Since the engineer would preferentially choose the best/worst values for the section parameters based on experience). To avoid this, the AUTO-SELECT feature of ETABS has been used. This entails defining all cross-section geometries allowed in the analysis, and adding them to a section list in the ETABS section definition dialog. By assigning the defined auto-select section list to all members, initial sections for the purpose of analysis would be selected automatically by the software. This also has an added advantage when it comes to design. Instead of just designing/checking the reinforcement using predefined sections, the software would now be directed to choose a section profile economically from the auto-select list and assign it to the appropriate member, indirectly optimizing the frame. After such progress, the computation of reinforcements could proceed in the usual way.

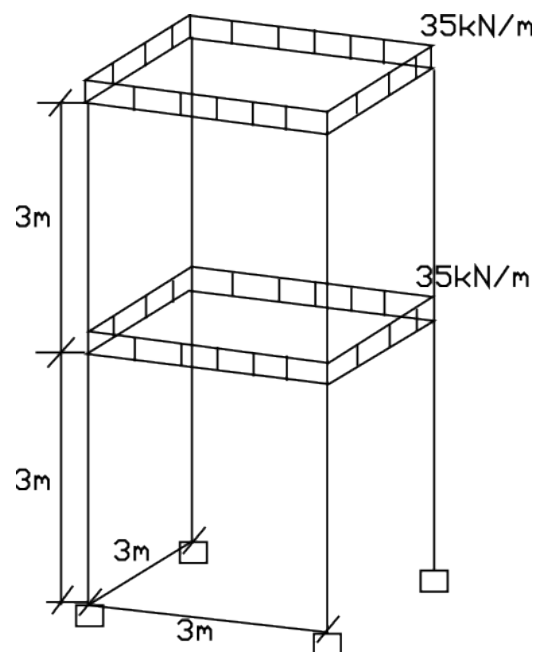


Figure 5.3: 2-story 1-bay Space Frame

5.5.2 Results and Comparison

The genetic algorithm simulations were done for 35 runs and an optimal solution was found at a fitness value of 6921.22 and had a total cost of €1870.6. These results were obtained after 200 iterations, using a population size of 200, mutation probability of 7% and a crossover probability 50%. The total number of solutions evaluated were $200 \times 200 = 40000$, and each simulation took 27 seconds to complete. Details of the result are given in tables 5.6 and 5.8.

The ETABS auto-selective design had a calculated cost of €1871.57⁴. The time benchmarks were not taken, since there were pauses between iteration confirmations, in addition to memory leaks that caused the software to freeze frequently. As it is evident, the results obtained by both software are close with an improvement of about 0.06%. This is so because ETABS uses relatively simple design evaluation mechanisms, that do not give considerations to the miscellaneous restraints (such as sizing, proportions and member efficiency) imposed in the previous chapter. Thus, the member designs obtained are highly economical unto themselves and do not consider the frame as a unit. Had it been so, much higher costs would have been obtained.

⁴ETB45,616.45 for GA as compared to ETB45,640.11 for ETABS

ID	Breadth	Height	Base	Extra Y.	Shear L.	Shear S.	Shear R.	ϵ_{BIAX}	ϵ_{VY}	Penalty
B-1	225	225	4 d12	1 d 12(TL), 2 d 14(TR), 1 d 12(B)	d8/ 100.0	d8/ 200.0	d8/ 120.0	94.72%	92.31%	0.36
B-2	275	300	4 d12	1 d 12(TL) , 2 d 12(TR) , 1 d 12(B)	d8/ 170.0	d8/ 200.0	d8/ 170.0	70.69%	100.00%	0.88
B-3	225	225	4 d12	1 d 12(TL) , 0 d 0(TR) , 1 d 12(B)	d8/ 130.0	d8/ 200.0	d8/ 120.0	96.70%	92.31%	0.15
B-4	225	225	4 d12	1 d 12(TL) , 1 d 12(TR) , 1 d 12(B)	d8/ 130.0	d8/ 200.0	d8/ 110.0	84.59%	84.62%	0.56
B-5	225	225	4 d12	1 d 12(TL) , 1 d 14(TR) , 1 d 12(B)	d8/ 130.0	d8/ 200.0	d8/ 130.0	97.06%	100.00%	0.09
B-6	225	225	4 d12	1 d 12(TL) , 1 d 12(TR) , 1 d 12(B)	d8/ 110.0	d8/ 200.0	d8/ 120.0	88.69%	92.31%	0.49
B-7	225	225	4 d12	1 d 12(TL) , 1 d 12(TR) , 1 d 12(B)	d8/ 130.0	d8/ 200.0	d8/ 100.0	95.99%	76.92%	0.27
B-8	225	225	4 d12	1 d 12(TL) , 1 d 12(TR) , 1 d 12(B)	d8/ 130.0	d8/ 200.0	d8/ 130.0	86.56%	100.00%	0.4

Table 5.6: Optimally Designed Beams for the GA simulation

ID	H	B	Top			Bottom			Shear			ID	H	B	Flex.	Shear
			Left	Mid	Right	Left	Mid	Right	Left	Mid	Right					
B-1	225	225	369	371	369	369	371	369	298	212	298	C-1	225	225	506.0	638.94
B-2	225	225	369	371	369	369	371	369	298	298	298	C-2	225	225	506.0	638.94
B-3	225	225	369	371	369	369	371	369	298	212	298	C-3	225	225	506.0	638.94
B-4	225	225	369	371	369	369	371	369	298	212	298	C-4	225	225	506.0	638.94
B-5	225	225	369	371	369	369	371	369	298	298	298	C-5	225	225	506.0	638.94
B-6	225	225	369	371	369	369	371	369	298	212	298	C-6	225	225	506.0	638.94
B-7	225	225	369	371	369	369	371	369	298	212	298	C-7	225	225	506.0	638.94
B-8	225	225	369	371	369	369	371	369	298	212	298	C-8	225	225	506.0	638.94

Table 5.7: ETABS Design Results

ID	Breadth	Height	Basic R.	Extra R.	Shear R.	ϵ_{BIAX}	Penalty
C-1	250	275	4 d12	1 d 12	d8/ 200.0	100.00%	0
C-2	225	225	4 d14	1 d 12	d8/ 200.0	100.00%	0
C-3	225	225	4 d12	1 d 12	d8/ 200.0	100.00%	0
C-4	250	275	4 d20	1 d 12	d8/ 200.0	100.00%	0
C-5	225	250	4 d12	1 d 12	d8/ 200.0	100.00%	0
C-6	225	225	4 d12	1 d 12	d8/ 200.0	100.00%	0
C-7	225	225	4 d12	1 d 12	d8/ 200.0	100.00%	0
C-8	325	350	4 d20	1 d 12	d8/ 200.0	100.00%	0

Table 5.8: Optimally Designed Columns for the GA simulation

5.6 Summary of Results

The following table summarizes the data and results obtained from the models developed in previous sections.

Parameters	Model 1	Model 2
No. of Members	25	18
Mutation Probability	3.2%	7%
Crossover Probability	50%	50%
Population Size	250	200
Number of Iterations	200	200
Optimal Fitness Value	18439.86	6921.22
Optimal Cost(GA)	€4644.8	€1870.6
Optimal Cost(ETABS)	–	€1871.57
Optimal Cost(SA)	€4887.5	–
Cost Difference	4.97%	0.06%

Table 5.9: Summary of Models 1 and 2

5.7 Effect of GA Parameters

As the operation of genetic algorithms by itself is stochastic and unpredictable, it is important to study the effect of simulation parameters on speed, performance, convergence and most importantly, the quality of solutions. The parameters tested were population size, number of iterations and mutation probability. By keeping all variables constant except for the target variable, its effect can be readily determined. The constant values the other variables can assume would be values for the best simulated results attained in the previous sections.

The 5-story by 2-bay frame used in the section 5.4 has been used here and was selected so as to have a comparable benchmark. To guarantee comprehensible results, each trial

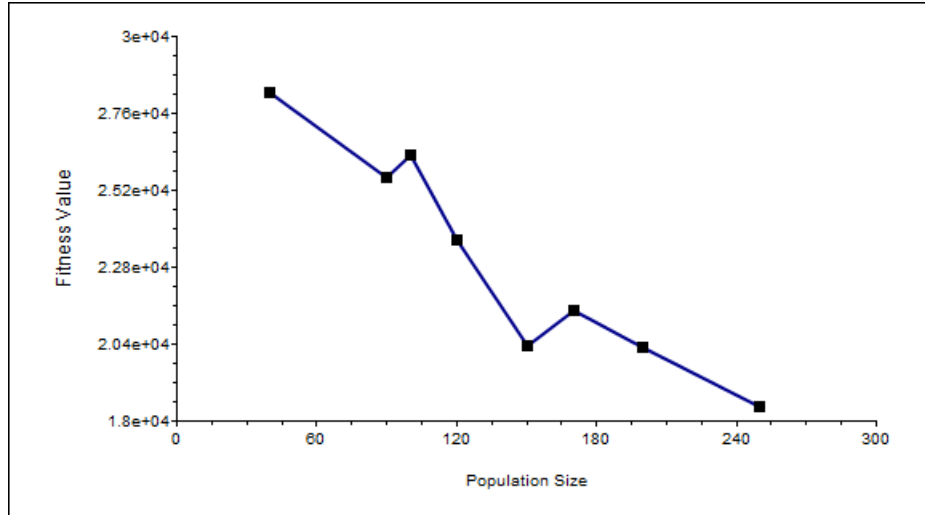


Figure 5.4: Population Size vs. Fitness

simulation has been done for a total of 5 runs. Loading configurations as well as material and cost parameters used in previous simulations have not been changed.

5.7.1 Effect of Population Size

The population is a group of individuals that are evaluated for fitness for a certain iteration. The size of a population for a given simulation determines the degree of variability between evaluated solutions. Additionally, genetic operators like crossover and selection help diversify types of individuals available and create highly fit solutions as the evolution proceeds further. As the evolution reaches the last stages however, most of the population would be saturated with similar potentially optimum individuals. In this case, the only way to ensure the creation of different individuals is through mutation.

Hence, given a constant mutation percentage, increasing population sizes tend to increase entropy of the simulation. This can be demonstrated by doing test simulations for variable population sizes while keeping other variables constant.

As seen from the table and plots, better results can be achieved by selecting larger population sizes. This is especially true for large sized problems with several design variables. Three points can be observed here:

1. Larger population sizes allow for a sufficiently large sector of the search space to be evaluated.
2. Such sizes also increase variability between solutions and hence help avoid premature convergence to local optima. Consequently, significantly large iteration counts are required for such simulations to converge.
3. Though optimum cost values can be obtained at small population sizes, these cost values appear as a result of violated constraints along with a higher penalty value.

Ultimately, the total fitness value would be the deciding factor, to which these low-cost high-penalty individuals would attain large fitness values.

Population Size	$P_{mutation}$	$P_{crossover}$	Least Cost (€)	Best Fitness
40	0.032	0.5	4926.08	28226.44
90	0.032	0.5	5097.21	25592.04
100	0.032	0.5	4805.72	26287.29
120	0.032	0.5	4576.89	23635.51
150	0.032	0.5	4822.10	20359.45
170	0.032	0.5	4668.30	21472.80
200	0.032	0.5	4741.00	20281.998
250	0.032	0.5	4658.53	18439.86

Table 5.10: Simulation results for varying Population Size

5.7.2 Effect of Mutation

Mutation is a highly stochastic process that might take the iteration towards a local/global optimum or sway it away in another random direction. Mutation directly affects the composition of a population by injecting random genes at random locations at random times. This property helps in diversifying the population composition and allows individuals to have more diverse genetic material to work with. Consequently, the stochasticity of mutation helps the evolution avoid being stuck at local optima, increasing entropy of the system, which indirectly improves the chances of superior solutions to be formed. The amount of mutation introduced into a system should however be limited, as improper application could result in several problems. A large mutation probability can inject too much chaos into the variability of the population, greatly reducing the probability of finding an acceptable optima as well as causing divergence in extreme cases. On the contrary, a very low probability value won't have enough force to drive the evolution achieve enough diversity. Thus, it is highly likely that a premature convergence at a local optimum would occur. In summary, small mutation percentages can benefit the evolutionary process but misuse can result in problematic or even senseless results. As seen from the data summary (figure 5.5 and table 5.11), the larger the mutation value gets the larger the fitness value attained, resulting in unusable results, even though small costs were obtained for large mutations.

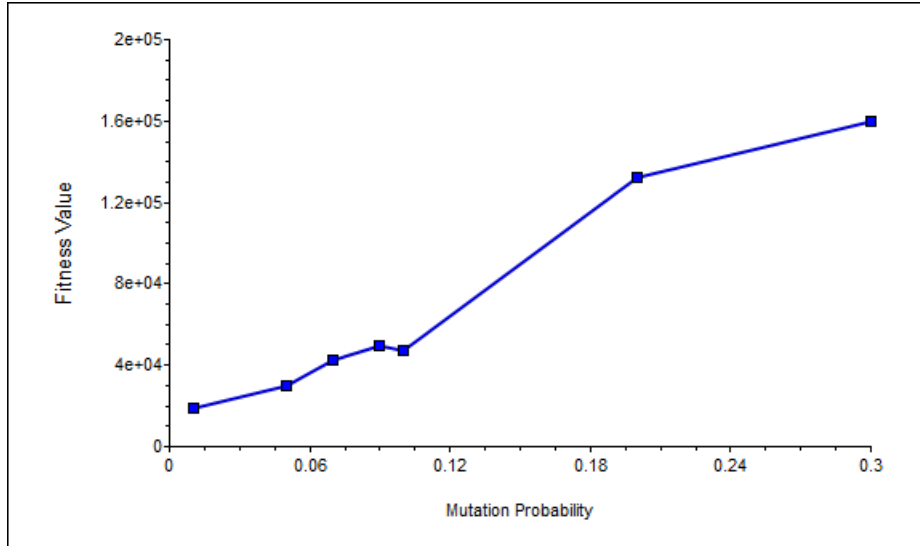


Figure 5.5: Mutation Probability vs. Fitness

Population Size	$P_{mutation}$	$P_{crossover}$	Least Cost (€)	Best Fitness
250	0.01	0.5	5215.18	19139.7106
250	0.05	0.5	4503.52	29948.408
250	0.07	0.5	4488.64	42327.8752
250	0.09	0.5	4260.29	49585.51531
250	0.1	0.5	4252.51	46947.7104
250	0.2	0.5	4864.12	132547.27
250	0.3	0.5	5650.93	159977.8283

Table 5.11: Simulation results for varying mutation probabilities

5.7.3 Effect of Evolution Period (Generation Size)

For a particular simulation, as the number of generations specified increases, the probability of convergence also increases. Once convergence is achieved however, the simulation needs to be stopped. If it is allowed to continue, the optimizer will start to generate individuals with deviant genes which might lead to divergent behavior or to significant modifications of the currently optimal individual. Therefore, the number of iterations needed for each simulation should be selected carefully. This should also be done with other genetic parameters in mind. For example, higher values for the mutation probability should be accompanied with lower values of generations. As mutation always drives the evolutionary process towards instability, such high mutation percentages increase the probability for divergence at later stages of the iteration. Higher population sizes dictate higher number of iterations be used, since such populations tend to have a lot of variability in them, causing instability for a large portion of the evolution period.

For large problems, that is, for structures with large number of elements, the selection

of these parameters needs special attention because computation time will start to have a significant effect. Additionally, the size of the search space would also be large enough that convergence to a global optima would be unlikely. Through several attempts, large population sizes with high mutation probabilities and longer evolution spans have been observed to achieve highly varied populations as well as faster convergence rates and acceptable local optima solutions.

5.8 Performance and Profiling

This section quantitatively discusses the computational complexity of genetic algorithms when applied to structural optimization of concrete frames. It is the authors believe that identifying and quantifying the time and memory overheads introduced by each procedure can be key in changing key aspects of the algorithm's construction towards a more efficient variant.

To profile the algorithm, regular portal frames of different sizes have been optimized and the result time and memory consumption were recorded. Apart from the total degrees of freedom available, the total number of non-zero entries (NNZ) in the structure stiffness matrix is a good indicator of the size of the problem. Table 5.12 gives a summary of the profiling results obtained for several size of frames optimized using the same genetic parameters.

Frame Size			DOFs	NNZ	$T_{Iteration}$	T_{Total}^*	RAM(MB)
N_{story}	$N_{bays,X}$	$N_{bays,Y}$					
1	1	1	48	152	0.0063**	4	423.4
2	2	2	162	984	0.045	16	460.5
3	3	3	384	2944	0.09	49	539
5	5	5	1296	12120	3	299	551
7	7	7	3072	31416	15	1860	612
10	10	10	7986	87392	136	12300	1019
* $P_{mutation} = 0.05, P_{crossover} = 0.05, N_{population} = 120, N_{iteration} = 90$							
**All times were measured in Seconds.							

Table 5.12: Simulation time for different frame sizes

5.8.1 Computation Time

It should not come as a surprise that GA simulation times increase as the size of the problem increases. The size of a problem can be seen from two different perspectives.

The first involves the topology of the structure to be optimized. Estimating the total time vs. the size of the problem for increasing complexity in topology is problematic. This happens mainly because of the variability in the time taken for structural analysis. Structural analysis takes 90-95% of the total time consumed for any particular simulation,

with the rest 5-10% being mainly distributed between computation of internal actions and design check of members among other procedures. Thus, structural analysis dominates the time variation displayed in such cases.

The second involves the total number of assessed solutions. In this case however, the amount of time taken could be easily quantified as:

$$T_{total} = T_{iteration} * N_{iteration} \quad (5.1)$$

where $T_{iteration}$ is total time elapsed per iteration given as:

$$T_{iteration} = N_{population} * T_{individual} \quad (5.2)$$

and $T_{individual}$ is the total time taken to analyze and evaluate a single individual from a certain population. Table 5.12 gives a highlight of how the problem size exponentially increases the simulation time. To minimize bias, all simulations were made using the same genetic parameters(given in the table). It should be noted that these time measurements were taken from an intrusive in-line profiler and are generally much larger in magnitude than the time taken for non-profiled simulations.

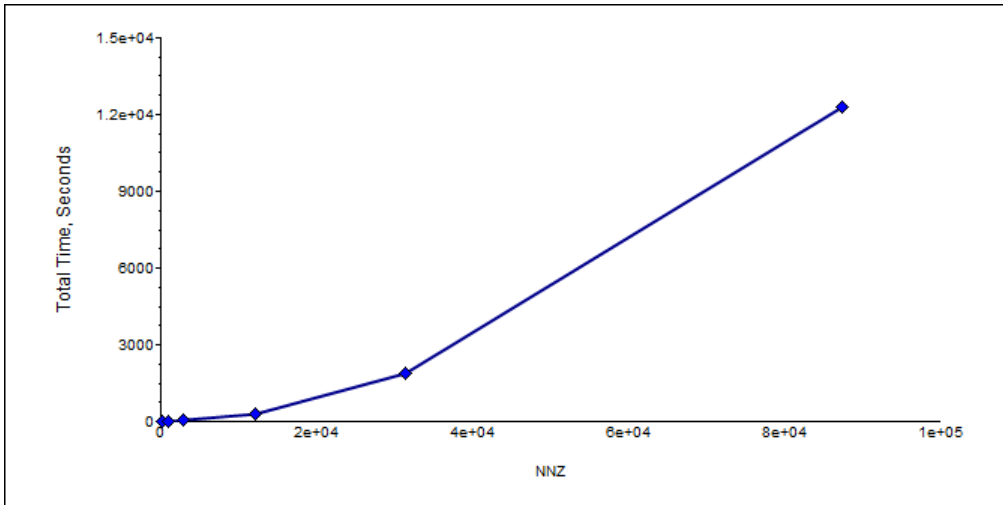


Figure 5.6: Total Time Vs. NNZ

5.8.2 Memory Usage

If the program is applied for large problems, certain computational resources become increasingly critical. One such resource is available RAM. If the total utilized RAM exceeds the total maximum allocated RAM for the JVM, the program will cease to operate and crash. This can be avoided by manually increasing the available JVM RAM(which in turn is dependent on total RAM installed in the computer) using the respective command line routines in operating systems such as Linux and Windows. Table 5.12 shows memory usage with respect to problem size. It is evident that large problems take up significant

portions of the JVM RAM. This is mainly due to the exponentially increasing number of non-zero entries in the SSM and its reduced variant. Other types of data also increase in size such as interaction diagram coordinate data, internal action coordinate data.

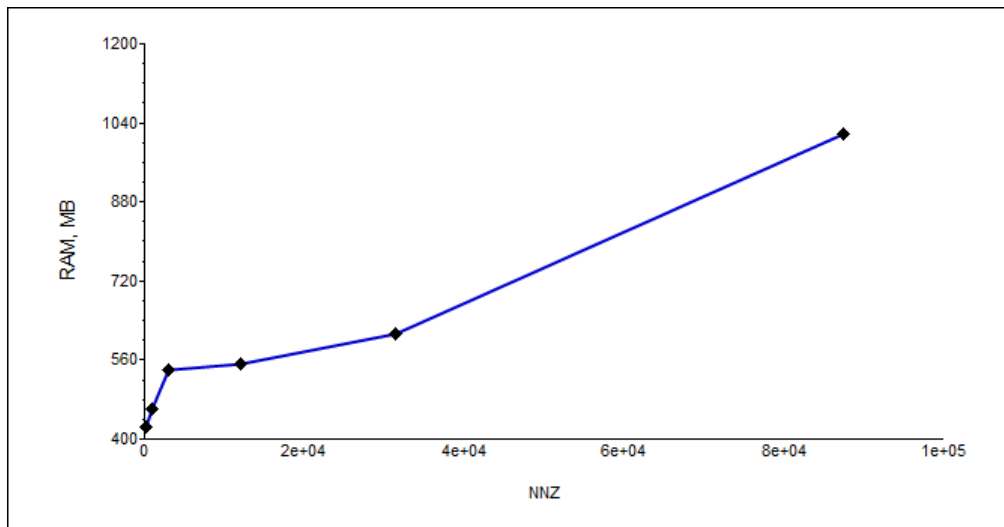


Figure 5.7: Memory Usage Vs. NNZ

Chapter 6

Conclusions

6.1 General

The research work has tried to develop a methodology for achieving local and global optima for two and three dimensional concrete frames with regards to cost and strength. Genetic algorithms have been used for the optimization, while the design procedures were based on those described in Eurocode-2. To implement and test the method, a software has been written for custom analysis, design and optimization. Two benchmark frames have been used to verify the results obtained from the developed method. From the research, it has been observed that genetic algorithms provide an efficient way to navigate through the design search space while compromising for cost and imposed constraints simultaneously.

6.2 Achievements and Observations

6.2.1 Achievements

The research work has achieved the following major tasks:

1. A design method for the cost optimization of two and three dimensional frames using genetic algorithms has been developed. The GA method proves to be a reliable design alternative in contrast to traditional trial-and-error methods. In addition, comparisons with other non-heuristic methods have shown that genetic algorithms prove to be superior in speed and efficiency at navigating the design search space. In addition to providing constructible design solutions, the developed algorithm has taken shear and torsion effects, reinforcement schedules as well as sizing constraints into consideration.
2. The choice of genetic parameters has also been evaluated for test problems from which important patterns and observations have been made. In general, the choice

of these parameters is problem dependent, and should consider the size and complexity of the structural topology, nature of the objective function and computational time. For small problems, it was observed that median values for mutation probabilities and moderately large population sizes give acceptable results, whereas large population sizes coupled with small mutation probabilities have been seen to give stable simulations for large problems.

3. The computational complexity of the procedure has also been quantified using profiling techniques. It was observed that for small to moderately large problems, the procedure could be used to achieve local as well as global optimum solutions. For large problems, achieving global optima is unlikely as a consequence of an exponentially larger search space. However, the results obtained from such simulations could be used as starting iteration points for non-heuristic mathematical methods or could be coupled with other heuristic methods to achieve better results. One can also conclude that studying the space and time complexity of these algorithms sheds light on which procedures take up the most resource.
4. By using appropriate algorithms, the computation time and memory for the optimization procedure could be greatly reduced. It has been observed that the bulk of memory and computation time is consumed by structural analysis and the rest by the evaluation of the design algorithms. To remedy this, utilizing more efficient storage and matrix processing schemes has proved to increase the efficiency of the optimization procedure. This aspect challenges the traditional notion that heuristic methods are computationally inefficient.
5. Value encoding of individuals has proved to be well suited for structural optimization problems both in memory management and decode-encode schemes. The ease of changing key-value pairs also makes it a valuable contender to other encoding schemes such as binary encoding. The use of uniform crossover (in contrast to single point crossover) has achieved stronger variance between solutions, helping the algorithm efficiently navigate several regions of the search space.
6. Though the incorporation of shear design and variable reinforcement schedules has exponentially increased the size of the design space, it has allowed for economic estimations of cost to be feasible and constructible designs be achieved.

6.2.2 Observations

Two verification models have been constructed to test the viability developed algorithms. The first involved a comparison between the GA simulation and results from previous tests done using simulated annealing while the second was a direct cost comparison with

results obtained from ETABS. Comparisons show an improvement in cost with savings of up to 5% when using the genetic algorithm based optimization.

A second series of tests involved determining which combination of genetic parameters resulted in optimal results and achieved computational efficiency through swift convergence. Low mutation probabilities ranging from 2% up to 7% and population sizes ranging from 100 to 250 have been observed to achieve low fitness-low cost results. High values for the number of iterations, with a maximum of 200-250 have also been observed to attain stable and realistic results.

The last series of tests involved studying the computational complexity of the algorithm. Tests for memory consumption and overall computational times indicate that when the size of the structure increases, the total amount of RAM utilized and time taken increases exponentially. The consumption of 90% of these resources can be attributed to the structural analysis algorithm, while the rest could be divided up between determination of design actions and construction of interaction coordinates.

6.3 Future Work

The author of this research believes that the procedures discussed in this work could be improved and extended far beyond what they are now. The subsections that follow highlight some of these improvements.

6.3.1 Basics

Many of the restraints discussed in section 1.4 could be removed by any future researcher, to enhance the capabilities of the current platform. Traditional tools and features could also be incorporated into the software, such as loading options (loading patterns, case-based loading), stress visualizers, solver options, advanced material and section definition tools and many others. Important features such as dynamic and non-linear analyses can also be added with effort. Support could also be extended for optimizing using other materials such as structural steel, timber and composite materials. Cross-sectional geometries other than rectangles could also be added with relative ease.

6.3.2 Multi-objective Optimization

The objective that has been considered in this research was cost. However, with a few modifications of the GA fitness formulation, other objectives such as stress, weight and displacement could be imposed along with cost.

6.3.3 Structural Analysis

In this work, the stiffness method has been utilized for performing structural analysis. While this allows for a major performance boost, it follows that only 1D frame or truss elements can be processed with it. To incorporate a more general analysis procedure for handling 2D and 3D elements for the analyses of slabs, foundations and other types of structures, the finite element method is a good candidate. The major advantage in integrating FEM is that, since the stiffness solver is identical as to that used in the stiffness method, only the pre-processor(mesh generator, load vector analyzer...) and post-processor(Visualizer, Data Interpreter...) components need to be implemented. It is also worth to note that the superior performance of the solver implemented in this project can be harnessed for future endeavors.

On the flip side, structural reanalysis could also be taken as an additional measure for ensuring structures with similar cross-sections only get reanalyzed instead of getting fully analyzed, eliminating most of the time consuming procedures used in the structural analysis program.

6.3.4 Parallel Computing

Due to time constraints, HELIX was developed as a single-threaded application(i.e. only a single processor core is utilized for all its computational needs). However, with some modifications to the genetic optimization module, it can be made to utilize all available computing resources present in a PC/Workstation. This can drastically increase the speed of the simulation in that, instead of one instance of the application, n instances (for n available processor cores) of the simulation would be running simultaneously, cutting the total running time by a factor of n .

The daring researcher could also cook up a version of the algorithms that could run on one/multiple Graphics Processing Units(GPUs) instead of regular CPUs. This could allow for the program to run on hundreds of threads simultaneously, making it a lot faster than it currently is.

6.3.5 Custom Constraint Formulation

Currently, only hard-coded constraints and objective functions can be used by the optimizer to achieve results. If a change to these functions was desired, then the appropriate portion of the code would need to be changed and the source code recompiled. This is a grave inconvenience to any future user who wishes to make said changes but isn't in possession of the complete source code for HELIX.

Thus, the author is making preparations for a feature to be augmented in the current software architecture, which allows for the complete optimization formulation to be stated

symbolically and explicitly, and automatically be parsed and used by the optimization package. This would consequently require an appropriate symbolic manipulation library and an embedded language for interfacing with it.

Appendix A

Cost Evaluation for Benchmark

The following tables summarize the cost evaluation of the 2 bay by 5 story frame used in the research by Vidosa et. al. The cost computations are based on equations 4.4, 4.5, 4.6 and 4.7.

A.1 Volume of Concrete

Columns	A	B	Area	Volume	Beams	B	H	Area	Volume
C-1	0.25	0.25	2.25	0.1875	B-1	0.48	0.2	4.4	0.48
C-2	0.25	0.25	2.25	0.1875	B-2	0.5	0.2	4.5	0.5
C-3	0.25	0.25	2.25	0.1875	B-3	0.5	0.2	4.5	0.5
C-4	0.25	0.25	2.25	0.1875	B-4	0.51	0.21	4.65	0.5355
C-5	0.25	0.25	2.25	0.1875	B-5	0.54	0.22	4.9	0.594
C-6	0.25	0.45	3.45	0.3375	B-6	0.48	0.2	4.4	0.48
C-7	0.25	0.4	3.15	0.3	B-7	0.5	0.2	4.5	0.5
C-8	0.25	0.4	3.15	0.3	B-8	0.5	0.2	4.5	0.5
C-9	0.25	0.35	2.85	0.2625	B-9	0.51	0.21	4.65	0.5355
C-10	0.25	0.3	2.55	0.225	B-10	0.54	0.22	4.9	0.594
C-11	0.25	0.25	2.25	0.1875	Beam Total				3.30
C-12	0.25	0.25	2.25	0.1875					
C-13	0.25	0.25	2.25	0.1875					
C-14	0.25	0.25	2.25	0.1875					
C-15	0.25	0.25	2.25	0.1875					
Column Total				5.219	Total =3.30+5.219=8.519 m³				

Table A.1: Total Concrete Volume

A.2 Quantity of Reinforcement Steel

Beam ID	Top Reinforcement									Bottom Reinforcement					
	Base	ϕ	Volume*	Left	ϕ	Volume	Right	ϕ	Volume	Base	ϕ	Volume	Extra	ϕ	Volume
B-1	2	20	3141.6	-	-	0.0	1	25	490.9	3	12	1696.5	2	10	628.3
B-2	2	16	2010.6	1	10	78.5	2	20	628.3	3	12	1696.5	2	10	628.3
B-3	2	10	785.4	1	20	314.2	2	25	981.7	2	12	1131.0	1	20	1256.6
B-4	2	10	785.4	2	12	226.2	3	16	603.2	4	10	1570.8	1	16	804.2
B-5	2	10	785.4	1	16	201.1	2	25	981.7	4	10	1570.8	2	10	628.3
B-6	2	20	3141.6	-	-	0.0	1	25	490.9	3	12	1696.5	2	10	628.3
B-7	2	16	2010.6	1	10	78.5	2	20	628.3	3	12	1696.5	2	10	628.3
B-8	2	10	785.4	1	20	314.2	2	25	981.7	2	12	1131.0	1	20	1256.6
B-9	2	10	785.4	2	12	226.2	3	16	603.2	4	10	1570.8	1	16	804.2
B-10	2	10	785.4	1	16	201.1	2	25	981.7	4	10	1570.8	2	10	628.3
	Total		15016.81	Total		1639.91	Total		7371.75	Total		15331.0	Total		7891.7
* Volume is in $mm^2 - m$															

Table A.2: Flexural Reinforcement Schedule(Beams)

Column ID	Base	ϕ	Volume	Ext. A	ϕ	Area	Ext. B	ϕ	Area	Links	Volume
C-1	4	12	1357.2							21	593.8
C-2	4	16	2412.7							21	593.8
C-3	4	12	1357.2	2	12	226.2				21	593.8
C-4	4	12	1357.2				2	12	226.2	21	593.8
C-5	4	16	2412.7							21	593.8
C-6	4	12	1357.2				2	12	226.2	21	593.8
C-7	4	12	1357.2							21	593.8
C-8	4	12	1357.2							21	593.8
C-9	4	12	1357.2							21	593.8
C-10	4	12	1357.2							21	593.8
C-11	4	12	1357.2							21	593.8
C-12	4	16	2412.7							21	593.8
C-13	4	12	1357.2	2	12	226.2				21	593.8
C-14	4	12	1357.2				2	12	226.2	21	593.8
C-15	4	16	2412.7							21	593.8
	Total		24579.8	Total		452.39	Total		678.59	Total	8906.42

Table A.3: Reinforcement Schedule(Columns)

Beam ID	ϕ	Left Sp.	Volume	ϕ	Mid Sp.	Volume	ϕ	Right Sp.	Volume
B-1	6	0.1	452.4	8	0.25	351.9	8	0.3	301.6
B-2	8	0.2	427.3	8	0.25	351.9	8	0.3	301.6
B-3	10	0.3	471.2	6	0.15	311.0	6	0.15	311.0
B-4	8	0.2	427.3	6	0.15	311.0	8	0.3	301.6
B-5	6	0.15	311.0	8	0.3	301.6	8	0.3	301.6
B-6	6	0.1	452.4	8	0.25	351.9	8	0.3	301.6
B-7	8	0.2	427.3	8	0.25	351.9	8	0.3	301.6
B-8	10	0.3	471.2	6	0.15	311.0	6	0.15	311.0
B-9	8	0.2	427.3	6	0.15	311.0	8	0.3	301.6
B-10	6	0.15	311.0	8	0.3	301.6	8	0.3	301.6
		Total	4178.3	Total		3254.7	Total		3034.8

Table A.4: Shear Reinforcement Schedule(Beams)

A.3 Formwork and Scaffolding

Column ID	B	H	$A_{formwork}$	Beam ID	B	H	$A_{formwork}$	$A_{scaffold}$
C-1	0.25	0.25	2.25	B-1	0.48	0.2	4.4	2.4
C-2	0.25	0.25	2.25	B-2	0.5	0.2	4.5	2.5
C-3	0.25	0.25	2.25	B-3	0.5	0.2	4.5	2.5
C-4	0.25	0.25	2.25	B-4	0.51	0.21	4.65	2.55
C-5	0.25	0.25	2.25	B-5	0.54	0.22	4.9	2.7
C-6	0.25	0.45	3.45	B-6	0.48	0.2	4.4	2.4
C-7	0.25	0.4	3.15	B-7	0.5	0.2	4.5	2.5
C-8	0.25	0.4	3.15	B-8	0.5	0.2	4.5	2.5
C-9	0.25	0.35	2.85	B-9	0.51	0.21	4.65	2.55
C-10	0.25	0.3	2.55	B-10	0.54	0.22	4.9	2.7
C-11	0.25	0.25	2.25	Total			45.9	25.3
C-12	0.25	0.25	2.25					
C-13	0.25	0.25	2.25					
C-14	0.25	0.25	2.25					
C-15	0.25	0.25	2.25					
Total			37.65					

Table A.5: Formwork and Scaffolding

A.4 Cost summary

Item	Unit	Quantity	Cost/Unit	Total Cost(€)
Concrete	m^3	8.519	112.1	954.98
Reinforcing Steel	m^3	0.09233	10205	942.29
Beam Formwork	m^2	45.9	25.05	1149.80
Beam Scaffolding	m^2	25.3	38.89	983.92
Column Formwork	m^2	37.65	22.75	856.54
Total				4887.52

Table A.6: Cost Summary

Bibliography

- [1] Aga, A. A. A., & Adam, F. M. (2015), Design Optimization of Reinforced Concrete Frames, 74–83.
- [2] Bhowmik, R. (2008), Building Design Optimization Using Sequential Linear Programming. *Journal of Computers*, 58–64.
- [3] Beeby, A. W., & Narayanan, R. S. (2009), Designers' guide to eurocode 2: Design of Concrete Structures.
- [4] Bekiroğlu, S., Dede, T., & Ayvaz, Y. (2009). Implementation of different encoding types on structural optimization based on adaptive genetic algorithm. *Finite Elements in Analysis and Design*, 826–835.
- [5] European Committee For Standardization, (2004), Eurocode 2: Design of concrete structures - Part 1-1: General rules and rules for buildings, Brussels, Belgium.
- [6] European Committee For Standardization, (2004), Eurocode 2: Design of concrete structures - Part 1-2: General rules - Structural fire design, Brussels, Belgium.
- [7] Camp, C., Pezeshk, S., Cao, G. (1998), Optimized Design of Two-Dimensional Structures Using a Genetic Algorithm, *Journal of Structural Engineering*, Vol. 13, 551–559.
- [8] Christensen, P. W., & Klarbring, A. (2009), An Introduction to Structural Optimization, Springer, Linköping, Sweden.
- [9] Dede, T., Ayvaz, Y., & Bekiroglu, S. (2003), Optimization of Truss Structures using Value Encoding in a Genetic Algorithm, 1–15.
- [10] Fang, X. (2007), Engineering design using genetic algorithms.
- [11] Gaffney, J., Green, D., & Pearce, C. (2010), Binary Versus Real Coding for Genetic Algorithms: A False Dichotomy?, *ANZIAM Journal*, Vol. 51, 347–359.
- [12] Goldberg, D. (1989), Genetic algorithms in search, optimization, and machine learning (1st ed.). , Addison-Wesley Publishing Inc., Alabama

- [13] Gere, J., Weaver, W.(1980), Matrix Analysis of Framed Structures, 2nd ed., Van Nostrand Reinhold Company Inc., Wokingham, Berkshire
- [14] Gundersen, G. (2002), The Use of Java Sparse Arrays in Matrix Computation, Masters Thesis, University of Bergen, Norway
- [15] Guerra, A., & Kiouisis, P. D. (2006), Design Optimization of Reinforced Concrete Structures. Computers and Concrete, 313–334.
- [16] Holland, J., Langton, C., Wilson, S. (1992), Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, The MIT Press, Cambridge, Massachusetts.
- [17] Kaveh, A., & Jahanshahi, M. (2008). Plastic Limit analysis of Frames using Ant Colony Systems, 1152–1163.
- [18] Koza, J. R. (1998). Genetic Programming: On the Programming of Computers by Means of Natural Selection (6th ed.). The MIT Press, Cambridge, Massachusetts.
- [19] Kuhn, H. W., & Tucker, A. (1951), Nonlinear Programming, Proceedings of the Second Symposium on Mathematical Statistics and Probability, 481–492.
- [20] Lagaros, N. D., Papadrakakis, M., & Kokossalakis, G. (2002), Structural Optimization using Evolutionary Algorithms, 571–589.
- [21] Mahfouz, S. (1999), Design Optimization of Structural Steelwork: Design optimization of steel frame structures according to the British codes of practice using a genetic algorithm, University of Bradford.
- [22] Mosley, B., Bungey, J., Hulse, R. (2007), Reinforced Concrete Design to Eurocode 2, Palgrave Macmillan Publishing, Houndmills, Hampshire.
- [23] Patnaik, S. N., Guptill, J. D., & Berke, L. (1993), NASA Technical Report: Merits and Limitations of Optimality Criteria Method for Structural Optimization, 2–7.
- [24] Perea, C., Baitsch, M., Gonzalez-Vidosa, F. H. (1997), Optimization of Reinforced Concrete Frame Bridges by Parallel Genetic and Memetic Algorithms.
- [25] González-Vidosa, F., Yepes, V., Alcalá, J. , Carrera, M., Perea, C. , Payá-Zaforteza I. (2008) , Optimization of Reinforced Concrete Structures by Simulated Annealing, School of Civil Engineering, Universidad Politécnica Valencia, Spain
- [26] Rajeev, S., & Krishnamoorthy, C. S. (1998), Genetic Algorithm-based Methodology for Design Optimization of Reinforced Concrete Frames. Computer-Aided Civil and Infrastructure Engineering, Vol. 13, 63–74.

- [27] Reynolds, G. (2009), Genetic Optimisation of Structural Systems., University of Glasgow.
- [28] Torregosa, R., & Kanok-Nukulchai, W. (2002). Weight optimization of steel frames using genetic algorithms. *Advances in Structural Engineering*, 99–111.
- [29] Wang, B. P., & Chen, J. L. (1996). Application of Genetic Algorithm for the Support Location Optimization of Beams. *Computers and Structures*, 797–800.
- [30] Yang, X.-S. (2010). *Engineering Optimization* (1st ed.). Wiley & Sons, Inc, Cambridge, United Kingdom.
- [31] Yousif, S. T. (2013). Optimum Cost Design of Reinforced Concrete Columns Using Genetic Algorithms. *Al-Rafidain Engineering*, Vol.22.,123-141.