Chapter 1

Introduction

1.1 General

Building design can be divided into three parts: architectural design, structural design, and electrical and mechanical design. In addition, each part of building design may be divided into three stages: the conceptual design stage, the preliminary design stage and the detailed design stage. In architectural design, architects deal with the spatial aspects of activity, physical, and symbolic needs in such a way that overall performance integrity is assured. In structural design, structural engineers deal with design of structural integrity and stability. Because of the modern technology, present building architectures have become more aesthetic and complex. A complex shape of building results in a complicated building structure that is hard to be constructed. It is possible that changing a little architectural parts of the building results in big changes in design of the building structure. Thus, in all stages of architectural design, architects need to collaborate with structural engineers to produce the building design plan and construction documents.

In structural design, engineers first have to select a structural system that is suitable for the building by considering both architectural functionality and structural stability. There are many types of structural system used in practice (Lin and Stotesbury, 1981) such as rigid-frame systems, shear-wall systems, and tubular systems. A building's total structural system could be divided into two subsystems including vertical subsystems and horizontal subsystems. The horizontal subsystems carry floor loads through bending, and horizontal loads through diaphragm action. The vertical subsystems transfer loads from horizontal subsystems to foundations.

Conceptual structural design includes selections of vertical systems, lateral bracing systems, and floor systems. The vertical system includes columns or walls that transmit gravity loads from floors to foundations. The lateral bracing system is an interaction between columns (or walls) and the floor system to efficiently transmit lateral loads from wind or seismic loads to foundations. The floor system is the horizontal surface that transmits gravity loads on the floor including the floor self weight to foundations. The floor system or horizontal system can be designed as slabs, beams, or beam grid systems. There are various types of floor system such as flat-plate floors, beam-slab floors, joist-girder floors, and waffle floors (Lin and Stotesbury, 1981).

A flat-plate floor is a floor built from slabs without beams as shown in Fig. 1.1a. Flat-plate floors transfer floor loads directly to columns or walls with or without drop panels at columns. Flat-plate floors can be constructed of reinforced concrete or prestressed concrete. If reinforced concrete construction is used for flat slabs, their span can be up to 5 m to 8 m. Reinforced concrete flat plates always need the addition of

drop panels or capitals to increase the punching shear resistance at all columns. On the other hand, a prestressed concrete flat plate would have cables that hang over the top of the columns. These cables will help to carry much of the shear directly to columns. Furthermore, the compression produced by these cables onto the slabs helps to strengthen the slab against punching shear. Prestressed concrete flat plates can be relatively thin and span up to 8 m to 11 m between columns with a thickness of not over 15 cm to 22 cm.

Beam-slab floors are floors built from slabs and beams as shown in Fig. 1.1b. Beam-slab floors may be designed as one-way or two-way floors. The benefit of the beam-slab floor when the bay is nearly square is that beams can also be rigidly connected to the columns to form frames that resist lateral forces in both directions. A joist-girder floor shown in Fig. 1.1c is a variation on the beam-slab floor. Joist-girder floors comprise one-way slabs in perpendicular directions to joists. Joists transmit loads from slabs and their self weights to girders that support them. Actually the joist-girder

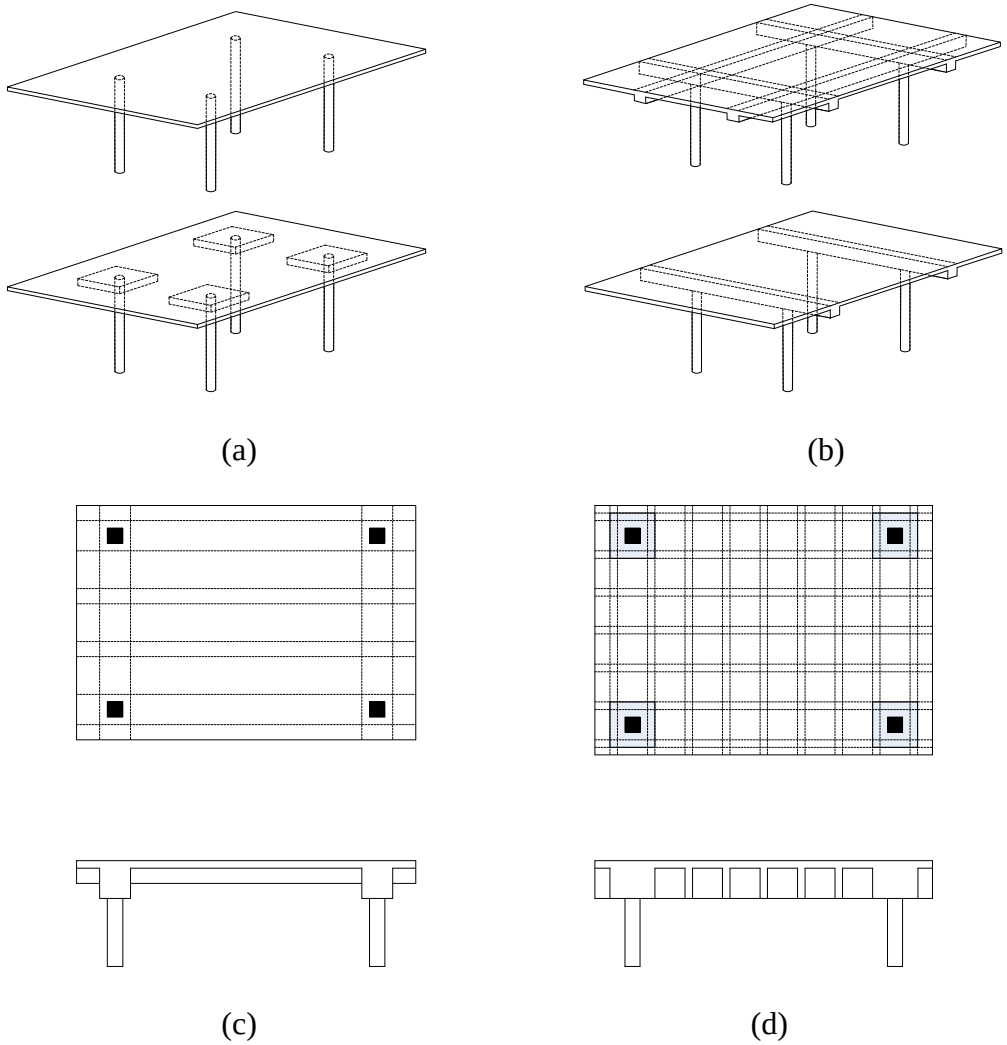


Fig. 1.1. Types of slab. (a) Flat-plate floor. (b) Beam-slab floor. (c) Joist-girder floor. (d) Waffle floor or beam-grid floor.

floor is a beam-slab floor when all slabs are one-way slabs, and girders are those beams supporting joists. Girders do not support slabs directly. The joist-girder floor is desirable when a thin concrete slab is required for various reasons such as economy or weight.

A waffle floor or a beam-grid floor shown in Fig. 1.1d is also a variation on the beam-slab floor. Waffle floors carry floor loads in two-way directions through their beams that are constructed in a grid. To be effective, waffle floors require that column bays be approximately square rather than rectangular. The slab panels over the beam grid can be very thin because the span is short and the load is carried in two directions. The concrete ribs or slabs can be reinforced or prestressed.

If flat-plate floors shown in Fig. 1.1a are used, the floor layout is already completed after obtaining column positions or column grids that are generally given as requirements from architects. Therefore, the remaining structural design of flat-plate floors is design of slab thickness. In case of waffle floors, patterns of beams and slabs are generally designed as a square grid. Therefore, layout design of waffle floors may be just design of beam spacing. Similarly, layout design of joist-girder floors is just to design of joist spacing, and girders are placed between columns. In case of general beam-slab floors in Fig. 1.1b, the first task of structural design of floors is layout design of beams and slabs. After obtaining layouts of beams and slabs, design of section details of the members can be performed. As a result, only general beam-slab floors in Fig. 1.1b are considered in this study.

Generally, preliminary structural design includes design of structural member layouts, estimation of structural member sizes, analysis of structures, and design of structures. Design of structural member layouts may be partially done in the conceptual design stage in collaboration with architects. Structural layout design starts from locating positions of columns or load-bearing walls depending on types of the vertical system used. Column layouts are often arranged in grids (Schodek, 2001) as shown in Fig. 1.2. Design patterns of columns or load-bearing walls inevitably affect patterns of

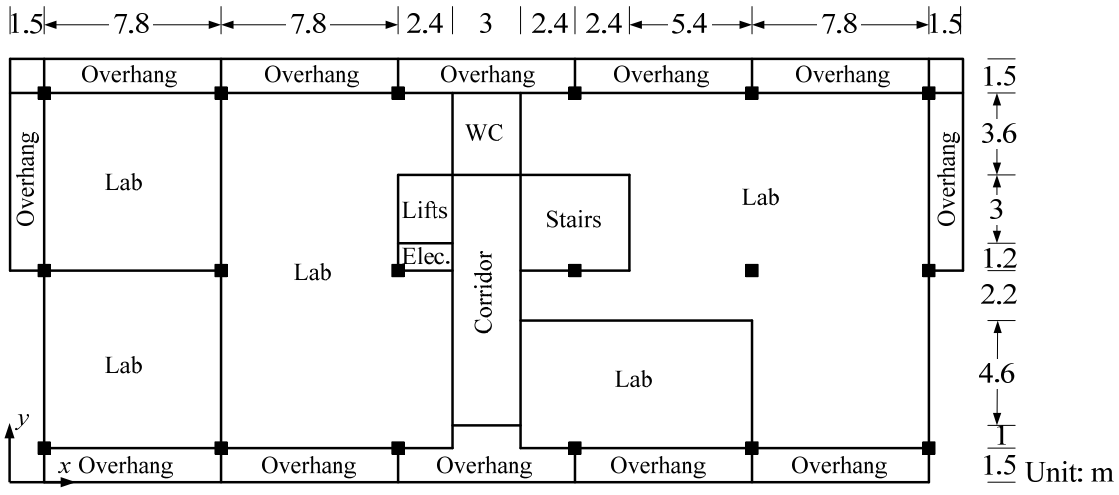


Fig. 1.2. An example of square column grid.

beams and slabs. In addition, positions of permanent walls, such as brick walls on the floors and the maximum size of slabs, always influence patterns of beams and slabs of the floors as shown in Fig. 1.3.

By considering beam-slab floors with prescribed column positions assigned by architects, design of structural layouts in the preliminary design stage is to design patterns of beams and slabs on floors. After obtaining the desired layouts, beam section details and slab section details can be determined next. Beam-slab layouts can be classified as one-way or two-way systems (Schodek, 2001). When a support grid consists of a square grid of columns, two-way beam-slab layouts can be obtained. However, beam-slab patterns may greatly be constrained by the requirement of beam positions to support permanent architectural walls that are used to partition floor areas to rooms. The space functions of the floors designed by architects naturally are unique for each building. Design of beam-slab layouts is heuristic. Consequently, it is hard to find a concrete algorithm for design of beam-slab layouts.

As mentioned earlier, design of vertical systems, lateral bracing systems, and floor systems of buildings naturally have some links to each other. Therefore, both conceptual design and preliminary design are possibly considered concurrently. In the relevant literatures, preliminary structural design and conceptual structural design are usually referred to as the same thing. Normally both architectural design and structural design are considered in three stages as stated earlier and their tasks can be performed in parallel. However, for small-scaled low-rise buildings such as houses, building design usually starts from preparing architectural floor plans, and then creating structural floor plans.

Nowadays computers have been widely used as a tool in design of buildings. There are plenty of state-of-the-art commercial programs assisting architects and

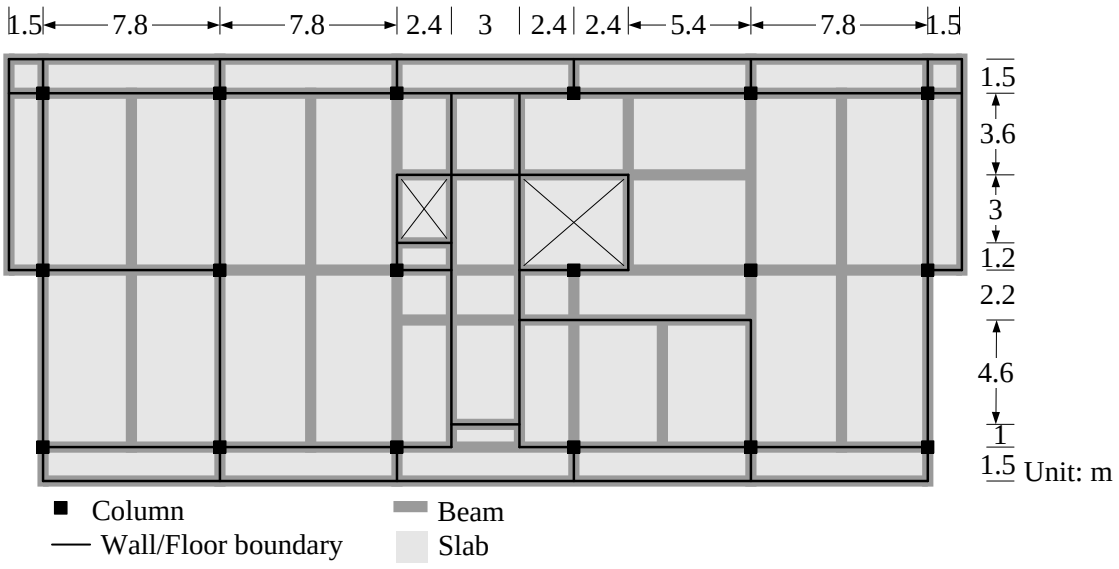


Fig. 1.3. An example of a beam-slab layout.

engineers in the detailed design stage. Examples are AutoCAD for architectural design of buildings, SAP2000 for structural analysis and design, and NASTRAN for general-purposed analysis and design. However, there are only some computer programs for conceptual design and preliminary design and they are mostly research programs. Examples include SEED-config (Fenves et al., 2000), BERT (Fuyama et al., 1997), CADRE (Bailey and Smith, 1994), CASECAD (Maher and Balachandran, 1994), BGRID (Sisk et al., 2003), and CADREM (Kumar and Raphael, 1997).

The early stage of conceptual design involves a balance of several requirements such as architectural requirements, structural requirements and mechanical requirements. Conceptual design is not suitable for procedural programming. It is because of its unclear design process. As a result, many researchers attempt to apply Artificial Intelligence (AI) to propose algorithms for conceptual design. Many past research computer programs for structural design of buildings fully support conceptual structural design but partially support preliminary structural design (Maher, 1984; Maher and Balachandran, 1994; Fenves et al., 2000). However, most of them do not take care of some important architectural constraints such as interior permanent walls.

1.2 Automated structural design of buildings

Structural design of buildings sequentially consists of conceptual structural design, preliminary structural design, and detailed structural design as shown in Table 1.1. The nature of tasks in structural design of buildings can be separated as tasks done by humans (architects and engineers) and tasks executed by computers. Some tasks can be easily done by humans while the others are more efficiently carried out by computers. Although tasks in conceptual structural design are heuristic, there are several successful systems using artificial intelligence for assisting engineers to select the best-concept structural system of buildings (Maher, 1984; Sabouni and Al-Mourad, 1997; Grierson and Khajehpour, 2002).

In structural building design, structural engineers usually have a little time to explore potential solutions resulting in precluding a qualitative evaluation of concepts. Therefore, conceptual structural design relies heavily upon design experience of the designers in selecting the most suitable structural system for the buildings. Conceptual structural design may be divided into two stages. The first stage of conceptual structural design is collaboratively done by two parties including architects and structural engineers. The second stage or preliminary structural design is exclusively accomplished by structural engineers. Automated conceptual structural design focuses on creating a structural system without the user interaction during the process.

The potential cost saving of the building project is largely affected by topology optimization that is generally more significant than sizing optimization. Therefore, this study is attempting to propose an automated intelligent system that supports preliminary structural layout design of buildings from pre-specified architectural design drawings. The automated system for structural design of buildings is the system that does not need engineers' intervention as shown in Table 1.2.

Table 1.1. Tasks in structural design of buildings.

Tasks	Nature of tasks		Usually done by	
	Heuristic	Computing	Engineer	Computer
Conceptual design				
Design of vertical systems	✓		✓	
Design of lateral bracing systems	✓		✓	
Design of floor systems	✓		✓	
Preliminary design				
Design of structural member layouts	✓		✓	
Estimation of structural member sizes	✓		✓	
Analysis of structures		✓	✓	
Design of structures		✓		✓
Detailed design				
Analysis of structures		✓		✓
Design of structures		✓		✓
Preparation of construction drawings		✓		✓

Table 1.2. Automated structural design of buildings.

Tasks	Done by engineer	Done by computer
Conceptual design		
Design of vertical systems		✓
Design of lateral bracing systems		✓
Design of floor systems		✓
Preliminary design		
Design of structural member layouts		✓
Estimation of structural member sizes		✓
Analysis of structures		✓
Design of structures		✓
Detailed design		
Analysis of structures		✓
Design of structures		✓
Preparation of construction drawings		✓

1.3 Optimization methods

This section is devoted to an introduction to optimization methods. Optimization methods are usually applied in solutions of mathematical problems by searching for the best solution in a prescribed solution space. In structural design of buildings, when the design problem is written as a mathematical problem, an optimization method can be used as a tool to solve for the solution. Analysis tools such as the finite element method can also be used to facilitate the optimization. In design of structural member sections, the optimization can be used for selecting the best section from a list of available member sections. An optimization method will also be employed in this study.

Optimization methods can be roughly classified as the gradient-based optimization methods and the non-gradient-based optimization methods or the search-based optimization methods. One of the advantages of the search-based methods is that they do not need to find the gradients, making them easier to be implemented in the computer program. The gradient-based methods include, for example, the Newton method, the steepest descent method and the conjugate gradient method (Deb, 1995). The gradient information is usually difficult to find even in simple problems. Consequently, the gradient-based methods may not be applicable to some practical optimization problems. There are many powerful algorithms for the simple search methods such as the exhaustive search method, the Fibonacci search method and the Golden Section search method (Deb, 1995). However, the use of the simple search methods is normally limited to one- or two-variable non-constrained problems. Complex constrained problems can be solved using the advanced search methods including, for example, Genetic Algorithms (GAs), Ant Colony optimization (ACO), and Simulated Annealing (SA).

GAs are global probabilistic search algorithms inspired by Darwin's survival-of-the-fittest theory. They have received considerable attention because of their versatile application to several fields. A GA starts from many points in the search space at the same time. These starting search points are usually selected randomly. Through the consideration of the fitness values of these search points, which are given based on their merit, and the randomized information exchange among the points, a new set of the search points with higher merit is created. The process is then repeated until the satisfactory result is obtained. Since the technique utilizes information from many search points at the same time, there is less chance for the search to be trapped in any of the local optimal points. Another distinguishing characteristic of GAs is that the algorithms work with coding of the parameter set not the parameters themselves. Generally, the binary code is used. Because of the discrete nature of GA coding, the algorithms are the perfect choice for those problems with discrete variables.

An ACO is developed for combinatorial optimization problems. The ACO mimics the foraging behavior of ant colonies in the real world. The efficient foraging behavior of ants is achieved by indirect communication between ants via the use of pheromone. It is well known that ants lay and follow pheromone trails. The ACO solves problems by simulating this natural behavior of ants while they find their shortest path to the food source. Artificial ants in the ACO algorithm will repeatedly walk on different paths which are considered as different solutions. The path or solution with

high quality will receive more pheromone by artificial ants. In the next round of walking, artificial ants will search for better solutions by looking for the path that has high level of pheromone. This process can be repeated until the satisfactory result is obtained.

An SA is inspired by an analogy between the physical annealing of solids (crystals) and combinatorial optimization problems. In the physical annealing process, a solid is first melted and the cooled very slowly, spending a long time at low temperatures, to obtain a perfect lattice structure corresponding to a minimum energy state. SA transfers this process to local search algorithms for combinatorial optimization problems. It does so by associating the set of solutions of the problem under consideration with the states of the physical system, the objective function with the physical energy of the solid, and the optimal solutions with the minimum energy states.

Optimization methods are extensively used in engineering design problems (Arora, 2004). Optimization methods have been also widely used in civil engineering problems. Examples are the applications of GAs to optimize truss structures (Rajan, 1995; Galante, 1996; Rajeev and Krishnamoorthy, 1997; Nanakorn and Meesomklin, 2001), the applications of ACO to optimize truss structures (Camp et al., 1998; Nimityongskul, 2004), and the application of GAs to optimize reinforced concrete beams (Coello et al., 1997; Griffiths and Miles, 2003).

1.4 Statement of problems

Structural floor layouts of buildings normally consist of beams and slabs. Layouts of beams and slabs greatly affect the final design of structural elements and subsequently the construction cost. Structural engineers practically use their engineering knowledge and experiences to create suitable beam-slab layouts that satisfy given architectural floor plans. Under the conventional wisdom, design of beam-slab layouts of buildings is a task that fully needs humans' involvement. In fact, all design tasks need different degrees of human intuition. Those design tasks that require a little of human intuition and can be systematically written as algorithms may be easily delegated to computers. In contrast, other design tasks that require a lot of human intuition and do not have clear algorithms cannot be done without designers' experiences. Table 1.3 shows a possible classification of structural design tasks in various design stages. The table aims to compare the degrees of heuristics and computing and also identify the roles of engineers and computers in these design tasks. It is quite apparent that the tasks in the conceptual and early preliminary structural design stages are heuristic and normally done by using engineers' experiences. On the other hand, those tasks in the later design stages are more computing oriented by their nature and, consequently, more suitable for computers. Although it may seem that some of the heuristic design tasks shown in Table 1.3 are not difficult and can be handled quite easily even by engineers in practice, these easy tasks unfortunately prevent the whole design process from being completely automated. In addition, since these heuristic tasks rely on engineers' experiences, their solutions will naturally come from the limited scope of each individual's experiences and may not include some good alternatives.

A number of attempts have been made to remove the hindrances to the development of complete design automation that are created by different heuristic design tasks. The utilization of artificial intelligence (AI) makes it possible to create a wide range of solutions for heuristic design tasks. Some popular branches of AI that are used to solve design problems include knowledge-based expert systems (KBESs), case-based reasoning (CBR) and genetic algorithms (GAs). Recently, many researchers have proposed computer systems to handle some heuristic tasks in structural design by using KBESs (Maher, 1984; Sriram, 1987; Balachandran, 1993; Tsakalias, 1994; Syrmakezis et al., 1996; Fuyama et al., 1997; Sabouni and Al-Mourad, 1997; Sacks and Warszawski, 1997; Syrmakezis and Mikroudis, 1997; Fenves et al., 2000; Sacks et al., 2000), CBR (Bailey and Smith, 1994; Maher and Balachandran, 1994; Kumar and Raphael, 1997), and GAs (Grierson, 1996; Park and Grierson, 1999; Grierson and Khajehpour, 2002; Rafiq et al., 2003; Sisk et al., 2003).

Automation of beam-slab layout design is actually an ill-defined problem, meaning that it is even not clear how to express explicitly the objectives of the layout design process. As a result, when a GA is used to solve the problem, the main issue becomes how to represent a layout design problem as a mathematical optimization problem. The quality of any proposed new GA for automated floor layout design therefore depends on how the representative optimization problem is written. This study aims to develop a new GA for beam-slab layout design. The primary input of the algorithm is an architectural floor plan with given positions of columns and walls. Before the development of the proposed GA can be done, a new coding scheme for beam-slab layouts must be developed. After that, the beam-slab layout design problem has to be written as an optimization problem. This is done by establishing appropriate objective and constraint functions for the problem. To make the proposed GA simple

Table 1.3. Roles of engineers and computers in structural design of buildings.

Process	Process characteristic		Role of engineers	Role of computers
	Degree of heuristics	Degree of computing		
Conceptual design				
Design of vertical systems	High	Low	Processor	Helper
Design of lateral bracing systems	High	Low	Processor	Helper
Design of floor systems	High	Low	Processor	Helper
Preliminary design				
Design of structural member layouts	High	Low	Processor	Helper
Estimation of structural member sizes	High	Low	Processor	Helper
Analysis of structures	Low	High	Helper	Processor
Design of structures	Low	High	Helper	Processor
Detailed design				
Analysis of structures	Low	High	Helper	Processor
Design of structures	Low	High	Helper	Processor
Preparation of construction drawings	Low	Low	Processor	Helper

and, as a result, more attractive, the simple GA (Goldberg, 1989) is employed as a core algorithm for the development of the proposed GA. To show the validity of the proposed algorithm, the algorithm is used to design beam-slab layouts of several example architectural floor plans.

1.5 Objectives

1) To develop a genetic algorithm for beam-slab layout design.
2) To establish an appropriate coding scheme, an objective function, and constraints for the proposed genetic algorithm.

1.6 Scope of this study

As mentioned earlier, this study aims to create a GA for beam-slab layout design of buildings. The algorithm will be implemented in C++. The scope of the study is as follows:

1) Only beam-slab floors with prescribed positions of columns and walls are studied.
2) Only rectangular and rectilinear floors are considered.
3) Construction costs are not considered.
4) The simple GA is employed as a core search algorithm.
5) Roulette wheel selection is used in the GA reproduction process.
6) One-point crossover and bitwise mutation are used in the GA generation process.
7) An adaptive penalty with bilinear scaling is used to handle design constraints.
8) Elitism is used to keep the best individual.

Chapter 2

Literature Reviews

In structural building design, beam-slab layout design is a task in preliminary structural design. Conceptual design and preliminary design are frequently considered as the inseparable design process. Recently, there are attempts to concurrently handle conceptual and early preliminary structural design that are heuristic by using computers. Many past related research works employ techniques of Artificial Intelligence (AI). In this chapter, the literature on conceptual structural design and preliminary structural design that employ AI techniques is presented below.

2.1 Conceptual structural design

Conceptual structural design is not a simple process in that it requires considering both quantitative and qualitative criteria. It is the unstructured process involving the application of the designer's experience and judgment in a qualitative manner to arrive at a number of alternative best-concept design scenarios. Many techniques of AI are employed to handle the conceptual structural design of buildings. Examples are knowledge-based expert systems (KBESs), case-based reasoning (CBR), and Genetic Algorithms (GAs). AI techniques assist engineers in exploring conceptual design alternatives and making design decisions by performing systematic search over a space of possible solutions under constraints.

A KBES is an interactive system consisting of a knowledge database and an inference mechanism. The knowledge database is a collection of general facts of the problem domain. The inference mechanism is an engine that carries out the reasoning whereby the expert system reaches its solution. CBR involves finding solutions to new problems through reusing available good solutions to similar past problems. CBR consists of three main processes, i.e. representation of cases, indexing and retrieval of cases, and adaptation of cases for the current problem. A GA draws an analogy from the biological evolution. It uses codes to represent solutions and improves the solutions by using genetic operators, i.e. reproduction, crossover, and mutation.

From Table 1.1, the first design stage is conceptual structural design, which mainly includes selection of structural systems. There are many computer-based techniques employing KBESs, CBR or GAs in selecting the most suitable structural systems of buildings. Some of these techniques perform not only conceptual structural design but also, at the same time, architectural layout design.

In the 1980s and 1990s, many researchers had proposed an interactive system employing AI for assisting engineers to select the structural system. Many AI researches concerning conceptual structural design of buildings mostly utilized KBESs and CBR.

For example, Maher (1984) had proposed an interactive KBES for conceptual and preliminary structural design of hi-rise buildings, called HI-RISE. HI-RISE performs conceptual structural design by generating feasible alternatives for two functional systems: the lateral load resisting system and the gravity load resisting system. The outputs of HI-RISE are alternative structural systems without complete solutions of preliminary structural design. HI-RISE represents the design information in a network of schemas. The structure of the network is predefined by schema templates stored in the knowledge base. In addition, HI-RISE performs an approximate analysis and preliminary proportioning to determine the feasibility of the alternative. Sriram (1987) employed a KBES with techniques that are similar to those of HI-RISE to solve similar problems.

Sabouni and Al-Mourad (1997) had developed a knowledge based expert system, called TALLEX, for preliminary design of tall buildings. The optimum structural system for tall buildings is selected based on the virtual number of stories. The main idea of the virtual number of stories is the taller the building, the more efficient the required structural system. The virtual number of stories is the actual number of stories of the building plus the additional number of stories that is converted from other factors such as the geometry of the building, the intensity of the design wind and earthquake loads, and the soil conditions. The additional number of stories is calculated from the multiplication of the actual number of stories and the summation of the numerical value of three factors: the load parameter, the material parameter, and the geometry parameter. The maximum addition number of stories is 0.4 times the actual number of stories. The numerical value of the load factor is based on the input values of the importance, the wind speed, and the seismic zone. The value of the material factor depends on the bearing capacity, the corrosion protection, and the fire protection. The value of the geometry factor depends on the possibility of providing enough space for a shear wall, the building symmetry, the possibility of providing closely spaced columns in the perimeter, and the opening requirements in the shear wall. The optimum structural system is selected from the available choices that are pre-defined in the system using the virtual number of stories and the rate of acceptance for each structural system that is based on the actual number of stories.

Balachandran (1993) had developed a knowledge-based optimum design system, called OPTIMA, coupling the symbolic processing and numerical computation. The proposed system comprises five major components: user interface, semantic interpreter, problem formulator, problem recognizer, and problem solver. The system is mainly developed for optimal structural design. Although the system is a domain specific system for optimal design of structures, it is a problem independent system in that, for example, the system can solve optimization design problems of floor layouts, and sizing optimization design of beams. The user interfaces are as follows: menu selection, question-answering, pseudo-English-phrase input, and graphical input. The problem formulator module carries out the task of formulating a design problem as a canonical optimization model. The problem recognizer performs the selection of an optimization algorithm which is suitable for the problem. There are many implemented optimization algorithms available in the system for the designer to select, such as linear programming, and nonlinear programming.

Maher and Balachandran (1994) had implemented the system, called CASECAD, for conceptual design of hi-rise buildings employing CBR and a multimedia database. The information of design cases are largely derived from structural drawings and project reports. The design prototype stored in the case base has many attributes that are categorized as the function, the behavior, and the structure. The visualization of structural data of the design case can be stored in a DXF file and can be display in general CAD programs. Two kinds of indexes are employed: the category indexes and the attribute indexes. The best similar cases are retrieved using the nearest neighbor technique. The similarity of a retrieved case to the problem is measured by the number of matching features in their specifications.

As mentioned above, there were many researchers using KBESs and CBR to develop the conceptual structural design of buildings. However, many systems employing KBESs have the difficulty of combining the knowledge of human experts with heuristic search in a computer system. Recently, many researchers have attempted to use genetic algorithms (GAs) in conceptual structural design and preliminary structural design. The literature concerning preliminary structural design is separately presented in the next section.

Mathews and Rafiq (1995) had used GAs in conceptual design of concrete buildings and had proposed the system for generating the grid system and selecting the floor type. The objective of the system is to maximize the size of clear functional spaces, representing lettable floor areas, by minimizing the number of bays that are partitioned in each direction of the plan layout by the structural grid lines. The search space is limited to medium-rise reinforced concrete office buildings without considering the lateral forces. By employing the heuristic knowledge regarding cost and functionality, the proposed system intentionally optimizes the design of the structural grid layouts. The design knowledge that the system employs to evaluate and appraise the generated floor layout are as follows. First, floor beams and slabs will span over columns to form a floor system. Second, the economic limit of span length is set to 4m to 8m for the RC floor system. Third, columns are arranged in lines in two orthogonal directions for the practical purpose. Finally, the structural grid lines are preferred to be regularly spaced for the cost-efficient and easier construction purposes. The design variables include the type of the structural system and the building dimensions. The dimensions of each bay are limited to whole-meter or half-meter intervals. Beams and columns are approximately designed by interpolating the required amount of steel from the existing design charts implemented in the system. The design chart is prepared in terms of the relationship between the beam span, loaded width, and the required amount of longitudinal reinforcement. The objective function is defined as follows:

$$\min(C) = (n-1)\sum_{i=1}^{m} x_i^2 + (m-1)\sum_{j=1}^{n} y_j^2 \tag{2.1}$$

where C is a measure of the number of bays and their distribution, to be minimized. In addition, x_i is the width of the ith bay in the x direction while y_j is the

width of the *j*th bay in the *y* direction, and *m* and *n* are the number of bays. The quadratic penalty functions defined as the factored standard deviation of x_i and y_j are also applied to encourage solution with bay widths in the economic range. Each bay width is encoded as a separated binary substring. After obtaining the optimized structural grid, the system has a successive process to determine the most economic structural system by using the estimated pre-set unit costs. The available structural systems in the system are classified as the concrete frame and the steel frame.

Grierson (1996) proposed a system for the conceptual structural design of buildings. The system employs both genetic algorithms (GAs) and a neural network (NN). The system is a semi-automated system in that computational techniques are employed to generate conceptual design while their evaluation depends on the user. Design variables are defined by the indices and then are coded as a binary string. The design variables include the building function, the building shape, the floor type, the vertical system, the lateral system, the lateral bracing and the foundation type. The proposed system starts with randomly creating a solution and then lets user to evaluate and then trains NN. After that the GAs are applied to generate the new solution, and then the user has to evaluate them again. If the current best solution satisfies the user, then the system will terminate. However, if the current best solution does not satisfy the user, then the user has to modify attribute's the current solution and then NN will be trained and GAs will be applied. The system will repeat the above process until the best solution satisfies the user.

Grierson and Park (1996) had employed GAs to propose a computer-based approach to conceptual topological design of a building framework. The proposed system has the limitation that it can search only for the optimal uniformly structural grid of the square-shaped building. In addition, the beam-slab-column structural system is only studied. The proposed system attempts to minimize the cost function consisting of costs for land, columns, and beams. The cost function has factors reflecting the variability of beams with slabs, of columns, and of land costs with changes in the span length and the site areas. By keeping the total required floor area constant, the number of bays and stories will be optimized. The square floor plan dimensions are calculated by the square root of the ratio of the total required floor area to the number of stories.

Park and Grierson (1999) had employed the Pareto-optimal multicriteria genetic algorithms to handle the conceptual structural grid layout design of rectangular-shaped buildings. Two objective functions are investigated including minimizing the estimated construction cost (f_1) and maximizing the flexibility of usable floor space (f_2). The estimated construction cost is composed of the floor system cost, the column cost and the land cost. The flexibility of floor space usage is quantitatively defined as the minimization of an exponential function that relates tributary load area to the spacing of columns. Five basic design variables comprise the two plan dimensions, the number of stories and the two numbers of bays. Five related design variables are the floor type system, the floor plan type, the two numbers of eliminated bays, and the starting floor number for any one of the floor plan types. In their study, the building may have one or two different floor plan types over its height. Four floor plan types are available. There are two types of constraints applied in their study: hard constraints and soft constraints.

Hard constraints are composed of the specified maximum dimensions of the building site, the specified building height restriction and the number of eliminated bays in the *x* and *y* directions required to create any of the floor plan types. A soft constraint is the required total floor area for the building. The soft constraint may be violated to some extent. However, the solution that violates any of hard constraints is not allowed to survive as a viable design in the search to find the Pareto-optimal design set. The fitness evaluation is based on a distance metric related to the Pareto-optimal set. The relative distance D is calculated using the Euclidean norm as shown in Eq. (2.2)

$$D(\mathbf{x}) = \min\left[\sum_i \left\{\frac{f_i(\mathbf{x}_j^0) - f_i(\mathbf{x})}{f_i(\mathbf{x}_j^0)}\right\}^2\right]^{0.5}, \quad i = 1,...,Q;\ j = 1,...,P \tag{2.2}$$

where, for the *i*th objective criterion, $f_i(\mathbf{x}_j^0)$ is the objective function value for Pareto design $\mathbf{x}_j^0$ and $f_i(\mathbf{x})$ is the objective function value of the design $\mathbf{x}$. The fitness value of the current population F is calculated using Eq. (2.3), i.e.

$$F(\mathbf{x}) = F^0 - D(\mathbf{x}) \tag{2.3}$$

where F^0 is the shared fitness of the current-generation Pareto-optima design set. For the initial population, F^0 is arbitrarily positive value that is large enough to ensure that F is not negative. For later generations, F^0 is the summation of the previous F^0 and D (using the previous generation $f_i(\mathbf{x}_j^0)$). In case that there is no new Pareto design, the F^0 value of the Pareto design set is retained. After assigning design fitness, the remaining process of the multicriteria genetic algorithms is the same as the simple GA. Each solution is encoded as a binary string. Each individual comprises 26 genes. The 4-bit substring represents the 16 types of bay size parameters. The 8-bit substring represents the numbers of bays in *x* and *y* directions. The 6-bit substring represents the numbers of eliminated bays in *x* and *y* directions. The 4-bit substring represents the starting floor number for any one of the floor plan types. The 2-bit substring represents the four floor plan types, and the 2-bit represents the four floor systems. Grierson and Khajehpour (2002) solved similar problems also by a multicriteria GA with Pareto optimization. Their algorithm uses three objectives, namely minimizing the capital cost, minimizing the annual operating cost, and maximizing the annual income revenue. Binary-string design variables employed include building dimensions and structural system types.

2.2 Preliminary structural design

The goal of preliminary structural design is to find a feasible arrangement of structural elements in space that are able to transfer, safely and efficiently, loads to the ground. The outcome of this stage is an initial description of the structural system in terms of the layout of its members with associated cross-sectional properties, connectivity and materials. However, the preliminary stage of a truss structure is probably performed simultaneously with the structural optimization.

The second design stage in Table 1.1 is preliminary structural design whose tasks include design of structural member layouts, estimation of structural member sizes, analysis and design of structures. For design of beam-slab layouts, the main purpose is to create beam-slab layouts for given architectural floor plans. Recently, there have been many researchers that try to solve problems of beam-slab layout design by using KBESs, CBR and GAs. For example, Tsakalis (1994) proposed a knowledge based system called KTISMA for synthesizing the structural model of asymmetrical skeletal buildings made of reinforced concrete. The KTISMA system is an approach to create a beam and slab layout for the ceiling of an apartment. The only input required by the system is a proper description of the architectural floor layout. The system creates a solution using a simulation whereas the structural model is iteratively altered and tested until a “good” solution is reached. The system acting on a common blackboard employs many different knowledge sources such as an architectural knowledge considering the structural model from the aesthetic perspective, and a static knowledge considering the evaluation of the merit of spans and indirect support. The system employs two heuristic strategies: the local level heuristic and the global level heuristic. The global level heuristic is used for producing more solutions in case that the solutions obtained from the local level are not good overall solutions. At the local level of reasoning, the system starts with an arbitrary quadrangle evaluated by the knowledge sources of the system such as the span’s length evaluation, the beams’ appearance evaluation, and the beam’s indirect support evaluation. At the global level of reasoning, the system employs a variation of the backtracking algorithm using a graph path that forms a feasible solution.

Syrmakezis et al. (1996) proposed a user-interactive commercial program called VK.EXPERT for preliminary seismic design of reinforced concrete multistory buildings using the knowledge-based expert system. The program consists of five modules: DIRECTOR for controlling program flow, DRAW for entering the input data and displaying the results based on a small CAD kernel, MAKE for generating alternative structural system configurations satisfying all constraints prescribed by the user, SIZE for providing an initial sizing of members (columns-beams-slabs), ASSESS for performing the structural analysis. The most important module is the MAKE module that generates a set of feasible configurations of the structural system. MAKE employs two basic geometric shapes, rectangles and triangles, for matching the structural system outline to the user-input architectural outline of the building on each floor level. Columns and beams are placed in the limits of currently defined architectural spaces. The structural layout results from combining existing architectural spaces. MAKE generates alternative configurations by modifying accordingly the influence of each design parameter based on fuzzy logic. The criterion for validating the generated solution is based upon the user-definable optimality of the dimensions of generated

slabs while the program provides reasonable default value based on the overall building dimensions. The program locally searches for the best solution and uses the backtracking technique in case the local optimal is not a good overall solution. The improved research-based version of the VK.EXPERT system is ERDES (Syrmakezis and Mikroudis, 1997).

Sacks and Warszawski (1997) and Sacks et al. (2000) proposed a project model for an automated building system. The system represents the project information by a tri-hierarchical project model—spaces, assemblies, and activities. The system employs knowledge-based modules to automatically generate information for the design and construction planning of a building project. The system employs the object-oriented technique to invent the intelligent parametric templates of building layouts and work assemblies. A prototype program had been implemented on the AutoCAD platform using AutoLISP++ extension. The proposed system can be separated into 7 stages. Stage 1 is the input of the user's requirements such as the location of the building, the function of the building, and its service area. Stage 2 is the generation of the preliminary design information called the design brief, for instance, the floor areas, performance specification in terms of loads. Stage 3 is the conceptual design including the building footprint, elevation partitioned into floor, the general function, the permitted building height. Stage 4 is the general design involving the layout of each floor, the division of each floor into its "secondary spaces" such as rooms, corridors, and service areas. Stage 5 is the general design involving selection the main work assemblies. The assemblies—the structural system, the exterior envelope, the sanitary systems etc.—are the systems which the building employs to perform its functions. Stage 6 is the detailed design involving each of the work assemblies. Stage 7 is the construction planning—the construction schedule, cost estimate and budget. The proposed system has three types of knowledge—the knowledge for the project model, the knowledge modules for the procedure, and the knowledge for the external data such as costs. The value of each parameter in a template is determined by application of knowledge-based procedures to the context data. These procedures form an integral part of the template and are associated with them in the same way as methods are associated with object classes in the object-oriented paradigm, and they may contain rules, algorithms and functions. Templates include features of all the knowledge types mentioned before: object classes, procedures for their installation, and the data used for this purpose. The proposed system is an interactive system and limited to rectangular-shape multistory buildings with a uniform floor type of RC ribbed slab on beams. The system does not have any automated "learning" capability, although this is desirable in a full system.

Bailey and Smith (1994) had developed a case-based reasoning approach to be integrated with CADRE, the existing CAD system. The approach focuses on dimensional and topological adaptation of geometric methods of existing building information stored in the case base to find solutions for the new design problems. The proposed approach is a user-interactive system working on CADRE. The process of the system can be briefly explained as follows. The user selects an appropriate building case from the case base, then, the selected case is parameterized by CADRE and initial constraints describing both structural and architectural characteristics and their relationship to each other are generated automatically. The user also defines additional constraints concerning the new design problem. CADRE then attempts to solve the

problem using the defined constraints and the adaptation process. The generated constraints used in the system are, for example, span-depth ratio for beams and the spacing of structural frames and the overall length of a building. If the solution is not found after applying the constraints, the user can let the system do the topological adaptation process to solve the problem. It means that the solution will be changed considerably in details. Commonsense domain-independent rules are employed. For example, if longer spans are preferred, the number of frames must be less. Domain knowledge is employed for deciding when the type of construction needs to be changed. For example, a flat slab floor is suitable for spans up to 8 m, but longer than that, a beam-slab floor is preferable. In case that the room layout needs to be changed to satisfy the architectural constraints, the algorithm for generating alternative arrangements of rectangles is employed.

Kumar and Raphael (1997) had employed the case-based reasoning technique to propose the system called CADREM for the conceptual design of structural layouts of a building. The system represents cases for different tasks by design methods containing the sequence of steps used in individual design problems. The design methods are represented using a data structure called method-object. Moreover, the retrieval process is organized as a process based on examples called RBEX. By using individual examples of retrieval, more general retrieval methods (RMs) are generated heuristically. The CADREM processes can be briefly explained as follows. First, the top level design task and a set of variables describing the problem specification are sent to the RBEX. After that, the similar cases are retrieved using RMs. Some tasks of the retrieved method are set as the low level method which would directly give partial solution to the problems. The remaining subtasks are iteratively solved by RBEX until the low level method is obtained, giving more partial solution. Finally, all the partial solutions are integrated into the final solution. The contents of a case are (1) a set of preconditions for which the design is generated, (2) the definition of the design task, and (3) the method used to arrive at the design. The top level task consists of four subtasks: (1) arriving at a pattern, (2) designing a horizontal spanning system, (3) designing a vertical support system and (4) designing a lateral load resisting system. The arriving at a pattern process has the two following subtasks: generating rows of blocks and combining rows of blocks. The employed method for generating rows of blocks is combining blocks in the horizontal direction. CADREM has many alternatives for the horizontal spanning system such as a one-way slab system, a two-way slab system, a waffle slab system, and a truss system. The method for design of a floor system has two subtasks: first, selecting the type of floor system and second, designing the selected floor system. The selection of the type of the floor system has four steps: (1) checking the aspect ratio of the individual blocks, (2) checking whether the dimension of the blocks are suitable for one system or the other, (3) checking the continuity across the blocks, and (4) considering construction details.

Fenves et al. (1995) and Fenves et al. (2000) had proposed the conceptual-structural-design sub-module of the **S**oftware **E**nvironment to support the **E**arly phases in building **D**esign (SEED-config). The proposed system is one of three sub-modules of the SEED, an interactive computer program for building design. This proposed system represents design information by exploiting the hierarchical nature of building description and employing an object-oriented data model. The system represents every

building element as a building unit, a general container that encapsulates the entity's geometry, taxonomy, properties, relationships, and the design knowledge that generates it. The system encapsulates design knowledge in a set of technology nodes. A technology node has three basic functions: the prerequisite specification (e.g. Waffle slab has prerequisites: Two-way and Concrete), the applicability specification (e.g. Two-way slab is applicable if the slab has square dimensions and is supported on four sides), and the action specification. The system has a case-based reasoning system supporting the storing and retrieval of past solutions and their adaptation to similar problem solutions. Case-adaptation in the system relies on the technology nodes. If the technology node's antecedents are satisfied, they are applied to the current building entity. The adaptation process continues down the sequence of retrieved nodes in the case until an applicability requirement of a technology node is violated or until the end of the path is reached.

Because of the efficiency of GAs in finding good solutions from large search spaces, GAs have become a popular technique for structural design. Applications of GAs in structural design had been initially found in truss optimization (Goldberg, 1989; Rajeev and Krishnamoorthy, 1992; Adeli and Cheng, 1994). Later on, GAs have been used to solve more diversified types of structural design problem (Coello et al., 1997; Kameshki and Saka, 2001; Camp et al., 2003; Griffiths and Miles, 2003). For floor layout design problems, GAs have been mostly used for design of architectural floor layouts (Gero and Kazakov, 1998; Jo and Gero, 1998; Michalek et al., 2002; Michalek and Papalambros, 2002; Bausys and Pankrasovaite, 2005). They are rarely used for beam-slab layout design and there are only few related researches.

For example, Rafiq et al. (2003) proposed an interactive conceptual building design system based on the structured genetic algorithm technique, called shortly SGA, to represent the structural system of buildings. The search space is limited to the uniform-grid rectangular-shaped building. The chromosomes of SGA contain two types of genes: (1) parameter genes which represent design parameters and (2) switch genes used for activating or deactivating different segments of a chromosome. The advantage of using the SGA is that the user can control the system to find the best solution in the predefined search space. For example, the user can still find out what will happen if concrete is used by deactivating the steel-frame switch-gene and thereby forcing the GA to consider the concrete option only. The required input data are the required net lettable floor area, the rate for rent, the imposed loading on the floor, the cost of land, and the required service life of the building. The problem is formulated for the minimization of the profit determined by subtracting the capital cost if the building structure from the total income.

Sisk et al. (2003) had proposed a user-interactive GA-based decision support system for the conceptual design of multistory office buildings, called BGRID. The required input data are divided into four parts as follows: (1) the plan dimension and number of floors, (2) the site location and planning restriction such as the maximum allowable height, and the minimum floor-to-ceiling height, (3) the location of core and atria, and (4) the dimensional constraints. The design solutions consist of the structural system, the environmental strategy, and the uniform grid system. The design variables are the grid, the structural-service integration strategy, the environmental strategy, and

the floor-to-ceiling height. The search space is limited to the rectangular-shaped steel buildings with a specified number of stories and dimensional constraints. The design variables are coded as real-coded strings. The selection technique employed in BGRID is the standard fitness method (Bradshaw and Miles, 1997). The standard fitness method ranks the population using the raw fitness values and then allocates predetermined slot size on the roulette wheel. The slot sizes are computed using the standard deviation of the normal distribution. Each individual in the population can be divided into three paths including the substring representing the *x* coordinates of the columns, the substring representing the *y* coordinates of the columns, and the substring representing the floor-to-ceiling height. The crossover operator is applied individually to each of three substrings of an individual. The system has three objective functions including minimizing the cost, maximizing the clear span and maximizing the use of natural resources. In the evaluation process, the user has to select the weighting important factor for such three objectives. The objective function for minimizing the clear span is defined as the ratio of the average span to the largest span. The penalty method is used when the generated solution does not pass constraints. The constraints are composed of three parts as follows: (1) the height restriction, (2) The compatibility of the generated structural system with respect to the span length, and (3) the uniformity of the grid. The fitness function that is to be maximized is in the following form:

$$f_{iobj} = \frac{f_j - f_{ibad}}{f_{igood} - f_{ibad}} \tag{2.4}$$

where f_{iobj} = the fitness of the individual component; f_{ibad} = the value of the worst individual generated up to that point; f_{igood} = the value of the best generated up to that point; and f_i = the value of the evaluated parameter for the individual within the current population.

Shaw et al. (2008) have recently proposed a method of determining column layouts for orthogonal buildings using the sweep line algorithm coupled to an adjacency graph. The problem include framed buildings that are defined as buildings consisting of columns and beams with slabs to support the floors. The proposed algorithm is called OBGRID (Orthogonal Building GRID). In some ways, OBGRID can be thought of as an enhancement of BGRID (Miles et al., 2001). The algorithm only devises layouts where the columns are arranged in rectangular grids. The user has to provide the dimensions of the boundary, the location and sizes of any atria, the maximum, allowable height and to specify the total number of stories. By using a sweep line algorithm, an orthogonal floor plan is partitioned into rectangles that are subsequently assigned column spacing, and an adjacency graph is constructed. The representation focuses on aligning columns in rows. The representation uses a separate string to store the x and y coordinates. The variable length genome is used. Each genome is divided into three distinct sections with real number encoding. Sections 1 and 2 of the genome contain values of column spacing in the *x* direction and values of column spacing in the *y* direction. Section 3contains the height parameters. In the GA operation, the mutation

operator selects a gene from sections 1 or 2, and then the gene is replaced with a randomly generated value between 0 and the maximum x or y dimension. After the insertion of the mutated value, if required, the genome is sorted. If a gene from section 3 is chosen, it is mutated as normal. Single point crossover is used on each of the genome's three sections. The reproduction process employs a conventional tournament selection technique. OBGRID applies the same fitness function as BGRID to each individual partition's genome and aggregates the results. A quadratic penalty function is employed to compute the augmented fitness. For orthogonal buildings, having selected the individual to mutate, the mutation operator randomly chooses one partition and applies the mutation procedure. Having mutated its genome, the section is placed back into the building and all adjacent sections are updated to prevent column alignment mismatches. Once recombination has been accomplished, the altered sections are reinserted into the building and all other adjacent partitions updated. The algorithm has been tested on a number of examples. The scientific rigor of the evaluation is limited by the lack of test cases; however visual inspection using the heuristic of reasonably consistent column spacing provides a good measure of performance. This shows that the method works well except for complex buildings which result in a lot of partitions.

Chapter 3

Genetic Algorithms

Genetic algorithms (GAs) belong to a class of stochastic search methods. The concept of GAs is based on Darwin's theory of natural selection. In addition, GAs operate on a population of solutions at any one time. GAs are efficient and broadly applicable global search procedures especially for engineering optimization problems (Rajeev and Krishnamoorthy, 1992; Mathews and Rafiq, 1995; Rajan, 1995; Galante, 1996; Grierson, 1996; Camp et al., 1998; Soh and Yang, 1998; Grierson and Khajehpour, 2002; Krishnamoorthy et al., 2002; Foley and Schinler, 2003; Rafiq et al., 2003; Sisk et al., 2003; Kicinger et al., 2005). GAs do not require gradient information and continuity assumption. Moreover, they work with a coded parameter set, not with the parameter themselves, and they search simultaneously from multiple points, not a single point. As such they are broadly used for solving nonlinear multidimensional problems that are usually met in the civil engineering problems.

The basic idea of GAs is to start with a set of solutions, randomly generated using the allowable values for each design variable. A set of solution is generally called *a population*. A solution is genetically called *an individual* or *a chromosome string* or shortly called *a string*. Actually a substring is a chromosome, but in some cases there is only one substring in an individual, thus a substring or a string is the same thing, and then an individual is shortly called as a chromosome or a string. A chromosome actually can be created from a lot of genes depending on the coding method for a design variable. Moreover, when the design variable is coded as one gene then a gene is usually referred to as a chromosome.

Each individual is also assigned a fitness value computed from an objective function. From the current population of individuals, a pair of individuals is selected randomly with a bias allocated to more fit members of the population. Random processes are used to generate new individuals using the selected subset of individuals. The size of the population is usually kept fixed. Since fitter individuals of the population are used to create new individuals, the successive populations have a higher probability of having individuals with better fitness values. The process continues until the stopping criterion is satisfied. GAs generally consist of a series of three processes: (1) coding and decoding design variables into chromosome strings or individuals, (2) evaluating the fitness of each individual of the population, and (3) applying genetic operators to generate the next *generation* of the population (Arora, 2004).

3.1 Objective and fitness functions

Although GAs are suitable for unconstrained optimization problems, a constrained optimization problem using GAs can be generally expressed as Eq. (3.1).

Maximize

$$F(\mathbf{x}) = F[f(\mathbf{x})], \quad \mathbf{x} = (x_1, x_2, ..., x_N) \in \mathbf{R}^N, \tag{3.1}$$

under constraints defined as

$$g_i(\mathbf{x}) \le 0, \quad i = 1, 2, ..., K,$$
$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, ..., P.$$

In a structural design optimization problem, **x** is an *N*-dimensional vector called the design vector, representing *N* design variables to be optimized. For example, in a truss structure, design variables are the cross-sectional areas of all the members. In floor layout design, design variables are positions of beams.

The function *f* in Eq. (3.1) is the objective function of the optimization problem. For example, for sizing optimization of a truss, *f* is the total weight of the truss. The constraints g_i and h_i, called inequality and equality constraints respectively, are to be satisfied. Example are stress limits, displacement limits, beam redundancy, and column redundancy. The fitness function *F* is defined as a figure of merit. For the maximization problems, the fitness function can be, though not necessary, the same as the objective function. In the case of the minimization problems, the fitness function is an equivalent maximization problem chosen such that the optimum point remains unchanged. The following fitness function is often used:

$$F(\mathbf{x}) = 1/[1 + f(\mathbf{x})]. \tag{3.2}$$

3.2 Coding and decoding

An essential characteristic of GAs is the coding of the design variables. There are many coding schemes available, such as binary codes and real codes (Deb, 1995). The most common coding scheme is to transform the design variables to a binary string of a specific length. For multivariable optimization problems, the coding is constructed by concatenating as many single variable codes as the number of the design variables in the design problem. The length of the coded representation of a design variable corresponds to its range and precision.

3.2.1 Binary coding

The binary-coded string comprising 1's and 0's is broadly used. The length of the string is usually determined according to the desired solution accuracy. For

example, if four bits are used to code each variable in a two-variable function optimization problem, the string (0000 0000) and (1111 1111) would represent the vector of points

$$(x_1^{(L)}, x_2^{(L)})^T \quad \text{and} \quad (x_1^{(U)}, x_2^{(U)})^T ,$$

respectively, because the substrings (0000) and (1111) have the minimum and the maximum decoded values. Here, $x_1^{(L)}$ and $x_2^{(L)}$ are the lower bounds of x_1 and x_2, respectively, while $x_1^{(U)}$ and $x_2^{(U)}$ are the upper bounds of x_1 and x_2, respectively. Any other eight-bit string can be found to represent a point in the search space according to a fixed mapping rule. By employing the binary coding method with an L_i-bit coding for a design variable, the obtainable accuracy in that variable is approximately $(x_i^{(U)} - x_i^{(L)}) / 2^{L_i}$.

3.2.2 Decoding and mapping

The variable x_i is coded in a substring s_i of the length L_i. The decoded value, $s_i^{decoded}$, of a binary substring s_i is calculated as follows:

$$s_i^{decoded} = \sum_{i=0}^{L-1} 2^i s_i , \tag{3.3}$$

where $s_i \in (0,1)$ and the string s_i is represented as $(s_{L-1}s_{L-2}...s_2s_1s_0)$. For example, a four-bit string (0111) has a decoded value of $((1)2^0 + (1)2^1 + (1)2^2 + (0)2^3)$ or 7.

Different mapping rules can be established to fit different optimization problems. For example, the following linear mapping rule is used to decode an encoded design variable into an unsigned real number as shown below (Deb, 1995):

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{L_i} - 1}(s_i^{decoded}), \tag{3.4}$$

where $x_1^{(L)}$ and $x_2^{(L)}$ are the lower bounds of x_1 and x_2, respectively, and $x_1^{(U)}$ and $x_2^{(U)}$ are the upper bounds of x_1 and x_2, respectively. The length of a substring is L_i.

3.3 Genetic operators

The simple genetic algorithm (SGA) basically consists of three operators: (1) reproduction, (2) crossover and (3) mutation (Goldberg, 1989).

3.3.1 Reproduction

Reproduction is a process of selecting a set of design variables from the current population to create the next generation. There are many attempts to propose the selection techniques such as roulette wheel selection or stochastic sampling, the remainder stochastic sampling, and the stochastic tournament or the ranking method (Goldberg, 1989). All of selection techniques are biased toward more fit members of the current population.

Roulette wheel selection is powerful and relatively the easiest one such that it is used by many researchers (Goldberg, 1989; Deb, 1995; Burns, 2002). In the SGA, the reproduction process is the roulette wheel selection. The essential idea of the roulette wheel selection is that the individual with higher fitness value have larger probability of selection. Thus, the *i*th individual in the current population is selected with a probability proportional to its fitness F_i. Since the population size is usually kept fixed in the SGA, the sum of the probability of each individual in the population being selected for the mating pool must be one. Thus, the probability for selecting the *i*th individual, p_i, is as follows:

$$p_i = \frac{F_i}{\sum_{j=1}^{n} F_j}, \tag{3.5}$$

where *n* is the population size and F_i is the fitness value of the *i*th individual. Moreover, for the constrained optimization problem, the augmented fitness value F_i^a is used instead of F_i. The details of the augmented fitness value will be explained in the next section.

Unlike the roulette wheel selection, the tournament selection is to select the winner individual from a tournament competition among N_{ts} individuals (frequently N_{ts} = 2) that are randomly selected from the current population. The winner is the one with the highest fitness of the N_{ts} tournament competitors. The winner is inserted into the mating pool. The tournament selection process is repeated until the mating pool is completed. Thereafter, two individuals are orderly mated and applied the crossover and mutation operators.

3.3.2 Crossover

Once a new population is determined, the crossover operator is conducted as a means to introduce variation into a population by changing information among individuals of the mating pool. Many crossover operators exist in the literature such as one-point crossover, two-point crossover, and uniform crossover (Goldberg, 1989; Deb, 1995; Burns, 2002). One-point crossover is relatively the easiest one to apply. In the crossover process, two individuals called parent individuals are picked from the mating pool at random and then some portions of the two individuals are exchanged resulting in two new individuals called offspring.

One-point crossover is performed by randomly choosing a crossing point along the individual and by exchanging all bits on the right side of the crossing point, for example, as shown below. It can be observed that the bits next to the crossing point of two parent individuals are exchanged to two offspring individuals.

Two parent individuals
x_1 = 1 0 1 1 1 0|1 0 0 1
x_2 = 0 1 0 1 0 0|1 0 1 1

Two offspring individuals
x_1' = 1 0 1 1 1 0|1 0 1 1
x_2' = 0 1 0 1 0 0|1 0 0 1

In two-point crossover, two crossing points are randomly chosen and all bits between these points of both considered parent individuals are exchanged. It should be noted that using many crossover points reduces the performance of the GA. The problem with many crossing points is that the building blocks are more likely to be disrupted. However, an advantage of having more crossing points is that the problem space may be searched more thoroughly. An example of two-point crossover is as shown below.

Two parent individuals
x_1 = 1 0 1 |1 1 0|1 0 0 1
x_2 = 0 1 0 |1 0 0|1 0 1 1

Two offspring individuals
x_1' = 1 0 1 |1 0 0|1 0 0 1
x_2' = 0 1 0 |1 1 0|1 0 1 1

In uniform crossover, each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a binary crossover mask. The binary crossover mask is randomly created with the same length as the parent chromosome string. For each bit position in the mask, its value 1 or 0, respectively, indicates that the first parent or the second parent contributes its values in that position to the first offspring, and vice versa for the second offspring. An example of uniform crossover is illustrated as follows.

Two parent individuals

x_1 = 1 0 1 1 1 0 1 0 0 1

x_2 = 0 1 0 1 0 0 1 0 1 1

Mask

1 1 0 0 0 1 1 0 0 0

Two offspring individuals

x_1' = 1 0 0 1 0 0 1 0 1 1

x_2' = 0 1 1 1 1 0 1 0 0 1

The crossover probability (P_C) is a parameter to describe how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If P_C is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of individuals from the old population. However, the new generation may not be the same as the old population. This is because there will be mutation process after crossover process. The mutation will be explained in the next section. Crossover is made in hope that new individuals or chromosome strings will contain good parts of old individuals from the old population such that the new individual will be better. However, it is good to leave some individuals of the old population to survive to the next generation. Generally, the parametric study is needed to estimate the value of P_C required for finding good solutions.

3.3.3 Mutation

Mutation is the important operator mimicking the behavior of natural mutation to ensure the diversity of the population. Mutation is used to prevent the algorithm from being trapped in a local minimum. There are various forms of mutation for the different kinds of representation such as flipping, interchanging, and inversion. In case of flipping, mutation changes the value of each gene from 1 to 0 and vice versa with the mutation probability (P_M). In case of interchanging, two random positions of the string are chosen and the bits corresponding to those positions are interchanged. For inversion, two random positions are chosen and bits between those selected positions are reversed.

The mutation probability decides how often parts of the string will be mutated. Typically, P_M is quite low at about 1% or less. When P_M is 100%, whole chromosomes are altered while 0% P_M implies no change. If there is no mutation, offspring are generated immediately after crossover or directly copied without any change.

3.4 Stopping criteria

There are many stopping criteria employed in the literature (Ghasemi and Hinton, 1996; Ghasemi et al., 1999). For example, the total number of iterations can be used as a stopping criterion. The required number of successive iterations for the fittest solution that has not changed can also be used. In addition, the calculation may be set to stop when the difference of the fittest solution of the current iteration and that of the last 20 iterations is smaller than a specified value. Generally, the number of iterations or *generations* is widely used as the stopping criterion.

3.5 Elitism

Generally, some very good individuals that appear in the early GA generations may disappear from the later generations. This is because GAs employ probabilistic processes in their calculations. As a result, to ensure that the best-fit individuals in the current generation will survive in the next generation, it is possible to place them directly in the next generation. The process is called elitism and GAs that employ elitism are called elitist GAs. The main concept of all elitist GAs is that the best solution or solutions are placed directly in the population of the subsequent generation regardless of the reproduction, crossover and mutation operators. Since most GAs use constant population sizes, the best solutions cannot be added to the next generation but are frequently used to replace some worst solutions. It is also possible to randomly pick up individuals that are to be replaced by the best solutions.

3.6 Penalty function**s**

GAs are intrinsically suitable only for unconstrained optimization problems. However, there are many methods that enable GAs to handle constraints. Among these methods, penalty function methods have been mostly used (Nanakorn and Meesomklin, 2001; Krishnamoorthy et al., 2002; Nimityongskul, 2004). Penalty function methods penalize infeasible solutions, i.e.

$$F^a(\mathbf{x}) = F(\mathbf{x}) - P(\mathbf{x}) \tag{3.6}$$

where P is a penalty function whose value is greater than zero. In addition, F^a represents an augmented fitness function after the penalty. The penalty function can be generalized as follows (Nanakorn and Meesomklin, 2001):

$$P(\mathbf{x}) = \sum_{j=1}^{K} (\lambda_G)_j [G_j(\mathbf{x})]^{\beta} + \sum_{j=1}^{P} (\lambda_H)_j [H_j(\mathbf{x})]^{\beta} \tag{3.7}$$

where

$$G_j(\mathbf{x}) = \max[0, g_j(\mathbf{x})],$$
$$H_j(\mathbf{x}) = \text{abs}[h_j(\mathbf{x})].$$

where g_j and h_j are the constraint functions in Eq. (3.1). The vector G_j represents the degree of the inequality constraint violations. The vector H_j represents the degree of the equality constraint violations. In addition, $(\lambda_G)_j$, $(\lambda_H)_j$ and β are constants. Generally, the values of $(\lambda_G)_j$ and $(\lambda_H)_j$ are usually the same and set as a constant. The value of β is generally set as 1 or 2.

3.7 Fitness scaling

In reproduction process, the augmented fitness function F^a will be used in Eq. (3.5) instead of the original fitness F. Therefore, it is essential that all F^a must be positive. Consequently, the obtained F^a may not be directly usable as its value may be negative. Moreover, the difference between the highest F^a and the average F^a varies over generations. In early generations, the difference can be very large because it is common to have few extraordinary individuals with very high fitness in a population. As a result, the extraordinary individuals may take over a significant proportion of the population, and this can be undesirable and may result in premature convergence. In later generations, there may be insignificant diversity within the population. Consequently, the average F^a may be close to the highest F^a. If this situation is unchanged, individuals with average F^a and individuals with the highest F^a will have nearly the same numbers of copies in future generations. In this case, the survival of the fittest strategy necessary for improvement becomes a random walk. To prevent all of these problems, the fitness function is usually scaled into a specified positive range. Many fitness scaling have been proposed in the literature (Goldberg, 1989). Here only some scaling techniques are presented including linear scaling, σ-truncation, and bilinear scaling.

The linear scaling requires a linear relationship between the scaled fitness F^S and the raw augmented fitness F^a as shown in Eq. (3.8).

$$F^S(\mathbf{x}) = aF^a(\mathbf{x}) + b \tag{3.8}$$

The coefficient a and b in Eq. (3.8) can be chosen in a number of ways. For example, from Fig. 3.1 the average raw augmented fitness F^a_{avg} is scaled to 1. The maximum scaled fitness that is to be obtained from the best members is set to C. Thus, the chance of the best members being selected into the mating pool is equal to C times that of the average members. In other words, the number of copies of the best members in the mating pool is expected to be C times that of the average members. Linear scaling under normal condition can be illustrated by using Fig. 3.1. The coefficient of a and b in Eq. (3.8) can be computed using Eq. (3.9).

$$\begin{aligned} a &= \frac{C-1}{F^a_{\max} - F^a_{avg}}, \\ b &= \frac{F^a_{\max} - CF^a_{avg}}{F^a_{\max} - F^a_{avg}} \end{aligned} \tag{3.9}$$

In general, $C = 1.2$ to 2 has been used successfully (Goldberg, 1989). Normally there is no problem applying this linear scaling concept. Nevertheless the situation as shown in Fig. 3.2 may appear resulting in a negative scaled fitness value of the $F^a_{\min}$. This type of situation is common in a mature run when a few lethally bad individuals are far below F^a_{avg} and $F^a_{\max}$, which are relatively close together. To prevent obtaining negative scaled fitness, Goldberg (1989) suggests that the non-negative test be applied by checking whether $F^a_{\min} > (CF^a_{avg} - F^a_{\max})/(C - 1.0)$. If the condition is true, Eq. (3.9) can be used to scale all F^a. However if it is false, F^a will be scaled as much as possible using Eq. (3.10). This means that $F^a_{\min}$ is scaled to zero as shown in Fig. 3.3,

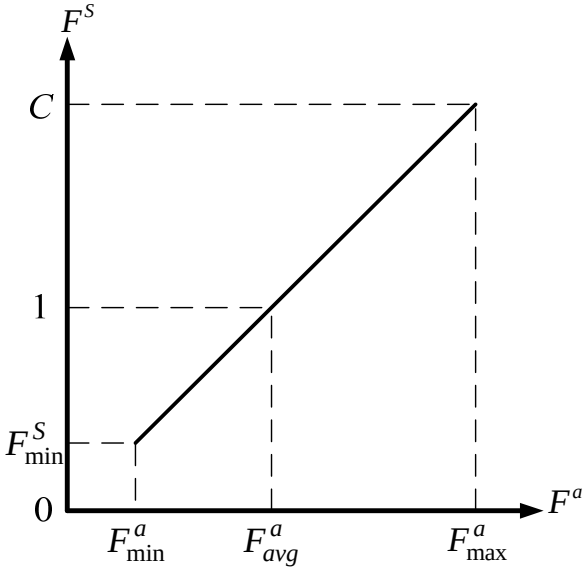


Fig. 3.1. Linear scaling.

instead of setting $F_{\max}^{S} = C$.

$$a = \frac{1}{F_{\max}^{a} - F_{avg}^{a}}\ , \qquad b = \frac{F_{\min}^{a}}{F_{avg}^{a} - F_{\max}^{a}} \tag{3.10}$$

The bilinear scaling is shown in Fig. 3.4. The minimum scaled fitness is set to zero to avoid negative fitness values while the scaled fitness of the average fitness of all individuals F_{avg}^{a} is set to one. Furthermore, the maximum scaled fitness that is to be obtained from the best members is set to C. Thus, the chance of the best members being selected into the mating pool is equal to C times that of the average members. Here, F^{S} denotes the scaled fitness. In addition $F_{\min}^{a}$ denotes the minimum fitness value after the penalty while $F_{\max}^{a}$ denotes the fitness value of the best members. This scaled fitness function F^{S} can be computed by using Eq. (3.11), i.e.

$$\begin{aligned} F^{S}(\mathbf{x}) &= \frac{C-1}{F_{\max}^{a} - F_{avg}^{a}} F^{a}(\mathbf{x}) + \frac{F_{\max}^{a} - CF_{avg}^{a}}{F_{\max}^{a} - F_{avg}^{a}} && \text{if } F^{a}(\mathbf{x}) \geq F_{avg}^{a} \\ F^{S}(\mathbf{x}) &= \frac{1}{F_{avg}^{a} - F_{\min}^{a}} F^{a}(\mathbf{x}) + \frac{F_{\min}^{a}}{F_{\min}^{a} - F_{avg}^{a}} && \text{if } F^{a}(\mathbf{x}) \leq F_{avg}^{a} \end{aligned} \tag{3.11}$$

There is one alternative method to circumvent obtaining negative scaled fitness in the linear scaling, which is called σ-truncation. This method is applied before using the linear scaling method in order to prevent negative scaled fitness. In this procedure, a constant is subtracted from raw augmented fitness value F^{a} to obtain $\bar{F}^{a}$ as

$$\bar{F}^{a} = F^{a} - (F_{avg}^{a} - c\sigma) \tag{3.12}$$

where the constant c is chosen as a reasonable multiple of the population standard deviation σ (between 1 and 3) and negative results ($\bar{F}^{a} < 0$) are set to zero. Following σ-truncation, the linear scaling can proceed as described earlier without that danger of negative scaled fitness.

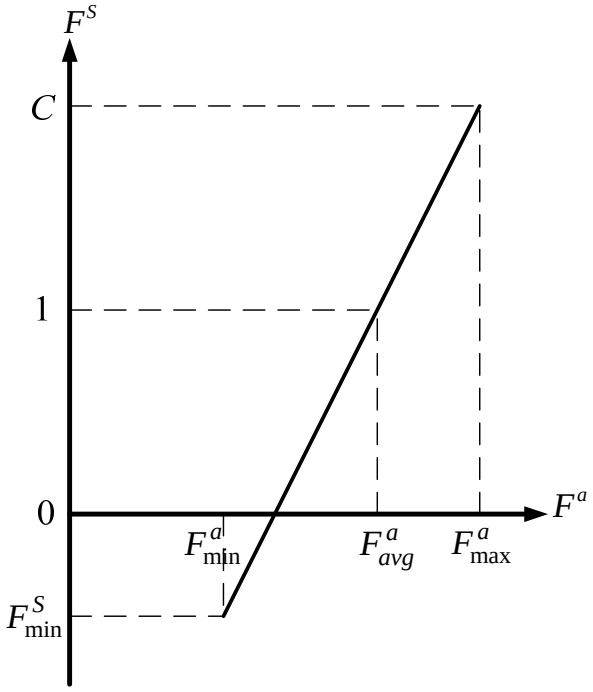


Fig. 3.2. Difficulty of the linear scaling.

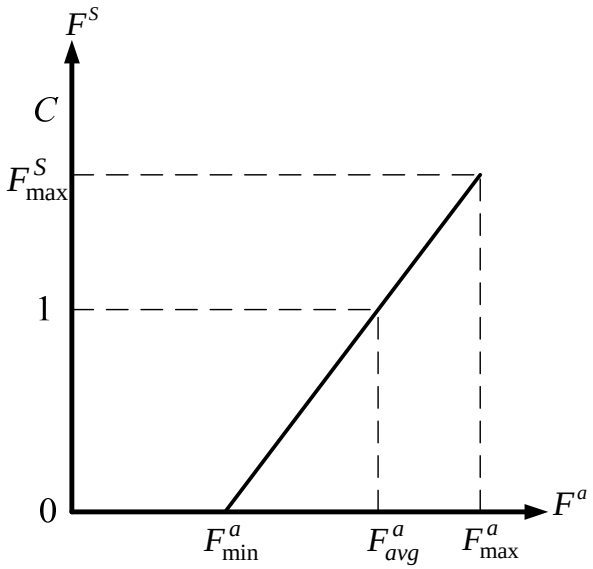


Fig. 3.3. Modified linear scaling.

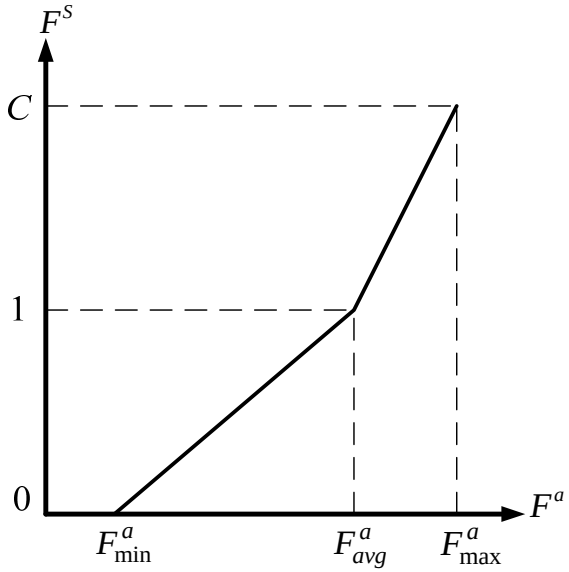


Fig. 3.4. Bilinear scaling.

3.8 Adaptive penalty function

The penalty scheme used in GAs plays an important role in the performance of GAs. Generally, the penalty is more important when the optimal solution lies on or close to the boundary between feasible and infeasible search spaces, which is often found in the structural design optimization problems. From Eq. (3.7), the general penalty function has many coefficients that need to be set before computing the value of the penalty. It is well known that, for various stages of the calculation, different degrees of penalty are needed. Nevertheless, it is not easy to set appropriate values of these coefficients for different generations. Therefore, all coefficients are usually constant for all generations. There are some penalty schemes whose penalty coefficients can be varied manually in order to adjust the strength of the penalty during the calculation (Adeli and Cheng, 1994; Rajan, 1995). Recently, a new penalty scheme had been proposed by Nanakorn and Meesomklin (2001). The basic idea of their adaptive penalty scheme is to penalize infeasible solutions so that the individual chance of the best infeasible members being selected into the mating pool with respect to that of the average feasible members is always the same in all generations. Here, only the main concept of this adaptive penalty scheme is discussed. Complete details of the scheme, including its implementation, can be found in the work by Nanakorn and Meesomklin (2001).

In the work by Nanakorn and Meesomklin (2001), a modified bilinear scaling scheme shown in Fig. 3.5 is employed for the fitness scaling. In the figure, F^S is the scaled fitness and $F^a_{\min}$ represents the minimum augmented fitness in the generation. In addition, $F^{a,\tilde{\mathbf{F}}}_{avg}$ and $F^{a,\tilde{\mathbf{F}}}_{\max}$ denote, respectively, the average and maximum augmented fitness values of all feasible individuals in the generation. Note that, for feasible

individuals, the augmented fitness F^a and the original fitness F are the same. It can be seen from the figure that the scaled fitness of the average fitness of all feasible individuals is set to one while the maximum scaled fitness that is to be obtained from the best feasible members is set to *C*. Hence, the individual chance of the best members being selected into the mating pool is equal to *C* times that of the average feasible members. For the case where there is only one feasible individual in the generation, the scaled fitness of this individual will be set to one.

Next, the individual chance of the best infeasible members being selected into the mating pool is set to be equal to φ times that of the average feasible members, i.e.

$$F^S(\mathbf{x}_d) \le \left(\varphi F_{\text{avg}}^{S,\tilde{\mathbf{F}}} = \varphi\right) \qquad \text{for } \forall \mathbf{x}_d \in \tilde{\mathbf{U}}, \tag{3.13}$$

where $\tilde{\mathbf{U}}$ denotes the infeasible search space with respect to the constraints. In addition, $F_{avg}^{S,\tilde{\mathbf{F}}}$ is the scaled value of the average fitness of all feasible individuals, which from Fig. 3.5 is equal to one. Eq. (3.13) allows the factor λ in Eq. (3.7) to be calculated in each generation. In summary, the employed adaptive penalty scheme requires two constant input parameters to be prescribed prior to the calculation. The parameters are *C* and φ In the penalty scheme by Nanakorn and Meesomklin (2001), Eq. (3.11) becomes

$$\begin{aligned} F^S(\mathbf{x}) &= \frac{C-1}{F_{\max}^{a,\tilde{\mathbf{F}}} - F_{avg}^{a,\tilde{\mathbf{F}}}} F^a(\mathbf{x}) + \frac{F_{\max}^{a,\tilde{\mathbf{F}}} - CF_{avg}^{a,\tilde{\mathbf{F}}}}{F_{\max}^{a,\tilde{\mathbf{F}}} - F_{avg}^{a,\tilde{\mathbf{F}}}} && \text{if } F^a(\mathbf{x}) \ge F_{avg}^{a,\tilde{\mathbf{F}}} \\ F^S(\mathbf{x}) &= \frac{1}{F_{avg}^{a,\tilde{\mathbf{F}}} - F_{\min}^{a}} F^a(\mathbf{x}) + \frac{F_{\min}^{a}}{F_{\min}^{a} - F_{avg}^{a,\tilde{\mathbf{F}}}} && \text{if } F^a(\mathbf{x}) \le F_{avg}^{a,\tilde{\mathbf{F}}} \end{aligned} \tag{3.14}$$

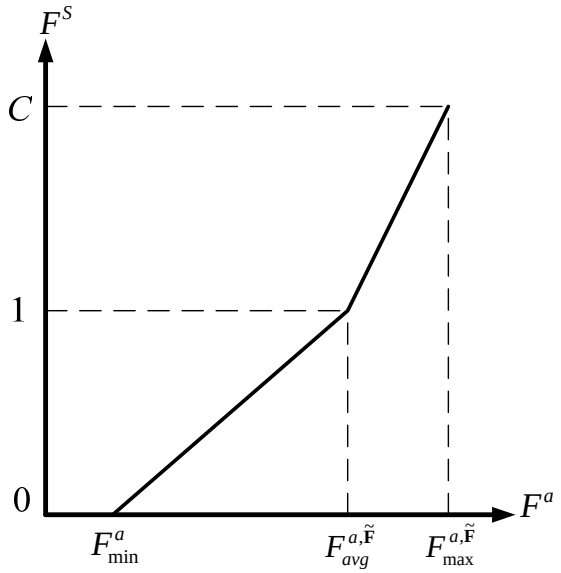


Fig. 3.5. Bilinear scaling for the adaptive penalty scheme.

Chapter 4

Layout Design of Beam-Slab Floors using a GA

A new GA for beam-slab layout design is proposed in this study. Details of the proposed algorithm are presented in this chapter. The algorithm implementation will be presented in the next chapter. The primary input of the algorithm is an architectural floor plan with given positions of columns and walls. First, the representation of beam-slab layouts is presented. After that, the fitness function and design constraints are established. Finally, the proposed GA is developed. The proposed GA uses the simple GA (Goldberg, 1989) as its core algorithm. The simple GA is modified by adding the elitism and the adaptive penalty function.

4.1 Primary representation of beam-slab layouts

In order to use a GA for beam-slab layout design, it is essential to establish how beam-slab layouts are coded in the algorithm. In this study, binary strings will be used to represent beam-slab layouts. To begin coding, a grid is superimposed on a given architectural floor plan in such a way that there are grid lines passing through all columns and wall lines. Each line segment of this grid represents a possible position of a beam segment. Therefore, the spacing of the grid can be set based on the required degree of precision for beam positions. A one-bit chromosome is attached to each line segment. If the value of the one-bit chromosome is one, it means that there is a beam segment on that line segment. Naturally, if the value is zero, there is no beam segment. A slab is simply defined as a rectangular area that is completely surrounded by beams. For example, Fig. 4.1a is a rectilinear floor plan. By considering the columns and wall lines, Fig. 4.1a grid in Fig. 4.1b can be obtained. Examples of a beam segment and a slab are also shown in Fig. 4.1b. In addition, Fig. 4.1c shows an example of a beam-slab layout for the floor and Fig. 4.1d depicts its corresponding code. Note that the representations of columns, walls, beams, and slabs used in Fig. 4.1c will be employed throughout this study.

4.2 Geometrically invalid layouts

All possible beam segments in the grid that are not prescribed in advance as part of the problem setup are used as the design variables of the problem. Placing beam segments arbitrarily may result in layouts that are not geometrically valid. Geometrically invalid layouts are those layouts that consist of at least one invalid beam segment. In this study, as shown in Fig. 4.2, invalid beam segments include

1) any isolated beam segment,
2) any beam segment with one free end,

3) any two beam segments that form an L-shaped interior beam, and

4) any two beam segments that form a concave L-shaped beam on the outer boundary of the floor.

As shown in Fig. 4.2, an isolated beam segment is a beam segment that is not connected to any other beam segments. This isolated beam is considered as an invalid beam because it is obviously unrealistic. The rest are considered as invalid beams since all of them result in non-rectangular slabs. Non-rectangular slabs cannot efficiently transfer floor loads to the supporting beams. Note that, in the consideration of two beam segments that form a concave L-shaped exterior beam, only beam segments that are on the outer boundary of the floor are included.

4.3 Proposed coding scheme

During the GA iteration, it is likely that some individuals that appear in the population may represent geometrically invalid layouts. A penalty scheme may be used

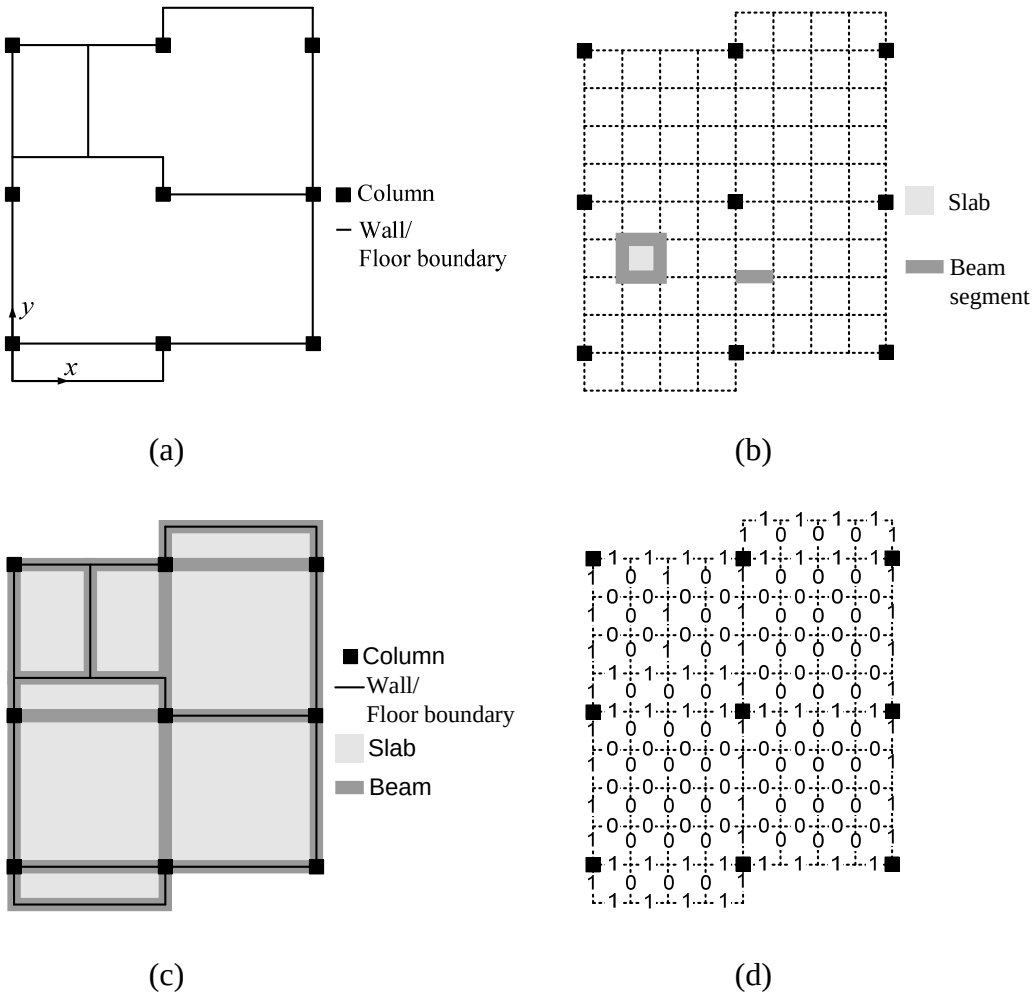


Fig. 4.1. A rectilinear floor. (a) An example of a floor plan. (b) A grid. (c) A beam-slab layout. (d) The corresponding code.

to take care of these geometrically invalid layouts. However, it is difficult to devise this penalty scheme because it is not clear how to evaluate the degrees of the disadvantages of these layouts. If the penalties for these geometrically invalid layouts are not known, it will not be possible to obtain the penalized fitness of the layouts. This study proposes a new concept to handle this problem. The concept is not to treat geometrically invalid layouts as bad layouts that have to be penalized. Instead, each geometrically invalid layout will be mapped to a geometrically valid layout whose fitness will be used as the fitness of the original geometrically invalid layout. In the real implementation, each individual will have two corresponding chromosome strings instead of one. The first chromosome string, called the original string, represents the original shape of the layout that may or may not be geometrically valid. The second chromosome string, called the derived string, represents the shape of the geometrically valid layout derived from the first string by mapping. If the original string already represents a geometrically valid layout, the derived string is the same as the original string. In the mapping process, the derived geometrically valid layout is obtained from the original geometrically invalid layout by removing invalid beam segments from the original layout. The proposed mapping algorithm is shown as follows:

Algorithm Chromosome_Mapping
Input: An original geometrically invalid chromosome string
Output: The derived geometrically valid chromosome string

- Copy the original chromosome string to the derived chromosome string
- **while** the derived chromosome string represents a geometrically invalid layout **do**
    - Remove all isolated beam segments by changing their chromosomes from one to zero in the derived chromosome string
    - Remove all beam segments with one free end by changing their chromosomes from one to zero in the derived chromosome string
    - Remove all pairs of beam segments that form an L-shaped interior beam by changing their chromosomes from one to zero in the derived

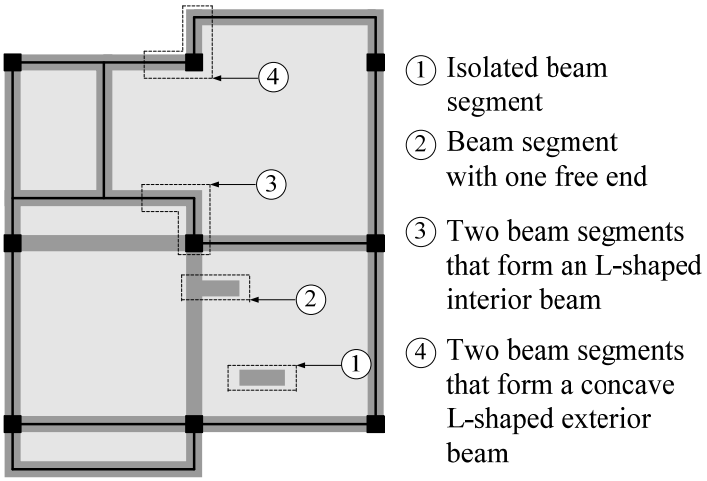


Fig. 4.2. Examples of all defined invalid beams.

chromosome string

- Remove all pairs of beam segments that form a concave L-shaped exterior beam on the outer boundary of the floor by changing their chromosomes from one to zero in the derived chromosome string.

Since the fitness of each individual is obtained from its derived geometrically valid layout, it can be said that the original string is in fact decoded into the geometrically valid layout represented by the derived string. As the search space of the problem is the space that contains all possible strings, the mapping process allows all individuals in the search space to be interpreted as geometrically valid layouts. As a result, there are seemingly no geometrically invalid layouts at all in the search space.

An example of how to derive a geometrically valid beam-slab layout of a rectangular floor is shown in Fig. 4.3. The first subfigure Fig. 4.3a shows the geometrically invalid beam-slab layout of a rectangular floor and the derived geometrically valid beam-slab layout obtaining from the proposed mapping algorithm. The detailed steps of the mapping process are shown in Fig. 4.3b. In this example it should be noted that there is no concave L-shaped beam on the outer boundary of the floor to be removed in the mapping algorithm. In case of a rectilinear floor, it will always have a concave L-shaped beam on the outer boundary to be considered. For example, Fig. 4.4 shows how to derive a geometrically valid beam-slab layout of a rectilinear floor. The first subfigure Fig. 4.4a shows the geometrically invalid beam-slab layout of a rectilinear floor and the derived geometrically valid beam-slab layout obtaining from the proposed mapping algorithm. The detailed steps of the mapping process are shown in Fig. 4.4b. Note that the derived geometrically valid beam-slab layout of a rectilinear floor may not occupy the whole floor area.

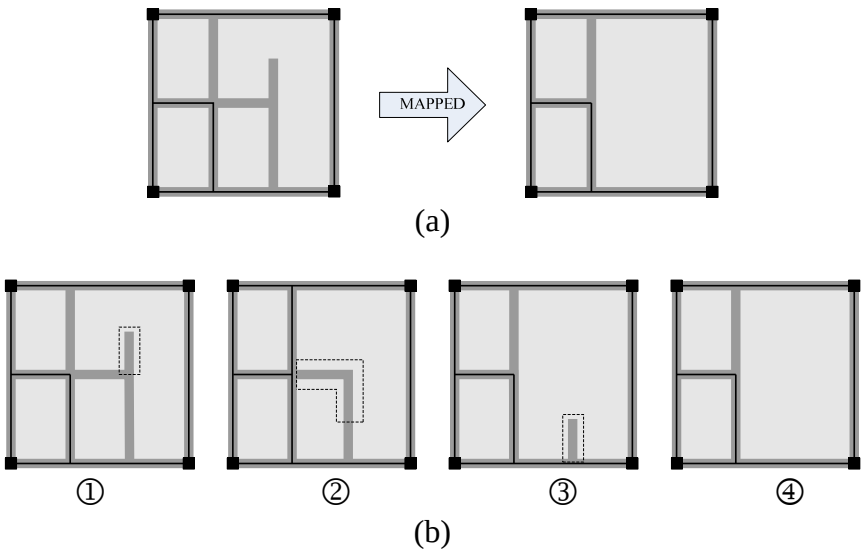


Fig. 4.3. (a) An example of geometrically invalid beam-slab layout of a rectangular floor and the derived geometrically valid beam-slab layout. (b) Steps of deriving the derived geometrically valid beam-slab layout of a rectangular floor.

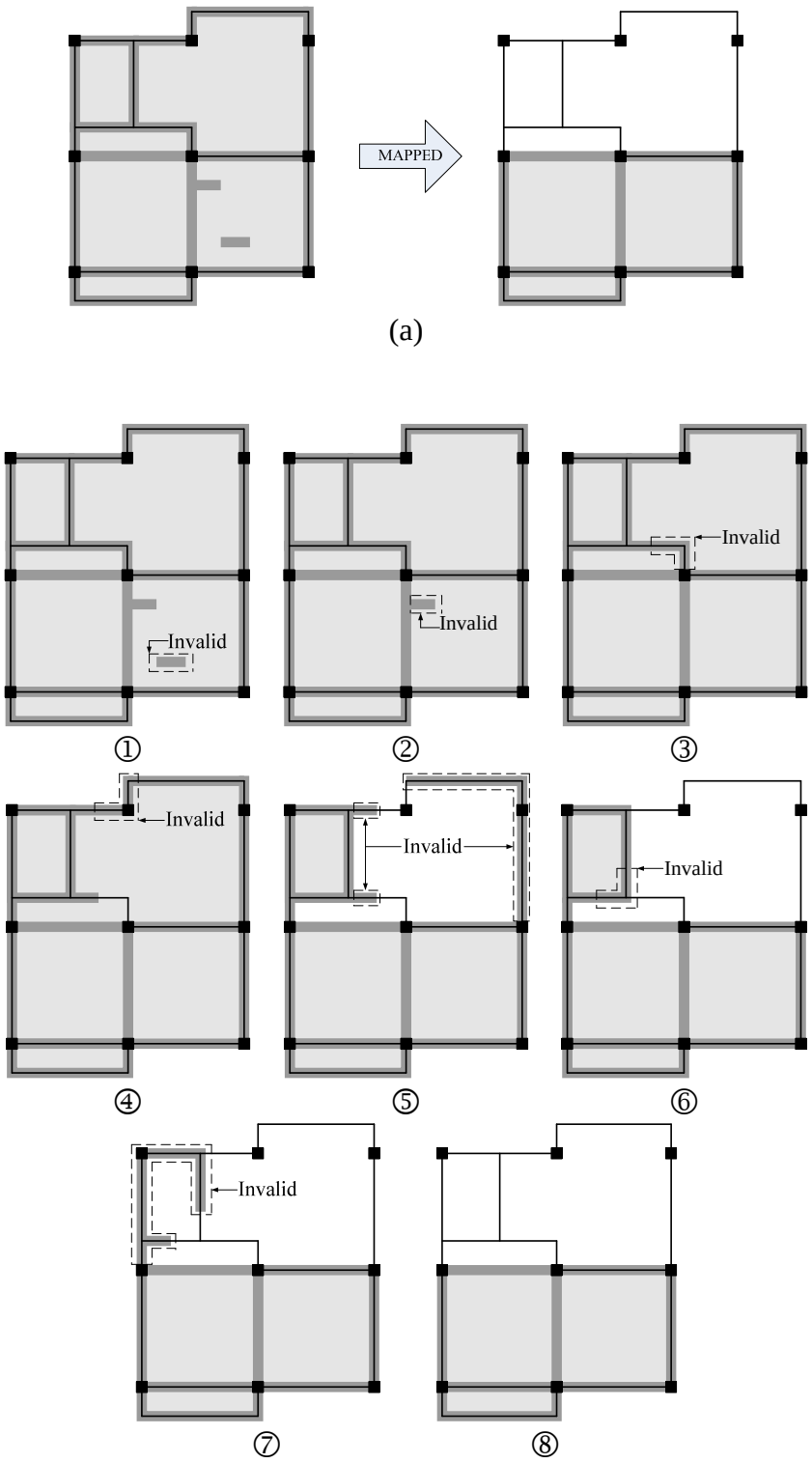
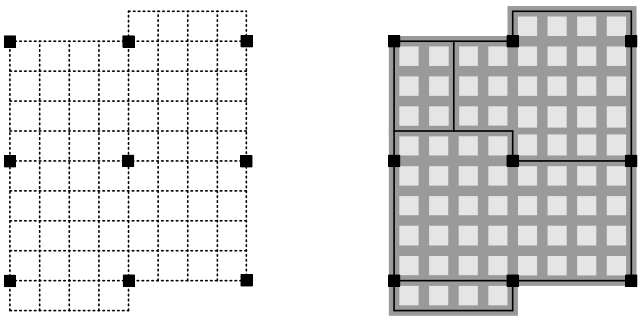


Fig. 4.4. (a) An example of geometrically invalid beam-slab layout of a rectilinear floor and the derived geometrically valid beam-slab layout. (b) Steps of deriving the derived geometrically valid beam-slab layout of a rectilinear floor.

Although the proposed mapping scheme enables each geometrically invalid individual to be interpreted as a geometrically valid layout and, subsequently, to be evaluated, the process does have its disadvantage. Many different geometrically invalid layouts can generally be mapped to the same corresponding geometrically valid layout. For example, a rectangular layout with beams only along its entire outer boundary can be the derived layout of many different geometrically invalid layouts that have the same outer-boundary beams and some additional interior beams that are not connected to the outer-boundary beams. This characteristic of the mapping scheme creates a bias in the reproduction process. The reason is that different geometrically valid layouts will not have the same number of representatives in the search space. If more geometrically invalid layouts are mapped to a certain geometrically valid layout, that particular geometrically valid layout will subsequently have more representatives in the search space. The proposed mapping scheme will in general yield more representatives in the search space for those geometrically valid layouts that have larger slabs and fewer beams. This is because, in the mapping process, beams are always removed from geometrically invalid layouts to obtain geometrically valid ones. Although layouts with larger slabs and fewer beams are generally more preferred, it is necessary that layouts with smaller slabs and more beams be adequately explored. In this study, the bias is alleviated by prescribing a special individual in the initial population. This special individual represents the layout that contains all possible beam segments. For example, Fig. 4.5b shows the special individual to be prescribed in the initial population for the grid in Fig. 4.5a. In addition, Fig. 4.6 shows examples illustrating the concept of the proposed coding of beam-slab layouts. Finally, Fig. 4.7 illustrates the crossover and mutation processes of two individuals A and B, denoted by their original strings. These two original strings are interpreted as geometrically valid layouts via their derived strings. The two original strings are used as the parent strings in the crossover process to obtain two offspring strings. The two offspring strings then mutate and, finally, two new individuals C and D are obtained. These individuals C and D are also interpreted as geometrically valid layouts via their derived strings.



(a) (b)

Fig. 4.5. (a) The grid. (b) The special individual that contains all possible beam segments.

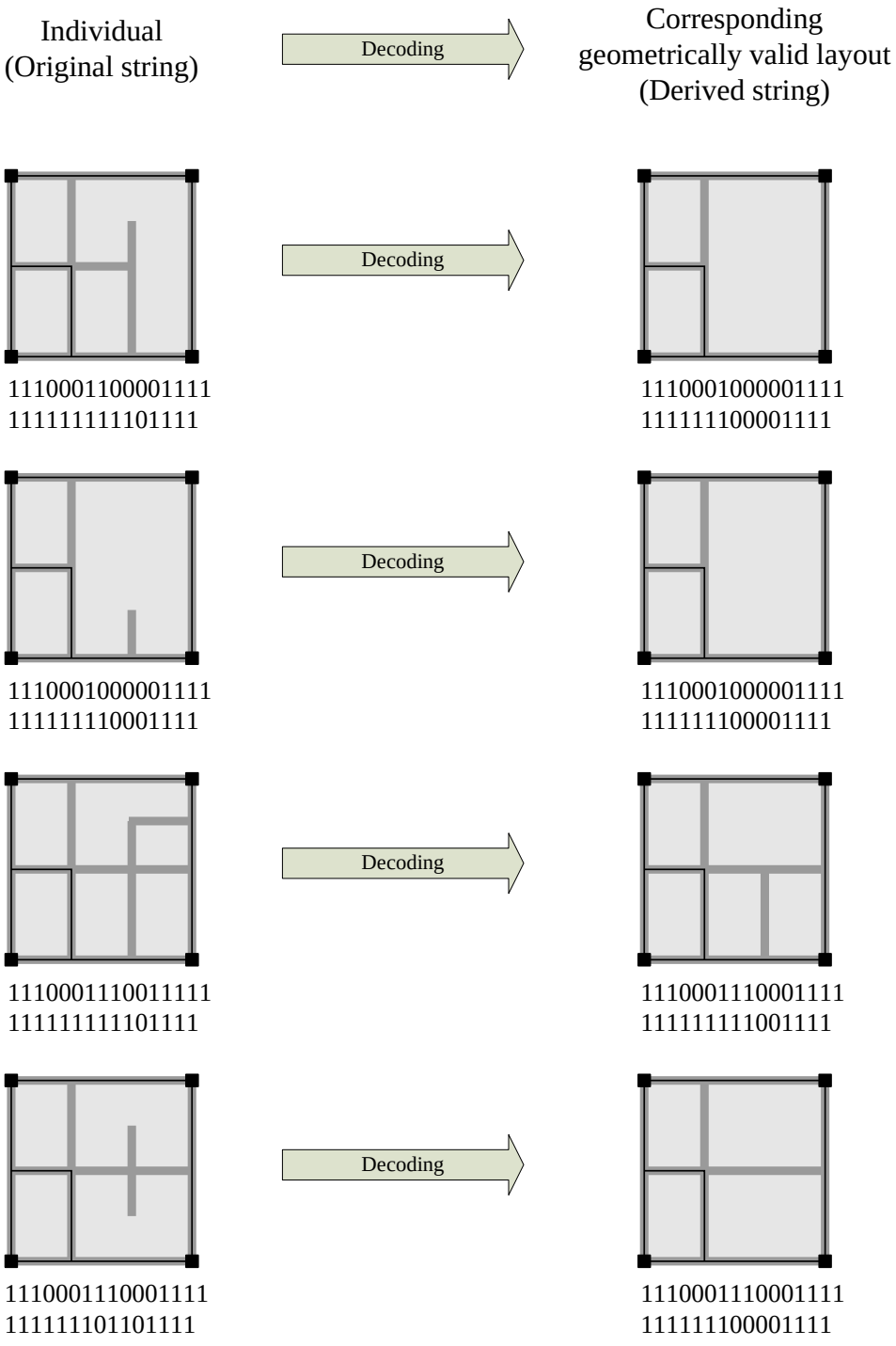


Fig. 4.6. Example illustrating the concept of the proposed coding scheme.

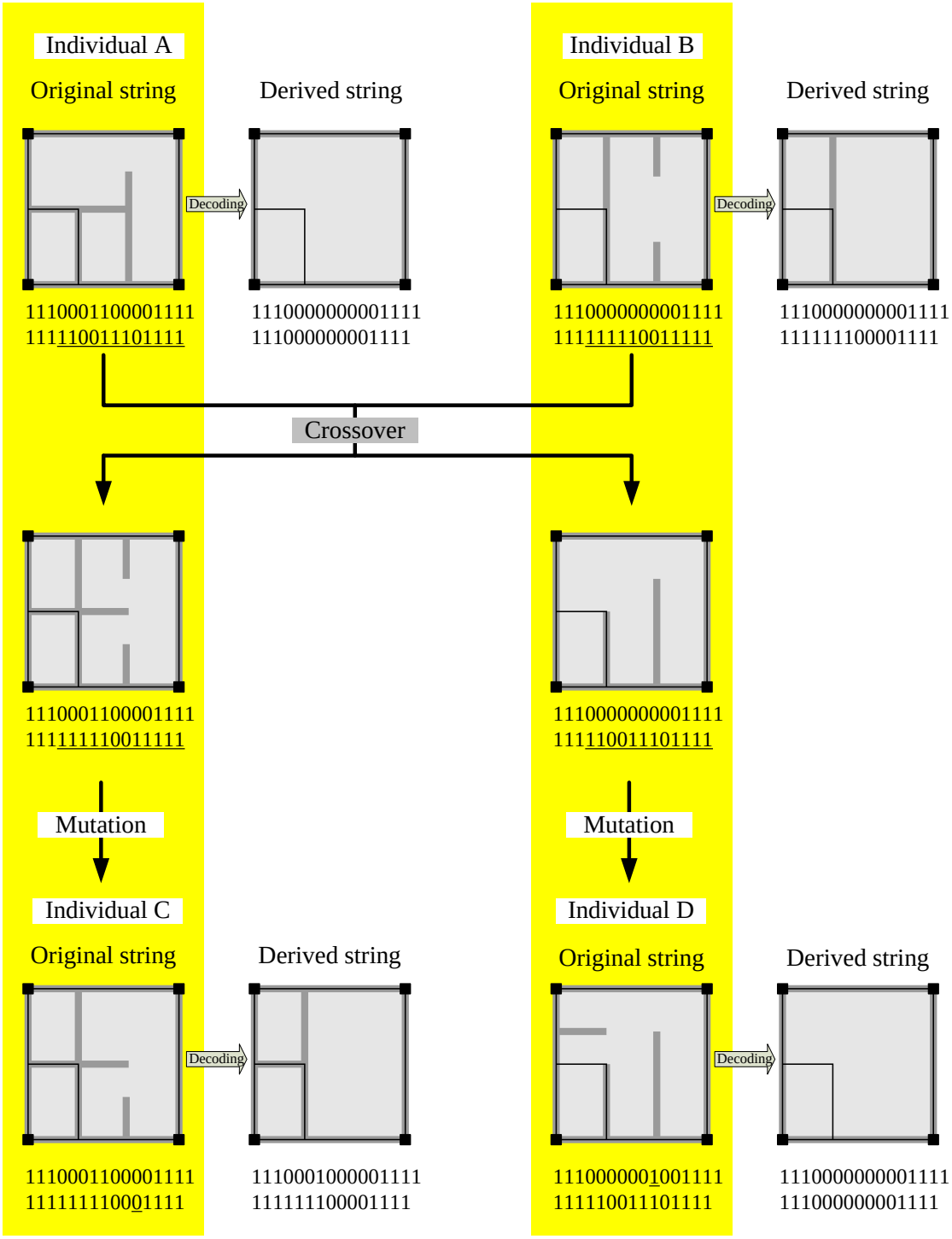


Fig. 4.7. Crossover and mutation processes.

4.4 Problem formation

After establishing the coding scheme, the next step is to write the representative optimization problem for beam-slab layout design. This is explicitly done by defining the problem's objective function as well as constraints. It is desirable that the representative optimization problem is simple but still able to yield reasonably good beam-slab layouts. In the literature, there are several kinds of objective functions such as the project profit (Rafiq et al., 2003), the flexibility of space (Sisk et al., 2003), and the weight of the structure (Nanakorn and Meesomklin, 2001). The objective and constraint functions used in this study are described in the next sections.

4.4.1 Objective function

As mentioned earlier, in this study, the primary input of the proposed algorithm is an architectural floor plan with given positions of columns and walls. If the beam-slab layout of the input architectural floor plan is to be prepared by a designer, it is expected that the designer will try to utilize the given columns to support beams and, subsequently, slabs as efficiently as possible. The efficient column utilization can be defined differently from one designer to another. This study mimics this kind of consideration by defining the objective function F of the problem as

$$F(\mathbf{x}_d) = \frac{1}{N_S(\mathbf{x}_d)} \sum_{i=1}^{N_S} S_i(\mathbf{x}_d). \tag{4.1}$$

Note that the proposed objective function is a function of the derived chromosome string denoted as $\mathbf{x}_d$. To obtain the objective function written in Eq. (4.1), a score is given to each slab based on the number of corner columns it has. In the expression of the objective function, S_i is the score assigned to slab *i*, and N_S is the total number of slabs in the layout. The slab score is given as 1, 0.75, 0.5, 0.25, or 0 if the slab has 4, 3, 2, 1, or 0 corner columns, respectively. Fig. 4.8a shows examples of all five types of slab with different corresponding slab scores. The idea behind the proposed objective function and the slab scoring is based on two assumptions pertaining to the efficient column utilization. First, for a slab, the column utilization is considered better if the slab has more corner columns. This is because corner columns allow efficient load transfer from the slab, via beams, to the columns. Second, the column utilization of the whole floor is considered better if there are fewer slabs in the floor for the given columns. The first assumption is considered in the objective function by the use of the slab score S_i and the second assumption by the use of the total number of slabs N_S. With the form of the objective function in Eq. (4.1), the representative optimization problem for beam-slab layout design becomes the maximization problem of the proposed objective function. Since a GA is to be used to solve this optimization problem, the proposed objective function can be directly employed as the fitness function of the proposed GA. According to this fitness function, a beam-slab layout that has fewer slabs and more

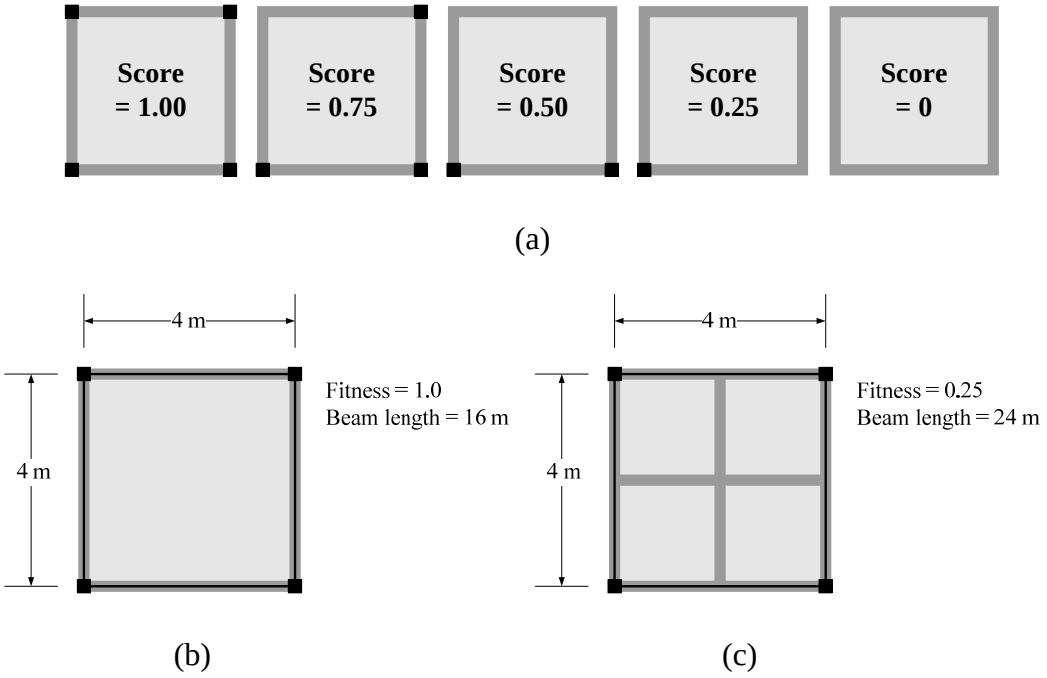


Fig. 4.8. Examples of slabs with different scores.

corner columns of slabs will have higher fitness. Since the employed fitness function encourages layouts that have fewer slabs, it will also encourage layouts that have fewer beams. As an example, Fig. 4.8b and c show two different layouts for the same floor. It can be seen that the total beam length of the layout with higher fitness is shorter than the total beam length of the layout with lower fitness. Using the fitness function that encourages layouts with fewer beams is desirable because, for the same floor plan, layouts with longer beam lengths generally have higher construction costs.

4.4.2 Design constraints

Two types of design constraint are employed in this study, i.e.
1) wall constraint, and
2) slab constraint.

The wall constraint states that all walls must be directly supported by beams. With an *x*-*y* coordinate system that is in alignment with the floor plan, the slab constraint states that the length of a slab in the *x* direction must not exceed a prescribed maximum length for the *x* direction, and the length in the *y* direction must not exceed a prescribed maximum length for the *y* direction. It should be noted that the spacing of the grid used in the calculation must not be set too large that the slab constraint cannot be satisfied. In addition, the slab constraint also states that the whole floor must be completely covered by slabs.

Define the penalty function *P* from these two constraints as

$$P(\mathbf{x}_d) = \lambda E(\mathbf{x}_d) = \lambda[H_{wall}(\mathbf{x}_d) + H_{slab}(\mathbf{x}_d)] \tag{4.2}$$

where λ is a non-negative factor and E is the total degree of constraint violation. In addition, H_{wall} and H_{slab} are the degrees of wall and slab constraint violation, respectively. They are defined as

$$H_{wall}(\mathbf{x}_d) = \frac{L'_W(\mathbf{x}_d)}{L_W}, \tag{4.3}$$

$$H_{slab}(\mathbf{x}_d) = \frac{A'_S(\mathbf{x}_d) + A''_S(\mathbf{x}_d)}{A_S}. \tag{4.4}$$

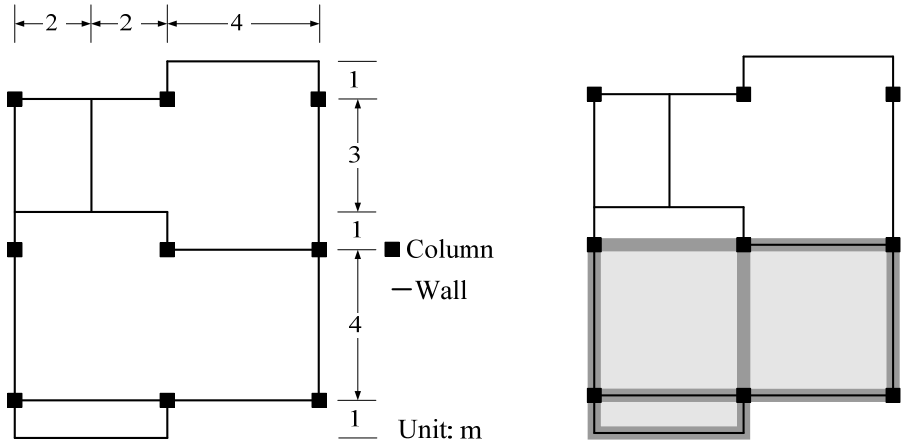
In Eq. (4.3), L'_W denotes the total length of wall segments that are not directly supported by beams. In addition, L_W denotes the total wall length. In Eq. (4.4), A'_S is the total area of slabs that have at least one side longer than the corresponding prescribed maximum length and A''_S is the total floor area that is not covered by slabs. Here, A_S is the total floor area.

In this study, the factor λ will be automatically determined in each GA generation using the adaptive penalty scheme proposed by Nanakorn and Meesomklin (2001). Brief details of the employed adaptive penalty scheme are presented in Section 3.8. By employing the penalty function in Eq. (4.2), the augmented fitness function F^a is obtained as

$$\begin{aligned} F^a(\mathbf{x}_d) &= F(\mathbf{x}_d) && \text{if } \mathbf{x}_d \in \tilde{\mathbf{F}}, \\ F^a(\mathbf{x}_d) &= F(\mathbf{x}_d) - P(\mathbf{x}_d) && \text{otherwise} \end{aligned} \tag{4.5}$$

where $\tilde{\mathbf{F}}$ denotes the feasible search space with respect to the wall and slab constraints.

Fitness and constraints using the above equations are explained by an example in Fig. 4.9. In the figure, a rectilinear floor and its example layout are given. The floor has three areas or rooms. The example layout has three slabs. The two top slabs have a score of 1 because they have four corner columns. The bottom slab has a score of 0.5 because it has only two corner columns. The total slab score is 1+1+0.5=2.5. The number of slabs is 3. Hence, the raw fitness computed using Eq. (4.1) is 2.5/3=0.83333. The total length of the walls is 51 m. The total length of the walls that are not supported by beams is 25 m. Hence, the wall constraint violation computed using Eq. (4.3) is 25/51=0.49019. By assuming the maximum slab size to be 4 m, all three slabs in the example layout do not violate the slab size constraint. Thus, A'_S in Eq. (4.4) is zero.



Slab score = 2.5
Wall constraint = 0.49019
Slab constraint = 0.5
Raw fitness = 0.83333
Augmented fitness = 0.83333-0.99019 λ

Fig. 4.9. Examples of fitness and constraint calculations.

Obviously, the top half of the example layout has no slab. Thus, A_S'' in Eq. (4.4) is 36 m^2. The total floor area of the given floor is 72 m^2. Therefore, the slab constraint violation computed using Eq. (4.4) is (0+36)/72=0.5. Hence, the penalty value P computed using Eq. (4.2) is (0.49019+0.5) λ =0.99019 λ. Finally, the augmented fitness in Eq. (4.5) becomes 0.83333-0.99019 λ. Noted that the value of λ can be computed after raw fitness and constraint violations of all individuals in the population are known. The value of λ is computed using the adaptive penalty scheme as explained earlier. After obtaining λ, the augmented fitness of all individuals can be computed.

4.4.3 Elitism

As noted earlier, the proposed coding scheme for beam-slab layouts introduces the bias toward layouts with larger slabs and fewer beams. To alleviate the bias, the layout that contains all possible beam segments is inserted into the initial population. With proper grid spacing, this special layout is always feasible since it always satisfies both wall and slab constraints. To make certain that the influence of this insertion does not disappear during the GA operators, elitism is employed in the proposed GA. The main concept of all elitist GAs is that the best solution or solutions are placed directly in the population of the subsequent generation regardless of the reproduction, crossover and mutation operators. In this study, the elitist solution is the best individual determined by using the following elitism rule of comparison.

Consider two layouts, $Layout_i$ and $Layout_j$, that are geometrically valid. $Layout_i$ is better than $Layout_j$ when

1. $Layout_i$ is feasible while $Layout_j$ is not.
2. Both layouts are feasible but $Layout_i$ has larger fitness than $Layout_j$.
3. Both layouts are feasible and have the same fitness. Nevertheless, $Layout_i$ has a shorter total length of beams than $Layout_j$.
4. Both layouts are feasible and have the same fitness and total length of beams. Nevertheless, $Layout_i$ has fewer beams than $Layout_j$. Note that connecting beam segments on the same grid line are counted as one beam. Fig. 4.10 shows examples of how the number of beams is counted.
5. Both layouts are infeasible but $Layout_i$ has a smaller total degree of constraint violation than $Layout_j$.
6. Both layouts are infeasible and have the same total degree of constraint violation. Nevertheless, $Layout_i$ has larger fitness than $Layout_j$.
7. Both layouts are infeasible and have the same fitness and total degree of constraint violation. Nevertheless, $Layout_i$ has a shorter total length of beams than $Layout_j$.
8. Both layouts are infeasible and have the same fitness and total degree of constraint violation. In addition, they also have the same total length of beams. Nevertheless, $Layout_i$ has fewer beams than $Layout_j$.

Since all individuals are always interpreted as geometrically valid layouts through their derived layouts, they can always be compared using the above elitism rule of comparison. Due to the inserted special individual in the initial population and the elitism process, there will always be at least one feasible individual in each generation. As a result, to obtain the elitist solution, it is actually not necessary to consider the comparison between two infeasible individuals. However, the comparison between two infeasible individuals is necessary for finding the worst individual to be replaced by the elitist solution in the elitism process. In this study, the elitism process is simply done by

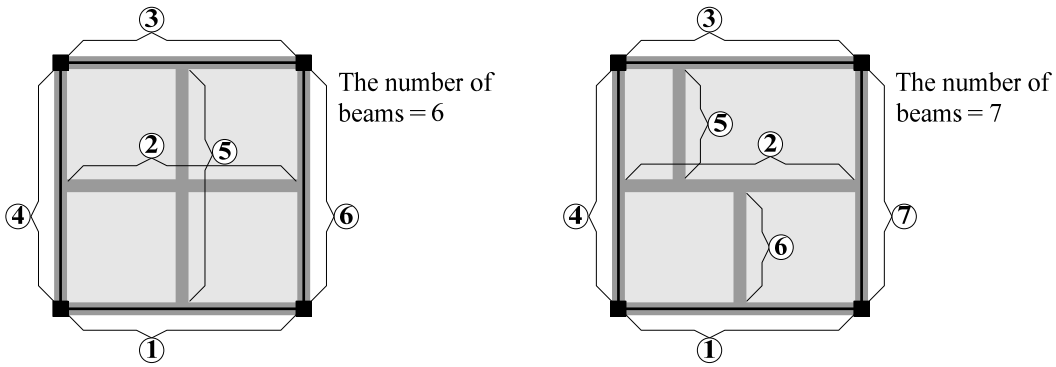


Fig. 4.10. The number of beams.

finding the best individual of the current population based on the above elitism rule of comparison. If this best individual is, based on the same rule of comparison, better than the existing elitist solution obtained from all the past generations, then the individual becomes the elitist solution. After that, this updated elitist solution is used to replace the worst individual of the generation. The worst individual of the generation is obtained also by using the above elitism rule of comparison. The elitism rule of comparison is used only to compare the superiority of individuals for the elitism process. In the reproduction process, the relative superiority of individuals is also considered. However, the reproduction process constructs the mating pool of each generation by using the scaled fitness values. Nevertheless, it can be seen that the best individual obtained by the elitism rule of comparison will also be the best feasible individual that has the highest fitness.

4.5 Algorithm

The GA operators used in this study include the roulette wheel selection, one-point crossover, and bitwise mutation. In addition, the elitism and the adaptive penalty scheme are employed. The flow chart of the proposed GA is shown in Fig. 4.11. The proposed GA for beam-slab layout design also can be summarized as follows:

Algorithm GA_for_Beam_Slab_Layout_Design
Input: An architectural floor plan, the maximum allowable slab length, the required number of generations (N_G), and the population size (N)
Output: The best beam-slab layout that is the elitist solution obtained from all generations

- Randomly generate the initial population of $N-1$ individuals
- Add one individual containing all possible beam segments to the initial population
- Create the derived chromosome strings from the original strings by mapping
- Determine the fitness and degrees of constraint violation of all individuals
- Find the best individual of the generation based on the elitism rule of comparison and set it as the elitist solution
- Obtain the augmented fitness of all individuals based on the adaptive penalty scheme
- Obtain the scaled fitness
- Implement the roulette-wheel selection, the crossover operator and the mutation operator
- Replace the old population with the new one
- **for** $j \leftarrow 1$ to N_G **do**
    - Create the derived chromosome strings from the original strings by mapping
    - Determine the fitness and degrees of constraint violation of all individuals
    - Find the best and worst individuals of the generation based on the elitism rule of comparison and update the elitist solution if necessary

- Replace the worst individual of the generation with the elitist solution
- Obtain the augmented fitness of all individuals based on the adaptive penalty scheme
- Obtain the scaled fitness
- Implement the roulette-wheel selection, the crossover operator and the mutation operator
- Replace the old population with the new one.

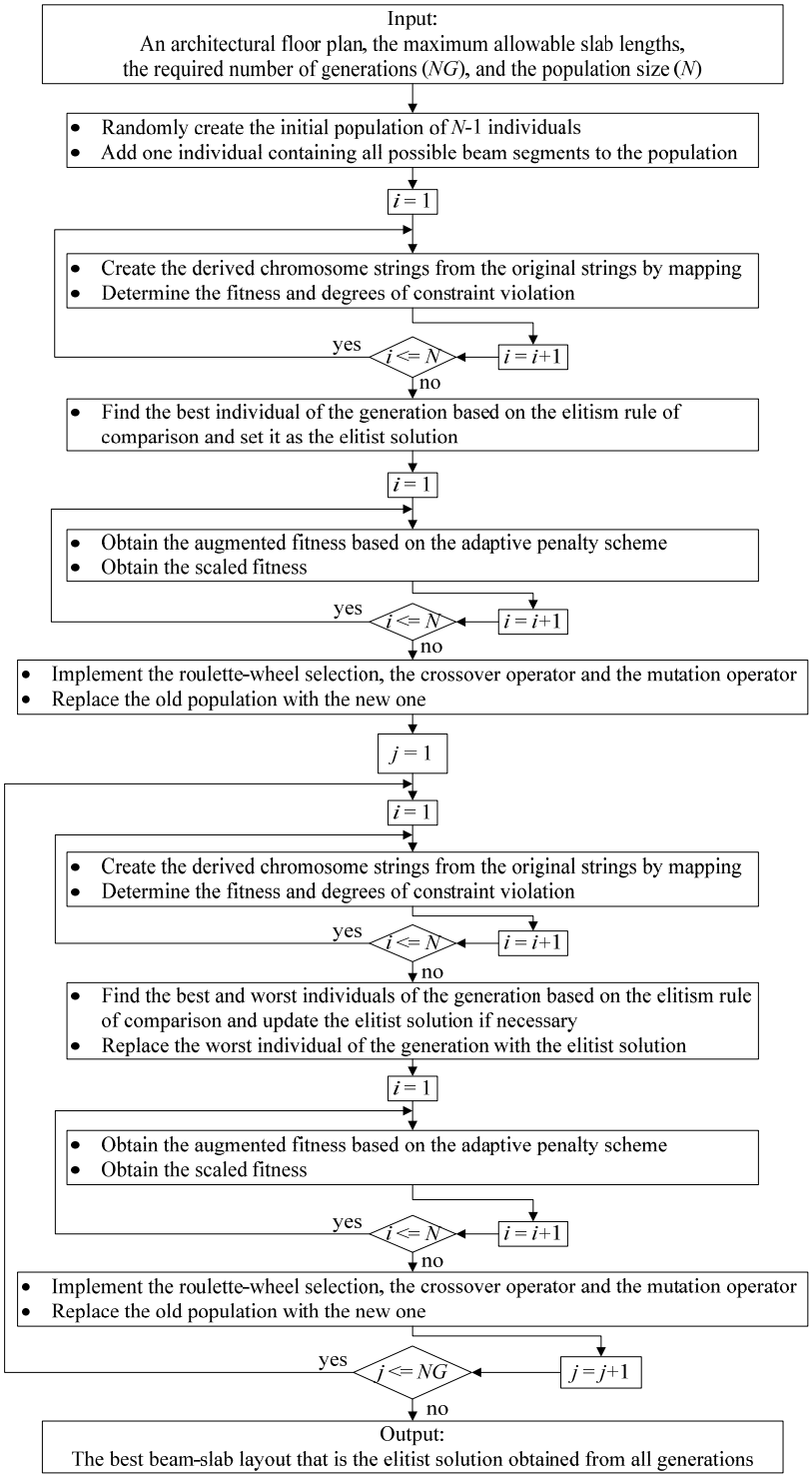


Fig. 4.11. Flow chart of the proposed GA.

Chapter 5

Examples and Results

To show the validity of the proposed GA for beam-slab layout design, the algorithm is used to solve nine beam-slab layout problems. The input data of all the problems are architectural floor plans with given positions of walls and columns. Since it is apparent that there must be beams on the outer boundary of the floor, beam segments are placed in advance on all line segments representing the outer boundary. Thus, these beam segments are removed from the list of the design variables. In all problems, the parameters C and φ in the fitness scaling and adaptive penalty processes are set to four and one, respectively. Setting $C = 4$ means that the individual chance of the best feasible members being selected into the mating pool is equal to four times that of the average feasible members. Setting $\varphi = 1$ means that the individual chance of the best infeasible members being selected into the mating pool is equal to that of the average feasible members. All together, the two parameters indicate that the individual chance of the best feasible members being selected into the mating pool is equal to four times that of the best infeasible members. In the GA process, the crossover probability of 0.85 and the mutation probability of 0.005 are used for all problems. To allow the efficiency of the proposed algorithm to be clearly discussed, the problems are solved by using various population sizes. In order to examine both the quality and uniformity of the obtained results, the algorithm is run for 100 times for each population size. The number of generations used for all runs in Problems 1 to 4 is 500 while the number of generations used in Problems 5 to 9 is 2,000. The 100 runs for each population size are collectively called a calculation set. The best solution of a run is the best layout found in that run, which is the elitist solution obtained from all generations. Since there are 100 runs in a calculation set, there will be 100 best solutions from these 100 runs. Among these 100 best solutions, the best one determined by the elitism rule of comparison will be the best solution of the calculation set. Note again that all individuals in the algorithm are always interpreted as geometrically valid layouts represented by their derived strings. As a result, solutions are shown here by using their derived layouts.

5.1 Problem 1

The first problem is an architectural floor plan shown in Fig. 5.1a. It can be seen from the positions of the walls that the floor consists of three separate areas. Two of these areas are of rectangular shapes while the third area is not. A uniform grid with spacing of 0.5 m is employed as shown in Fig. 5.1b. Rather small spacing is used here to show the validity of the proposed algorithm since smaller spacing results in a larger search space. By placing beams on the boundary of the floor in advance, this grid results in 418 design variables. The maximum allowable length of a slab is preset to 4 m. For this problem, four calculation sets for four different population sizes of 100, 200, 300

and 400 individuals are analyzed. As aforementioned, each calculation set consists of 100 runs.

Table 5.1 shows the statistics of the obtained results. For each population size or calculation set, the maximum, average, minimum and standard deviation (SD) values of the fitness of the 100 best solutions from the 100 runs are found. Note that the maximum fitness obtained from each calculation set is the fitness of the best solution among the 100 best solutions obtained from the 100 runs. It is found from the results that the best solutions of all calculation sets are in fact the same and this best layout is shown in Fig. 5.1c. It can be seen that the best layout in Fig. 5.1c is a feasible beam-slab layout that satisfies both wall and slab constraints. In addition, the layout is unquestionably a good layout that can really be used in the next design process. Three large slabs in the layout are suitably supported, through beams, by their corner columns. In the area where the presence of the walls necessitates more beams, smaller slabs are appropriately created. Note that this study does not intend to claim that this best layout from the algorithm is the best possible layout. In fact, the best possible layout can never be identified since different designers will have their own opinions of what the best layout should be. From Table 5.1, it can also be seen that the SD values for all calculation sets are very small. In fact, the maximum coefficient of variation of all calculation sets, which is found in the calculation set with the population size of 100, is only 0.13. This means that the algorithm provides rather uniform results. The table also reports the appearance percentage of the best solution of each calculation set. The obtained percentages for all calculation sets are very high, especially for the population

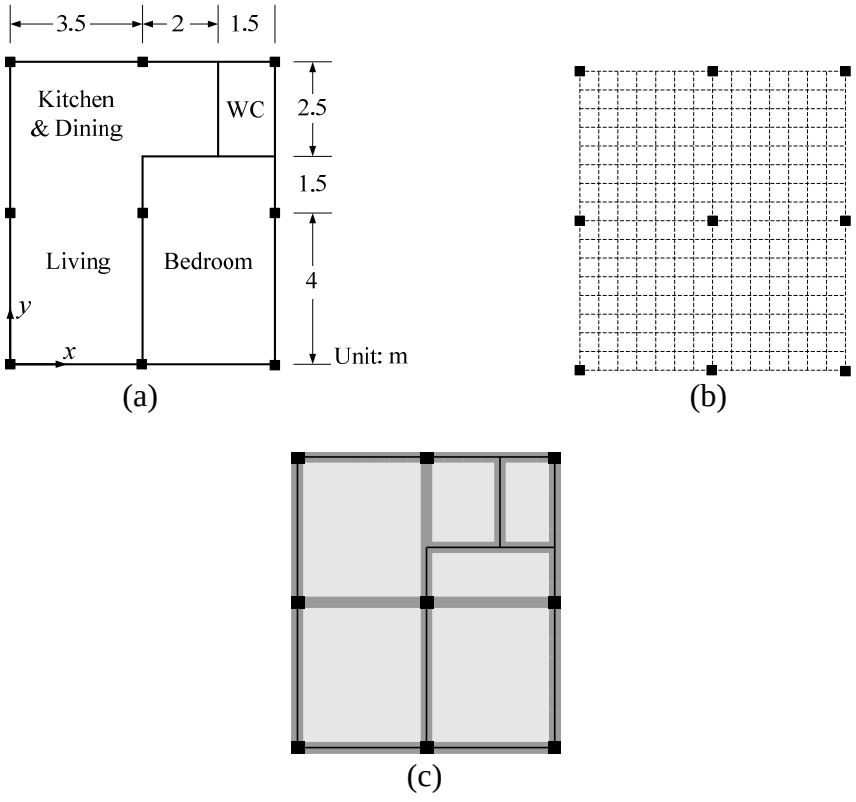


Fig. 5.1. Problem 1. (a) The given floor plan. (b) The grid. (c) The best solution.

sizes of 300 and 400. Also in the table, the average numbers of generations required for the solution convergence are reported. It can be seen that the algorithm requires, on average, less than 140 generations for obtaining its best layout. In fact, for the population size of 400, the algorithm requires, on average, less than 100 generations to get the best layout.

Fig. 5.2 shows an example evolution of layouts from a run with 100 individuals. The best layout of the run is obtained at the 128th generation. The best layout obtained is also the problem's best layout, which is shown in Fig. 5.1c. In addition, Fig. 5.3 shows the convergences of the fitness and the total beam length of the elitist solution from the same run shown in Fig. 5.2. It can be seen that the fitness rises quickly during the first 120 generations and reaches its convergence at the 128th generation. Concurrently, the total beam length decreases rapidly during the first 120 generations and reaches the lowest value of 51 m at the 128th generation. The decrease of the total beam length during the evolutionary process is expected even though the total beam length is not directly included in the objective function. As aforementioned, this is because, with all other conditions being the same, the employed fitness function encourages layouts that have fewer slabs and layouts with fewer slabs have fewer beams.

Table 5.1. Problem 1: statistics of the results.

	Calculation set of 100 runs			
	Population Size			
	100	200	300	400
Maximum fitness	0.667	0.667	0.667	0.667
Average fitness	0.636	0.652	0.665	0.664
Minimum fitness	0.214	0.389	0.500	0.428
SD of fitness	0.080	0.055	0.016	0.024
Appearance percentage of the best solution of the calculation set (%)	86	93	99	99
Average required number of generations for the solution convergence	130.1	113.8	110.8	94.2

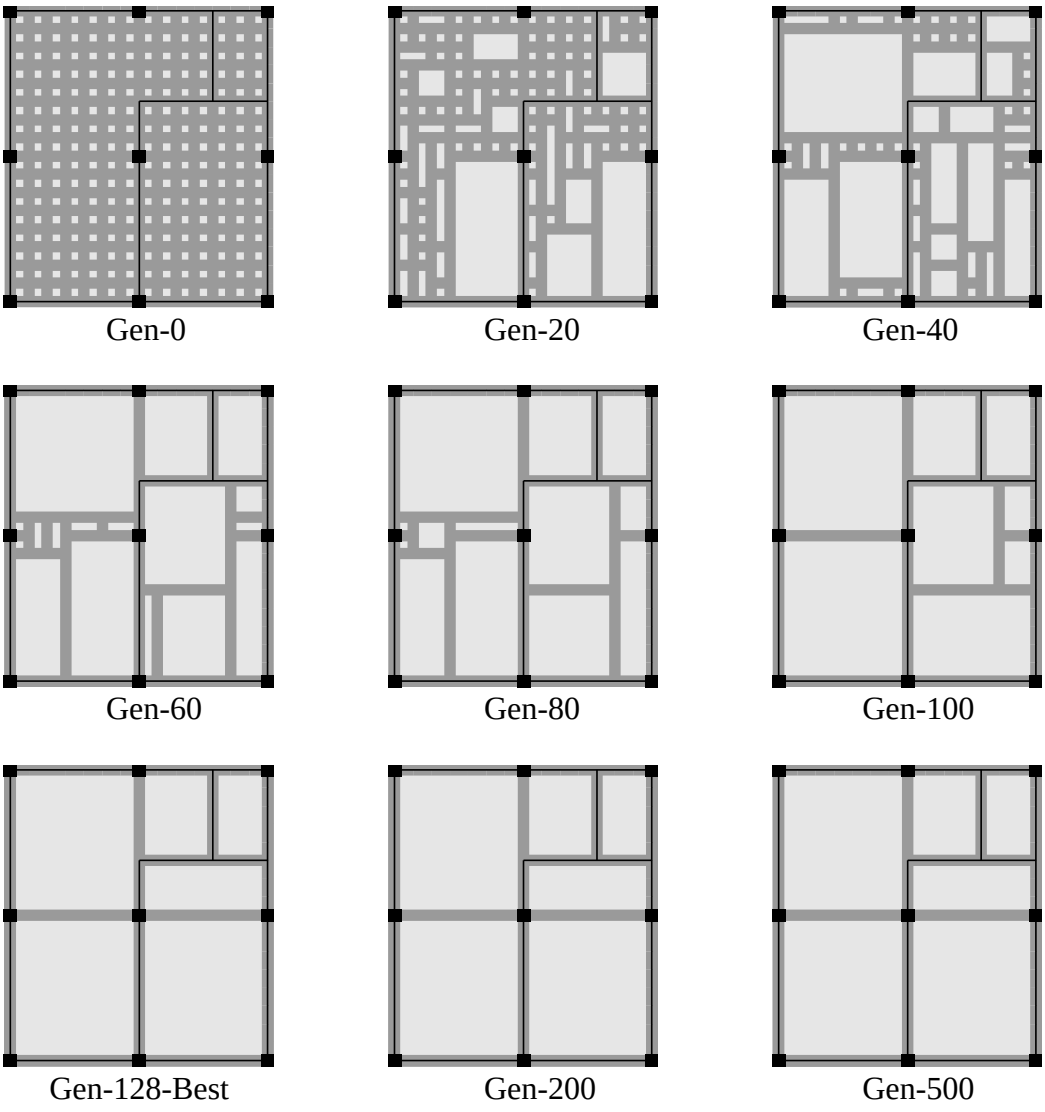


Fig. 5.2. Problem 1: a typical evolution of solution.

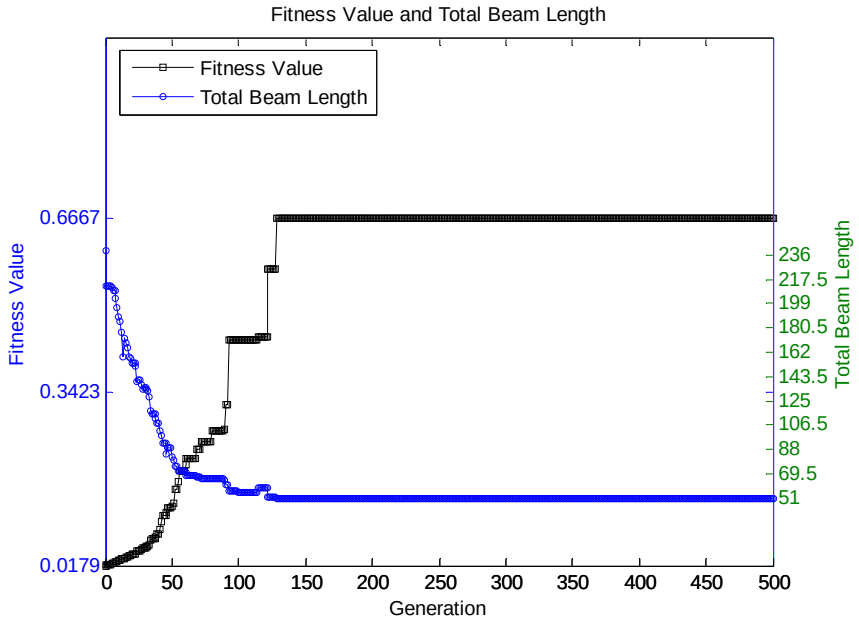


Fig. 5.3. Problem 1: a typical development of the fitness and the total beam length.

5.2 Problem 2

The second problem is an architectural floor plan shown in Fig. 5.4a. For this problem, a uniform grid with spacing of 0.5 m is employed as shown in Fig. 5.4b. By placing beams on the boundary of the floor in advance, this grid results in 480 design variables. Similar to the previous problem, the maximum allowable length of a slab is preset to 4 m. In addition, four calculation sets for four different population sizes of 100, 200, 300 and 400 individuals are also analyzed and each calculation set also consists of 100 runs.

Table 5.2 shows the statistics of the obtained results. Similar to the previous problem, the best solutions of all calculation sets are the same as shown in Fig. 5.4c and is found to be a good layout. From Table 5.2, the SD values for all calculation sets are very small. In fact, the maximum coefficient of variation of all calculation sets, which is found in the calculation set with the population size of 100, is only 0.06. Actually, for the population sizes of 300 and 400, the SD values are zero. This is because all 100 best solutions of all 100 runs within each of these two calculation sets are the same. These results further confirm that the proposed algorithm provides uniform results. It can be observed also from Table 5.2 that, for the population sizes of 200 and greater, the algorithm requires, on average, less than 100 generations to reach the convergence.

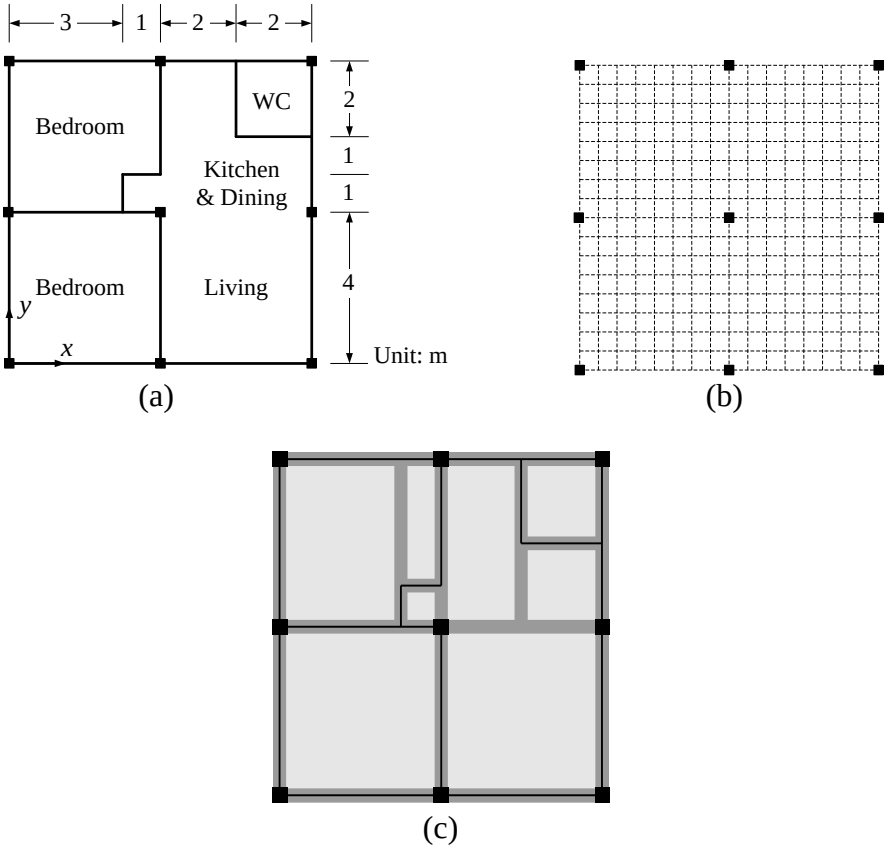


Fig. 5.4. Problem 2. (a) The given floor plan. (b) The grid. (c) The best solution.

Fig. 5.5 shows a typical evolution of layouts from a run with 100 individuals. The best layout of the run is obtained at the 124th generation. The best layout obtained is also the problem's best layout, which is illustrated in Fig. 5.4c. In addition, Fig. 5.6 depicts the convergences of the fitness and the total beam length of the same run shown in Fig. 5.5. The fitness rises quickly after the 75th generation and reaches its convergence at the 124th generation. The total beam length decreases rapidly also after the 75th generation and reaches the lowest value of 59 m at the 124th generation.

Table 5.2. Problem 2: statistics of the results.

	Calculation set of 100 runs			
	Population Size			
	100	200	300	400
Maximum fitness	0.500	0.500	0.500	0.500
Average fitness	0.490	0.498	0.500	0.500
Minimum fitness	0.375	0.375	0.500	0.500
SD of fitness	0.030	0.014	0	0
Appearance percentage of the best solution of the calculation set (%)	89	97	100	100
Average required number of generations for the solution convergence	124.6	99.2	98.1	76.6

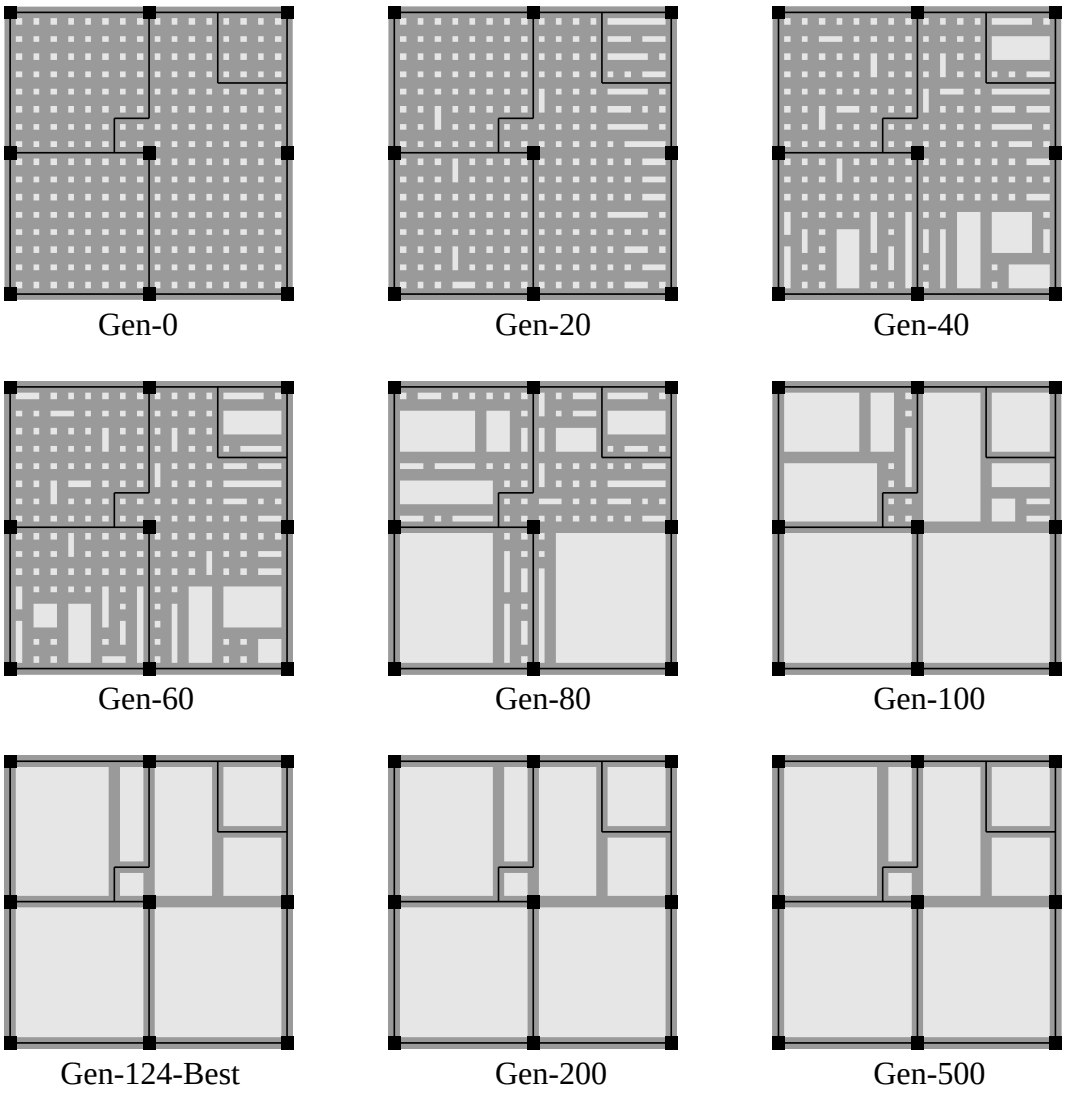


Fig. 5.5. Problem 2: a typical evolution of solution.

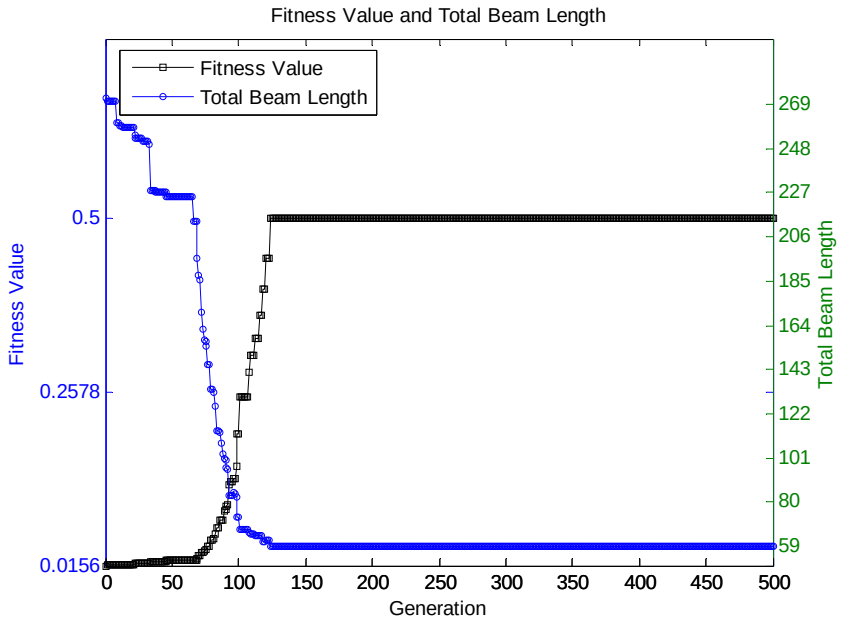
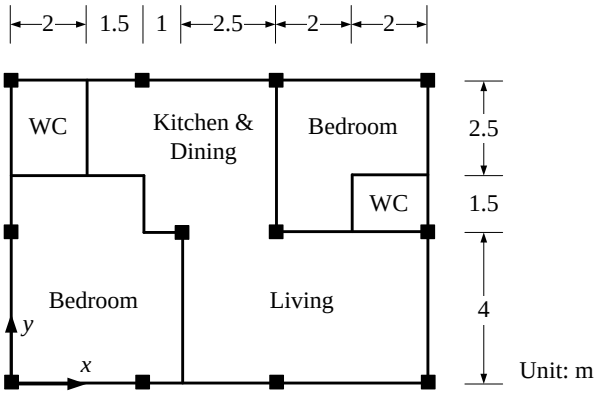


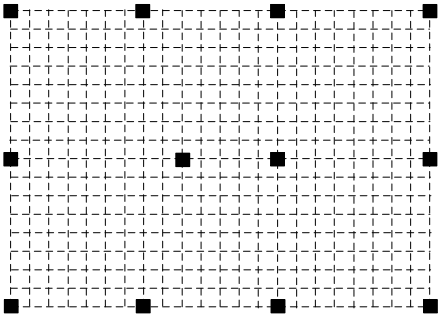
Fig. 5.6. Problem 2: a typical development of the fitness and the total beam length.

5.3 Problem 3

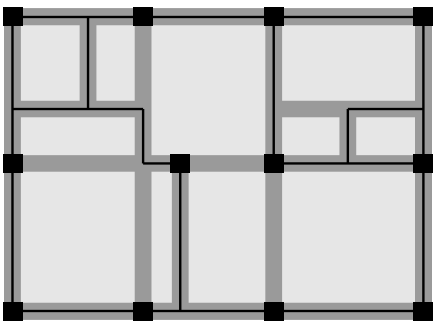
The third problem is an architectural floor plan shown in Fig. 5.7a. This floor is larger and more complicated than the previous two examples. The floor has one interior column that is out of alignment with the other columns. This irregularity is often encountered by engineers in practice and is intentionally adjusted herein to increase the difficulty of the problem. Similar to the previous problems, a uniform grid with spacing of 0.5 m is employed for this problem as shown in Fig. 5.7b. With the outer boundary beams placed in advance, this grid yields 666 design variables. The maximum allowable length of a slab is preset to 4 m. For this problem, four calculation sets for four different



(a)



(b)



(c)

Fig. 5.7. Problem 3. (a) The given floor plan. (b) The grid. (c) The best solution.

population sizes of 500, 600, 700 and 800 individuals are analyzed. Similar to the previous problems, each calculation set consists of 100 runs.

Table 5.3 shows the statistics of the obtained results. The best solutions of all calculation sets are the same. The best layout from the algorithm is shown in Fig. 5.7c. The algorithm efficiently provides simple beam-slab patterns in the area with simple wall lines and beam-slab patterns that are more involved in the area with complex wall lines. In addition, the algorithm handles the area around the column that is out of alignment quite well. As presented in Table 5.3, the SD values for all calculation sets are very small. The maximum coefficient of variation of all calculation sets, found in the calculation set with the population size of 500, is only 0.02. For this problem, the algorithm requires, on average, less than 190 generations to obtain the convergence.

Fig. 5.8 shows a typical evolution of layouts from a run with 500 individuals. The best layout of the run, which is also the best layout found for this problem shown in Fig. 5.7c, is obtained at the 187th generation. In Fig. 5.8, there is the change of pattern from Gen-150 to Gen-187 because the bottom left slab of Gen-187 has more corner columns than that of Gen-150. Fig. 5.9 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.8. The fitness rises quickly during the first 75 generations before reaching its convergence at the 187th generation. Similarly, the total beam length decreases rapidly during the first 75 generations and reaches the lowest value of 80.5 m at the 187th generation.

Table 5.3. Problem 3: statistics of the results.

	Calculation set of 100 runs			
	Population Size			
	500	600	700	800
Maximum fitness	0.500	0.500	0.500	0.500
Average fitness	0.496	0.497	0.497	0.497
Minimum fitness	0.454	0.417	0.458	0.458
SD of fitness	0.012	0.012	0.010	0.009
Appearance percentage of the best solution of the calculation set (%)	92	94	94	95
Average required number of generations for the solution convergence	187.4	177.6	163.2	175.6

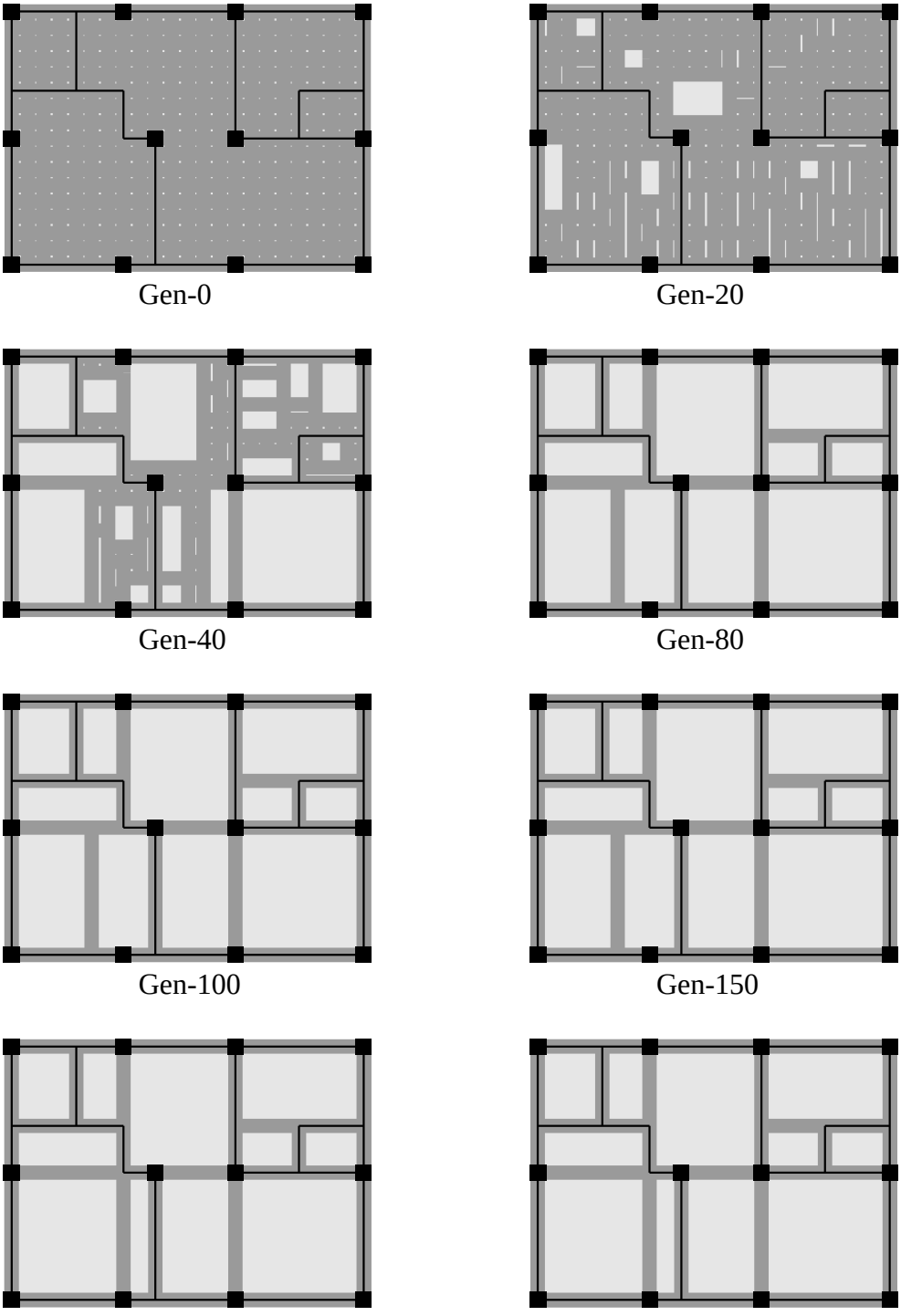


Fig. 5.8. Problem 3: a typical evolution of solution.

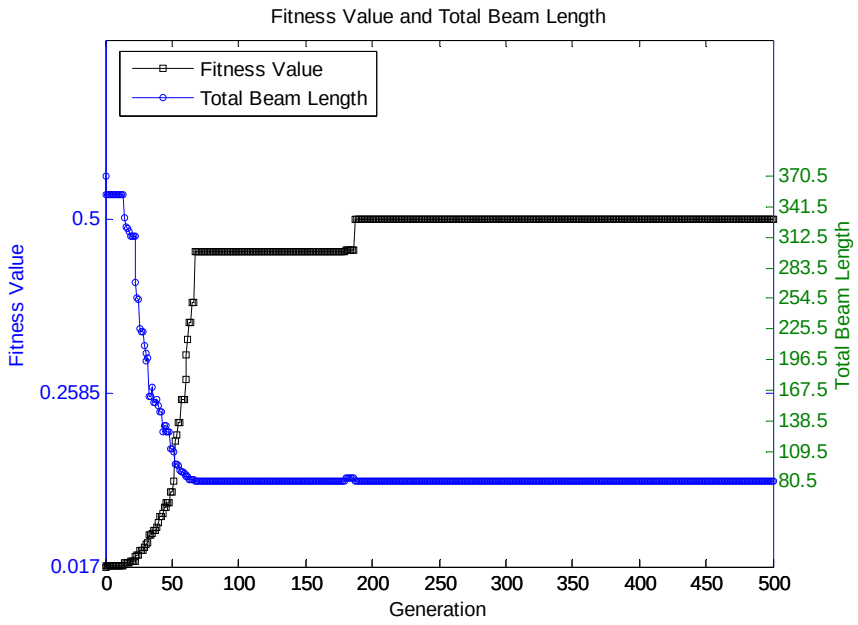
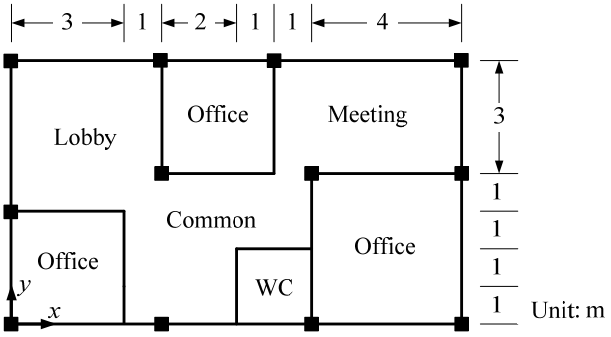


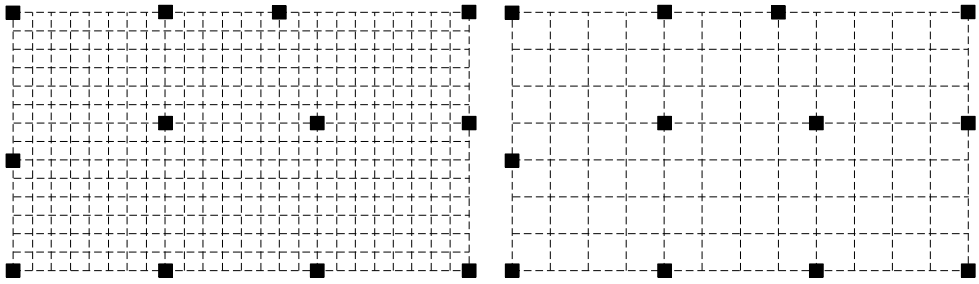
Fig. 5.9. Problem 3: a typical development of the fitness and the total beam length.

5.4 Problem 4

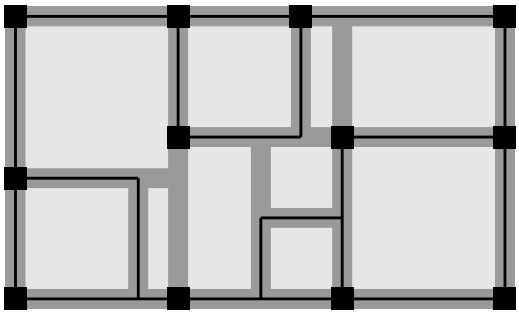
The fourth problem is an architectural floor plan shown in Fig. 5.10a. The floor has several rooms and the columns are not in perfect alignment. In this problem, two uniform grids with spacing of 0.5 m and 1 m are employed as shown in Fig. 5.10b and c. The larger grid spacing of 1 m is also used in this problem in order to show that, in real practice, larger values of spacing can be used as long as the values still allow the slab constraint to be satisfied. The maximum allowable length of a slab is again preset to 4 m. With the outer-boundary beams placed in advance, the 0.5-m grid requires 634 design variables while the 1-m grid involves 149 design variables. Four calculation sets for four different population sizes of 500, 600, 700 and 800 individuals are analyzed for the 0.5-m grid. For the 1-m grid, four different population sizes of 100, 200, 300 and



(a)



(b) (c)



(d)

Fig. 5.10. Problem 4. (a) The given floor plan. (b) The 0.5-m grid. (c) The 1-m gird. (d) The best solution.

400 individuals are used for the latter. Each calculation set consists of 100 runs. The smaller population sizes are adopted for the 1-m grid because the search space of the 1-m grid is smaller than that of the 0.5-m grid.

Table 5.4 shows the statistics of the obtained results for the 0.5-m grid. Similar to all previous problems, it is found from the results that the best solutions of all calculation sets are the same. The best layout is shown in Fig. 5.10d. It can be seen that the obtained best layout is reasonably good. All irregular wall and column locations are well taken care of by the algorithm. The layout can positively be used in the next design process. In addition, the SD values for all calculation sets shown in Table 5.4 are very small. The maximum coefficient of variation of all calculation sets, found in the calculation set with the population size of 500, is only 0.04. In addition, the algorithm requires, on average, less than 175 generations to obtain the convergence. Table 5.5 shows the statistics of the obtained results for the 1-m grid. The best solutions of all calculation sets with the 1-m grid are the same as the best layout obtained from the 0.5-

Table 5.4. Problem 4: statistics of the results for the 0.5-m grid.

	Calculation set of 100 runs			
	Population Size			
	500	600	700	800
Maximum fitness	0.550	0.550	0.550	0.550
Average fitness	0.541	0.543	0.543	0.543
Minimum fitness	0.500	0.500	0.500	0.500
SD of fitness	0.019	0.017	0.017	0.019
Appearance percentage of the best solution of the calculation set (%)	82	86	85	88
Average required number of generations for the solution convergence	167.3	171.7	165.9	157.1

Table 5.5. Problem 4: statistics of the results for the 1-m grid.

	Calculation set of 100 runs			
	Population Size			
	100	200	300	400
Maximum fitness	0.550	0.550	0.550	0.550
Average fitness	0.546	0.548	0.548	0. 550
Minimum fitness	0.500	0.500	0.500	0. 550
SD of fitness	0.014	0.010	0.009	0
Appearance percentage of the best solution of the calculation set (%)	90	95	97	100
Average required number of generations for the solution convergence	130.5	126.1	96.6	104.6

m grid. Nevertheless, the SD values for the calculation sets with the 1-m grid are much smaller than those from the 0.5-m grid. In addition, the maximum coefficient of variation of all calculation sets with the 1-m grid, found in the calculation set with the population size of 100, is only 0.03. Moreover, for the 1-m grid, the algorithm requires, on average, less than 135 generations to obtain the convergence.

Fig. 5.11 shows a typical evolution of layouts for the 0.5-m grid from a run with 500 individuals. The best layout of the run is obtained at the 159th generation. This best layout is also the best layout found for this problem, which is shown in Fig. 5.10d. Moreover, Fig. 5.12 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.11. It can be seen that the fitness rises quickly after the 80th generation and reaches its convergence at the 159th generation. The total beam length decreases rapidly after the 80th generation before reaching the lowest value of 76 m at the 159th generation.

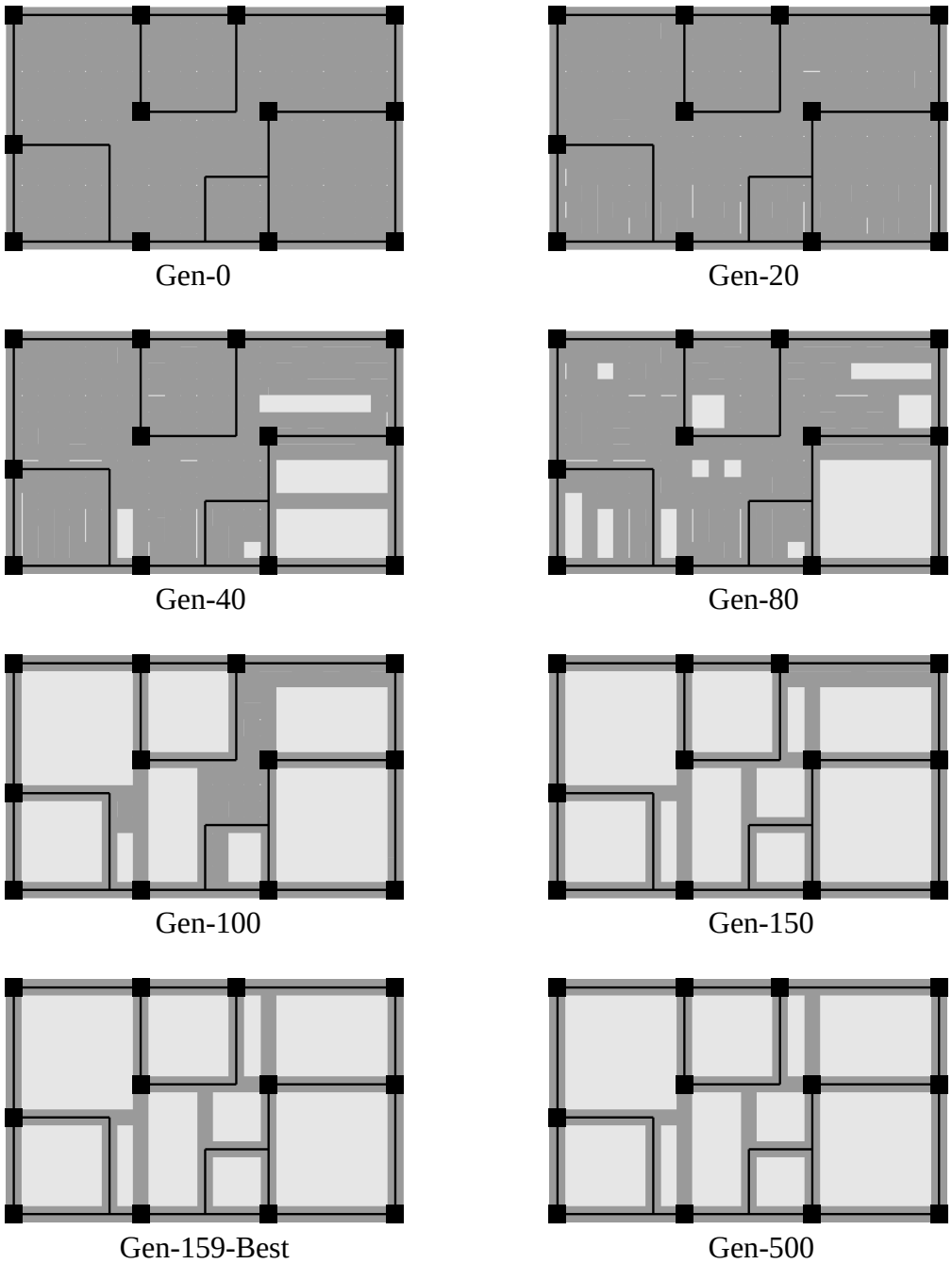


Fig. 5.11. Problem 4: a typical evolution of solution.

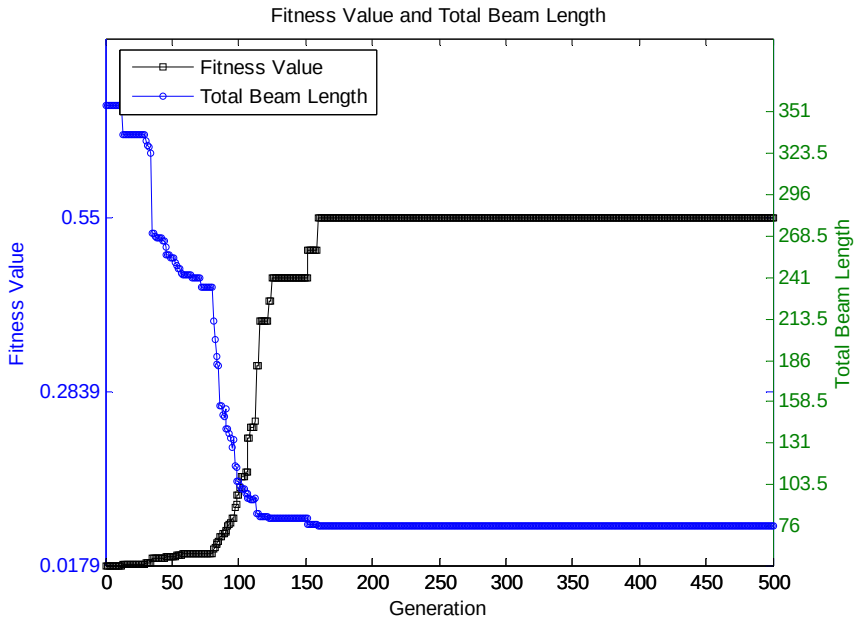
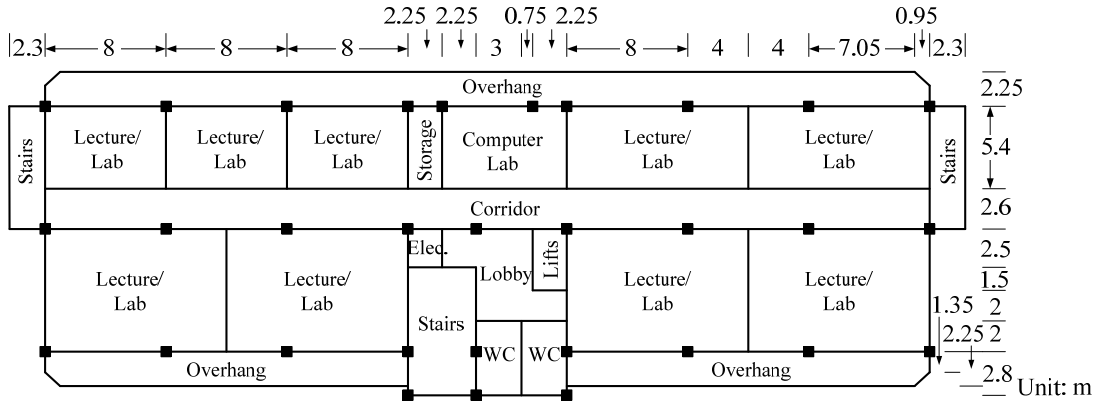


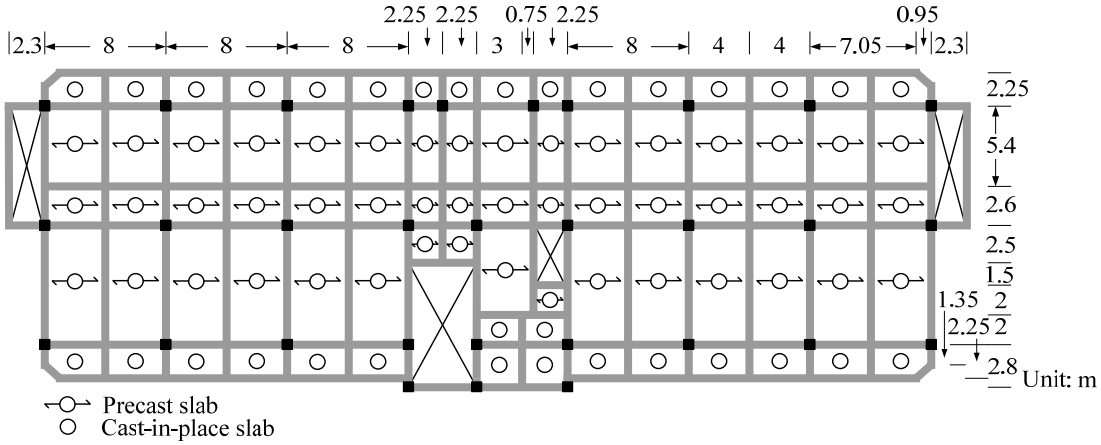
Fig. 5.12. Problem 4: a typical development of the fitness and the total beam length for the 0.5-m grid.

5.5 Problem 5

The fifth problem is taken from a floor plan of an existing public building. Fig. 5.13a shows the architectural floor plan of the building and Fig. 5.13b shows the real structural beam-slab layout. The architectural floor plan is slightly simplified to obtain a rectangular floor and the simplified plan is shown in Fig. 5.14a. This simplified plan is used as the input of the proposed algorithm. It can be seen from the real structural layout in Fig. 5.13b that both precast as well as cast-in-place slabs are used. The maximum length of the precast slabs is 4 m. In addition, the precast slabs are placed parallel to the *x* direction. To be able to compare the layout obtained from this study with the real structural layout, the maximum allowable slab length in the *x* direction is preset to 4 m in the algorithm. The maximum allowable slab length in the *y* direction is preset to 8 m, which is the maximum column spacing. In this problem, due to the complexity of the positions of walls and columns, a non-uniform grid is used. The maximum grid spacing is set to 4 m in order that the slab constraint can be satisfied in



(a)



(b)

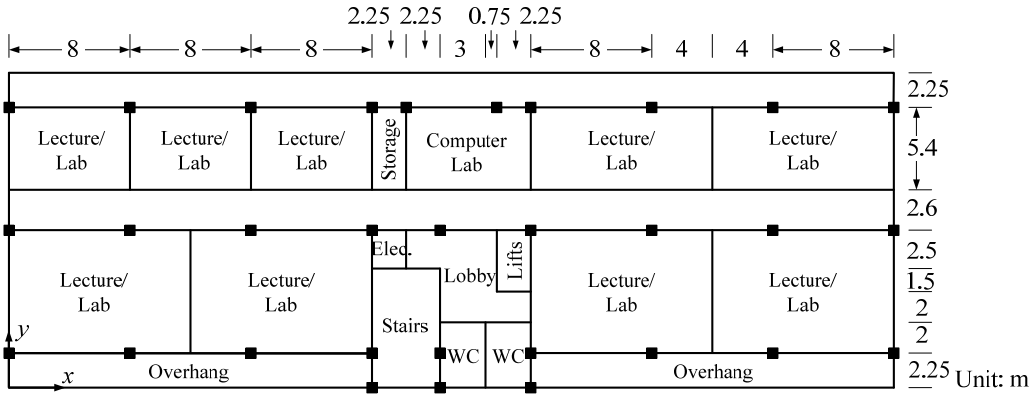
Fig. 5.13. Problem 5: (a) The real architectural floor plan. (b) The real structural floor plan.

both directions. To construct the non-uniform grid, grid lines are first placed on all columns and wall lines. After that, the spacing between all grid lines is checked. If any spacing is found to be greater than the maximum grid spacing of 4 m, an additional grid line will be inserted at the middle of the interval. Since there will be no beam in the stair and lift opening areas, the line segments of the grid in these two areas are removed. The obtained non-uniform grid is shown in Fig. 5.14b. With the outer-boundary beams placed in advance, this grid results in 269 design variables. For this problem, four calculation sets for four different population sizes of 20, 40, 60 and 80 individuals are analyzed. Each set consists of 100 runs. In the algorithm, the stair and lift openings are treated as part of the floor area, not as openings, and the calculation is performed as if there is no opening. However, a slab that fits any opening area exactly will not be penalized even if it violates the slab constraint.

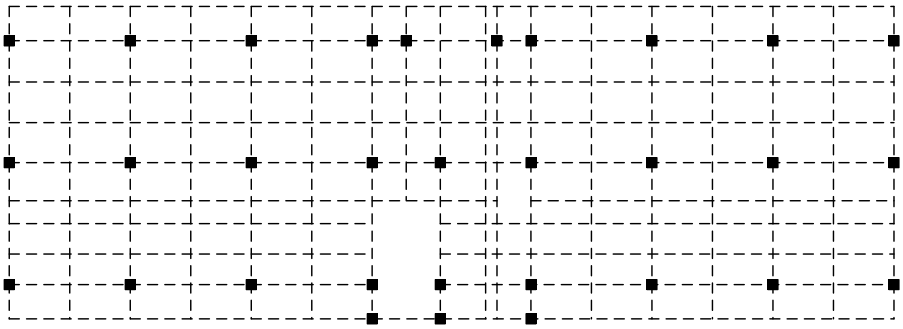
Table 5.6 shows the statistics of the obtained results. The best solutions of all calculation sets are found to be the same. The best layout from the algorithm is shown in Fig. 5.14c. This layout is found to be in good agreement with the real structural layout shown in Fig. 5.13b. In fact, if the simplified parts are disregarded, the two layouts are exactly the same. The SD values in Table 5.6 for all calculation sets are very small. The coefficients of variation of all calculation sets are found to be the same and equal to 0.01. For this problem, the algorithm requires, on average, less than 1,140 generations to obtain the convergence. Fig. 5.15 shows a typical evolution of layouts from a run with 20 individuals. Fig. 5.16 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.15.

Table 5.6. Problem 5: statistics of the results.

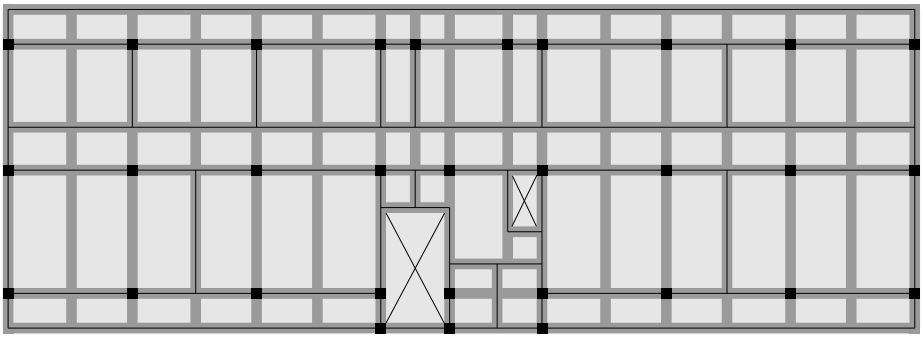
	Calculation set of 100 runs			
	Population Size			
	20	40	60	80
Maximum fitness	0.311	0.311	0.311	0.311
Average fitness	0.310	0.310	0.310	0.310
Minimum fitness	0.304	0.300	0.300	0.304
SD of fitness	0.002	0.002	0.002	0.002
Appearance percentage of the best solution of the calculation set (%)	37	52	42	46
Average required number of generations for the solution convergence	1074.3	942.8	1003.0	1136.6



(a)

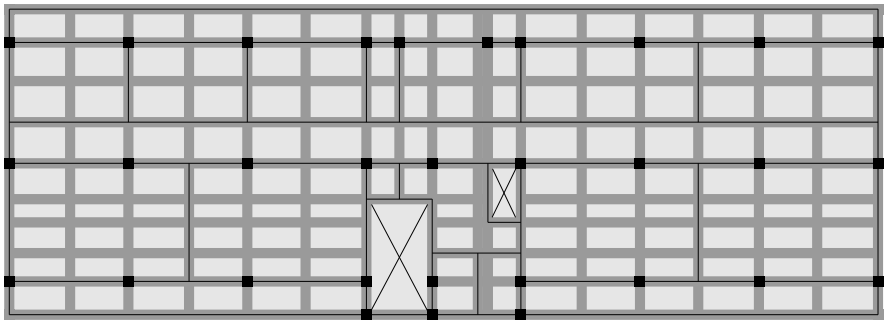


(b)

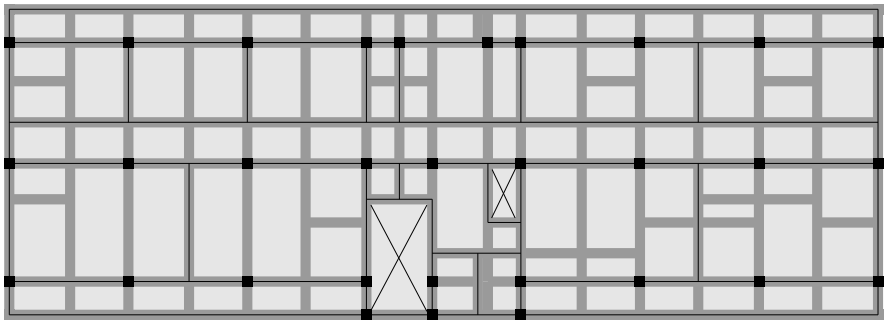


(c)

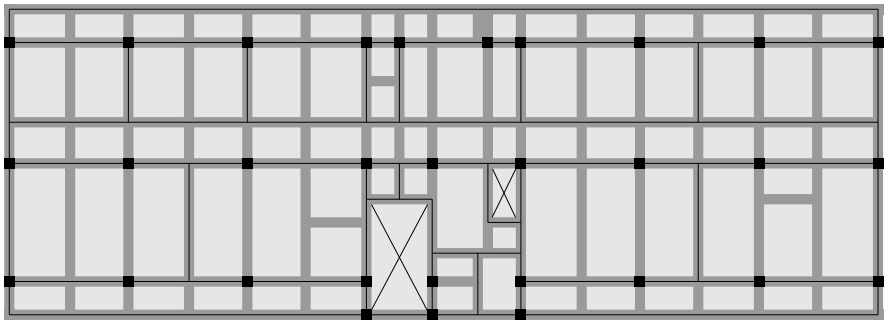
Fig. 5.14. Problem 5: (a) The simplified architectural floor plan. (b) The grid. (c) The best solution.



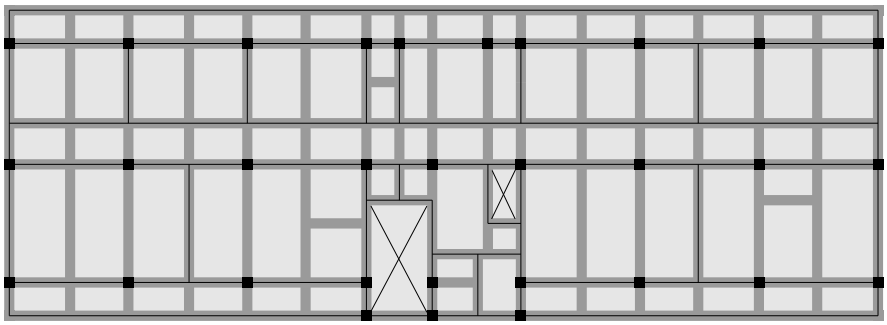
Gen-0



Gen-200

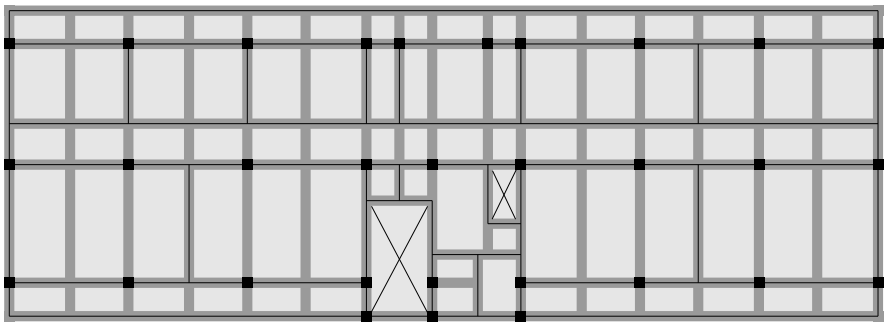


Gen-400

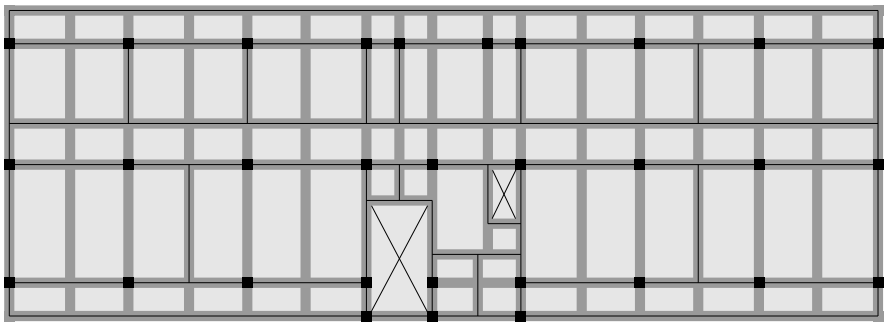


Gen-600

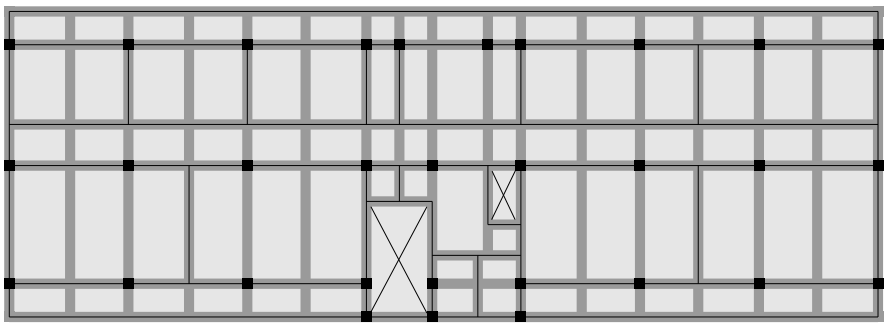
Fig. 5.15. Problem 5: a typical evolution of solution.



Gen-800



Gen-1004-Best



Gen-2000

Fig. 5.15. (*continued*)

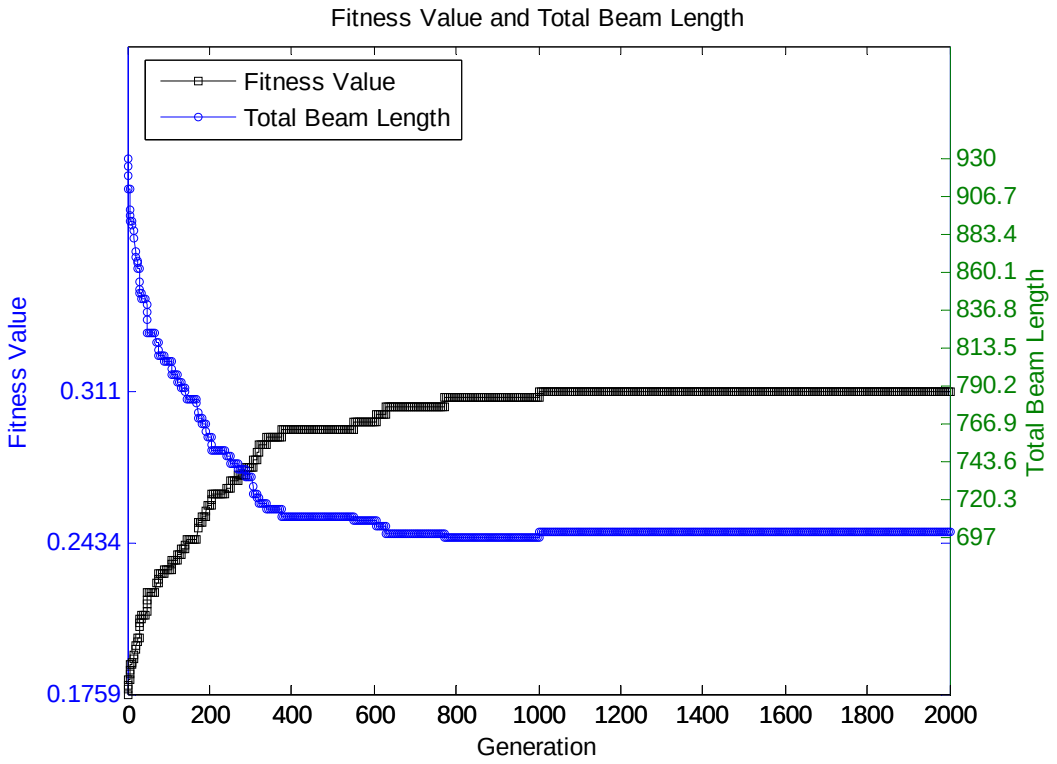


Fig. 5.16. Problem 5: a typical development of the fitness and the total beam length.

5.6 Problem 6

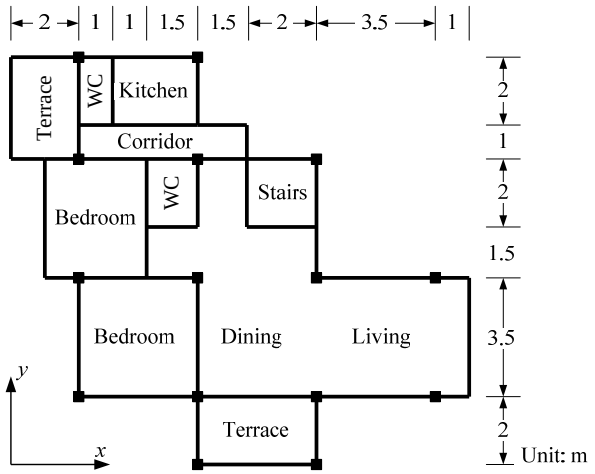
The sixth problem is a rectilinear floor plan shown in Fig. 5.17a. The floor has a stair opening. A uniform grid with spacing of 0.5 m is employed as shown in Fig. 5.17b. By placing beams on the outer boundary of the floor in advance, this grid results in 683 design variables. The maximum allowable length of a slab is set to 3.5 m. In addition, four calculation sets for four different population sizes of 20, 40, 60 and 80 individuals are analyzed, and each calculation set also consists of 100 runs.

Table 5.7 shows the statistics of the obtained results. It is found that the best solutions of all calculation sets are the same. This best solution is shown in Fig. 5.17c and is found to be a good layout. From Table 5.7, it can also be seen that the SD values for all calculation sets are small. In fact, the maximum coefficient of variation of all calculation sets, which is found in the calculation set with the population size of 20, is only 0.09. Table 5.7 shows that, for the population sizes of 40 and greater, the algorithm requires, on average, less than 1,000 generations to reach the solution convergence.

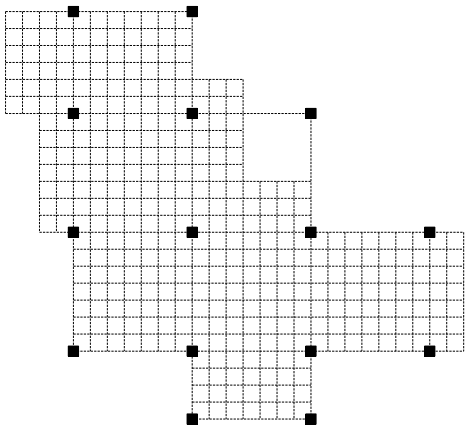
Fig. 5.18 shows a typical evolution of layouts from a run with 20 individuals. The best layout of the run is obtained at the 962th generation. This best layout is also the best layout found for this problem, which is shown in Fig. 5.17c. In addition, Fig. 5.19 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.18. The fitness rises quickly after the 50th generation and reaches its convergence at the 962th generation. Moreover, the total beam length decreases rapidly also after the 50th generation and reaches the lowest value of 101.5 m at the 962th generation.

Table 5.7. Problem 6: statistics of the results.

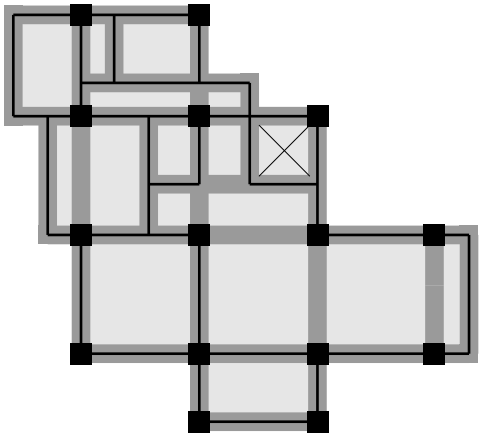
	Calculation set of 100 runs			
	Population Size			
	20	40	60	80
Maximum fitness	0.515	0.515	0.515	0.515
Average fitness	0.473	0.495	0.506	0.509
Minimum fitness	0.355	0.382	0.397	0.456
SD of fitness	0.044	0.031	0.022	0.016
Appearance percentage of the best solution of the calculation set (%)	38	60	78	88
Average required number of generations for the solution convergence	1078.5	999.1	859.2	918.0



(a)



(b)



(c)

Fig. 5.17. Problem 6. (a) The given floor plan. (b) The grid. (c) The best solution.

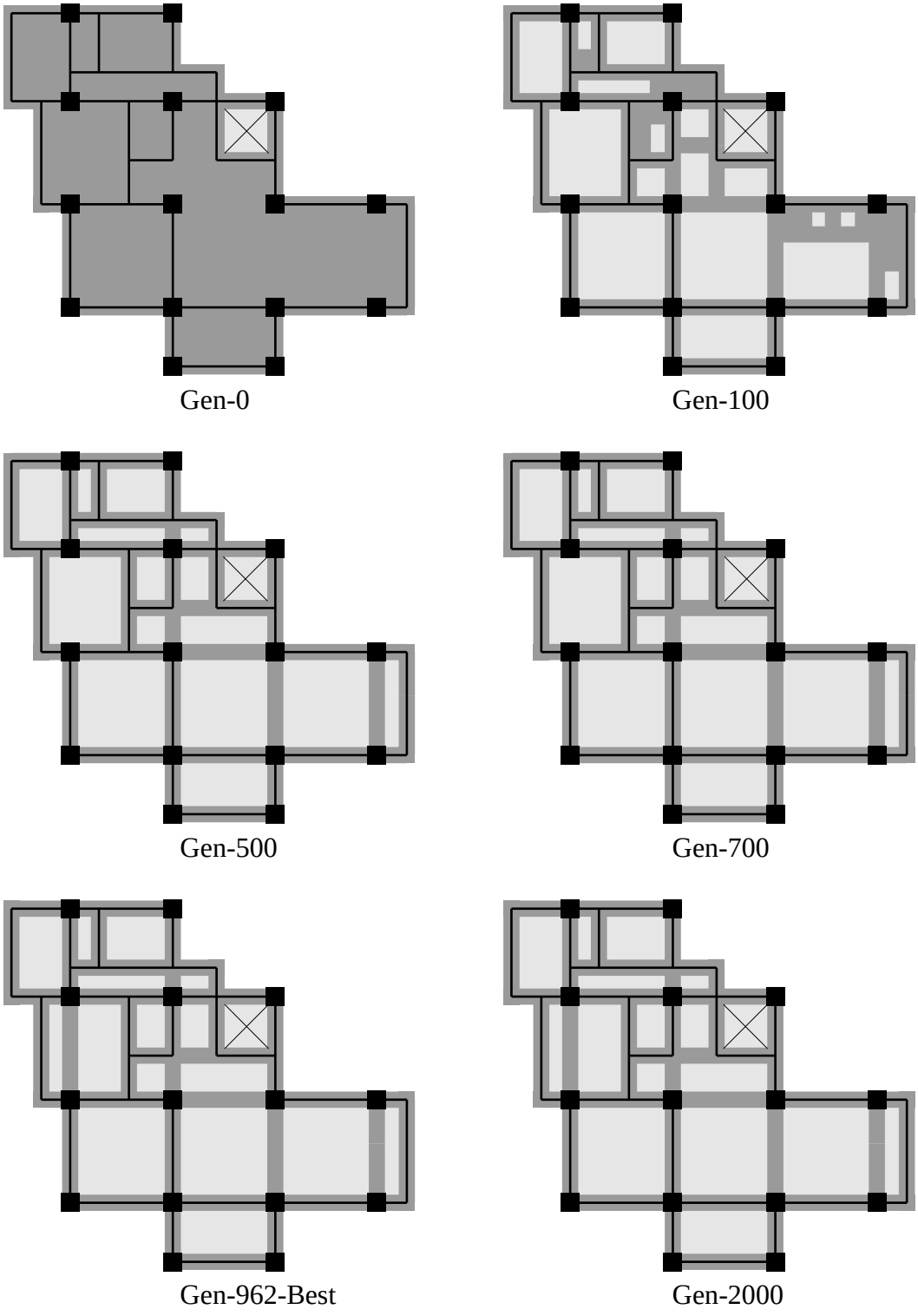


Fig. 5.18. Problem 6: a typical evolution of solutions.

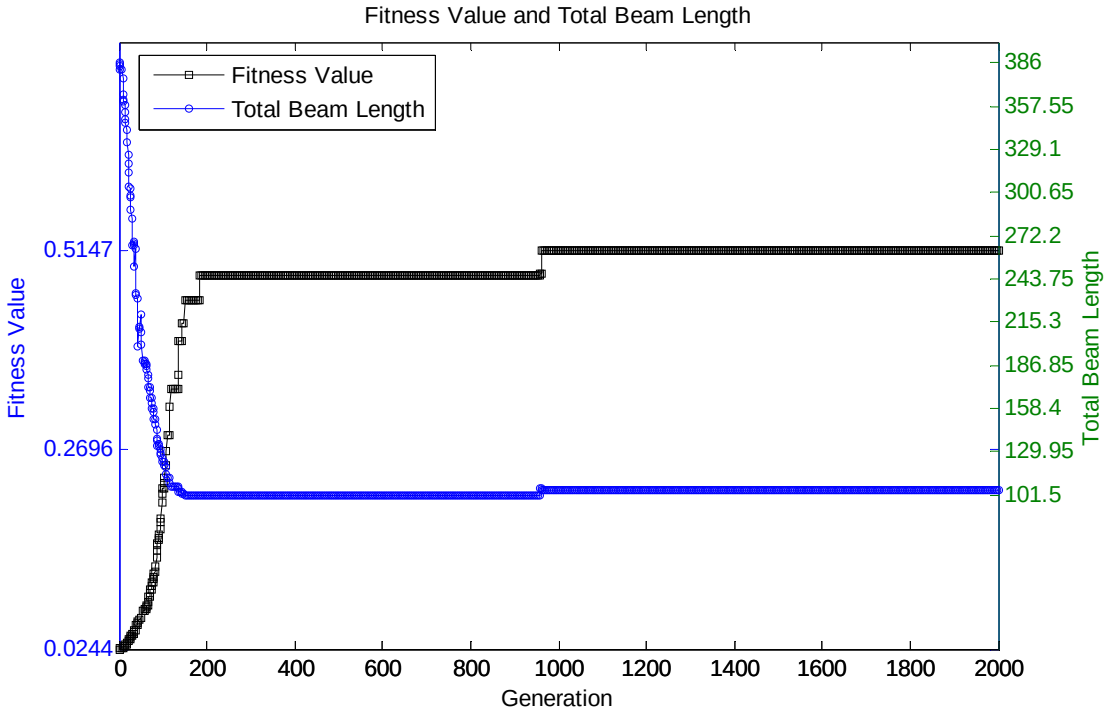
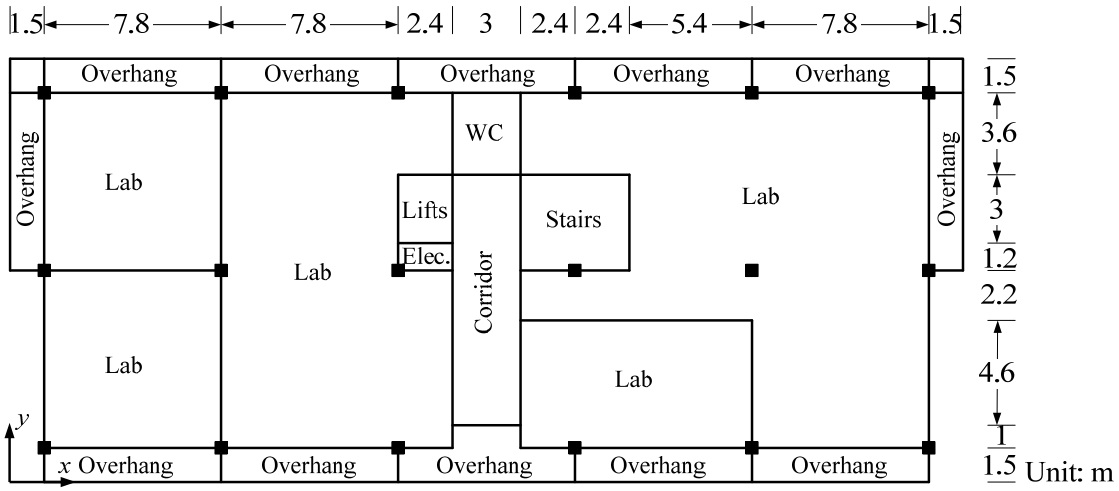


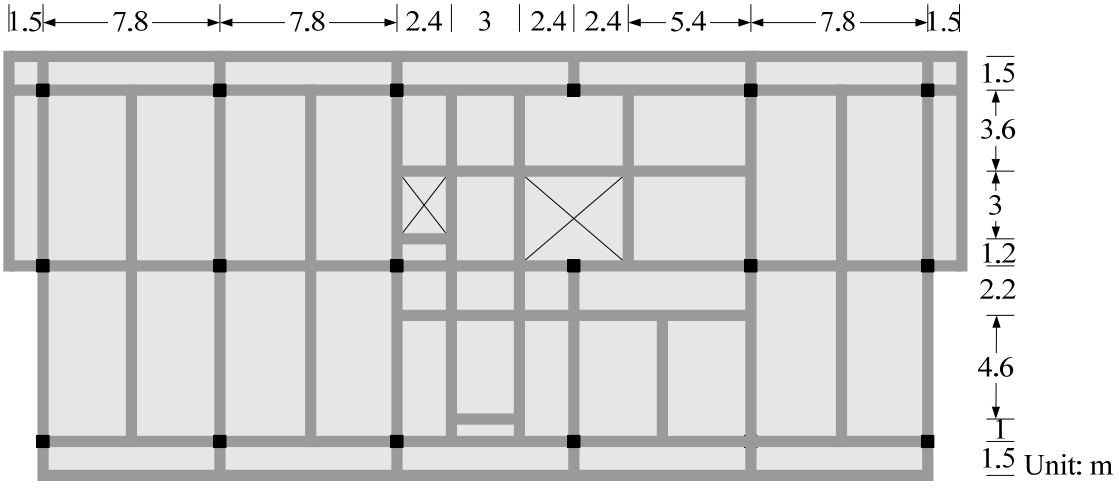
Fig. 5.19. Problem 6: a typical development of the fitness and the total beam length.

5.7 Problem 7

The seventh problem is taken from a floor plan of a real building. Fig. 5.20a shows the architectural floor plan of the building and Fig. 5.20b shows the real structural beam-slab layout. The plan in Fig. 5.20a is used as the input of the proposed algorithm. It can be seen from the real structural layout in Fig. 5.20b that one-way slabs with short spans of 3.9 m are mostly used in the floor. These one-way slabs are placed parallel to the *x* direction. To be able to compare the layout obtained from this study with the real structural layout, the maximum allowable slab length in the *x* direction is preset to 3.9 m. In addition, the maximum allowable slab length in the *y* direction is preset to 7.8 m, which is the maximum column spacing. Due to the complexity of the



(a)



(b)

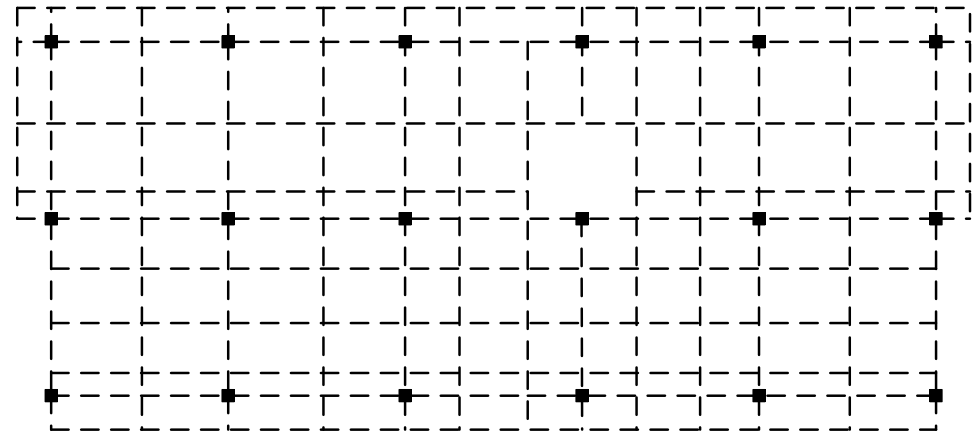
Fig. 5.20. Problem 7. (a) The real architectural floor plan. (b) The real structural floor plan.

positions of walls and columns, a non-uniform grid is used. The maximum grid spacing is set to 3.9 m in order that the slab constraint can be satisfied in both directions. The non-uniform grid is constructed using the same procedure as the previous example. Since there will be no beam in the stair and lift opening areas, the line segments of the grid in these two areas are removed. The obtained non-uniform grid is shown in Fig. 5.21a. With the outer-boundary beams placed in advance, this grid results in 203 design variables. For this problem, four calculation sets for four different population sizes of 20, 40, 60 and 80 individuals are analyzed. Each calculation set consists of 100 runs. The stair and lift openings are treated as part of the floor area, not as openings, and the calculation is performed as if there is no opening. However, a slab that fits any opening area exactly will not be penalized even if it violates the slab constraint.

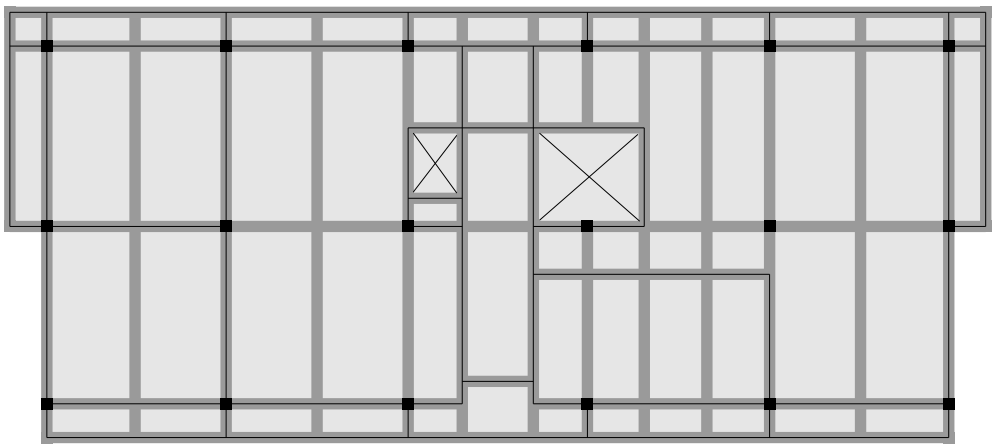
Table 5.8 shows the statistics of the obtained results. The best solutions of all calculation sets are found to be the same. The best layout from the algorithm is shown in Fig. 5.21b. This layout is found to be a good layout although it is not exactly the same as the real structural layout shown in Fig. 5.20b. The SD values in Table 5.8 for all calculation sets are very small. The maximum coefficient of variation of the calculation sets is equal to 0.004. For this problem, the algorithm requires, on average, less than 760 generations to obtain the convergence. Fig. 5.22 shows a typical evolution of layouts from a run with 20 individuals. Fig. 5.23 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.22.

Table 5.8. Problem 7: statistics of the results.

	Calculation set of 100 runs			
	Population Size			
	20	40	60	80
Maximum fitness	0.267	0.267	0.267	0.267
Average fitness	0.266	0.266	0.267	0.267
Minimum fitness	0.262	0.262	0.267	0.262
SD of fitness	0.001	0.001	0	0.
Appearance percentage of the best solution of the calculation set (%)	30	56	71	76
Average required number of generations for the solution convergence	649.8	705.5	731.4	754.9

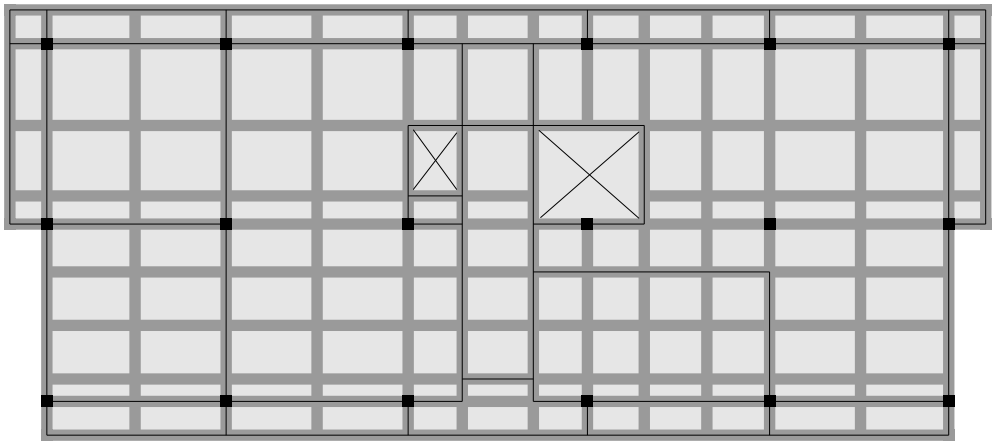


(a)

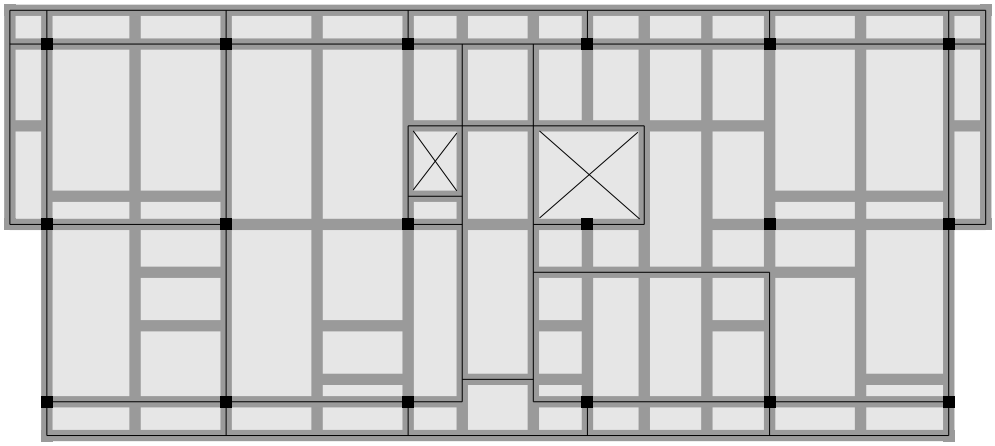


(b)

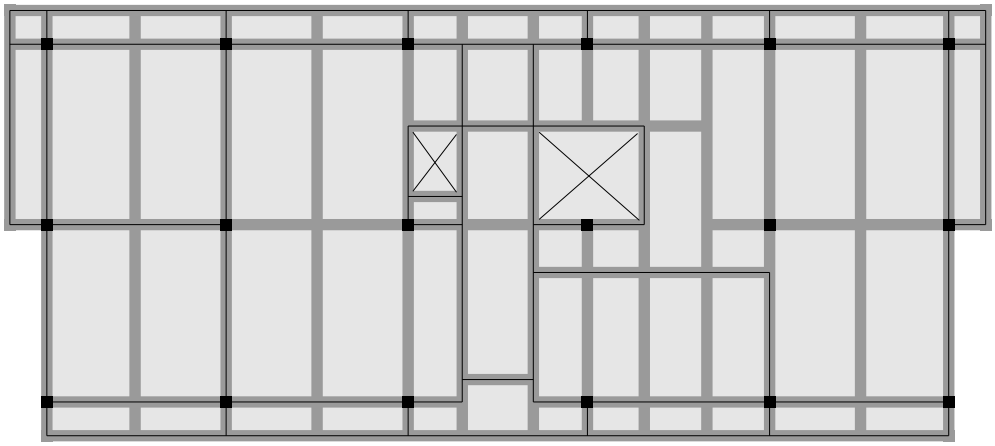
Fig. 5.21. Problem 7. (a) The grid. (b) The best solution.



Gen-0

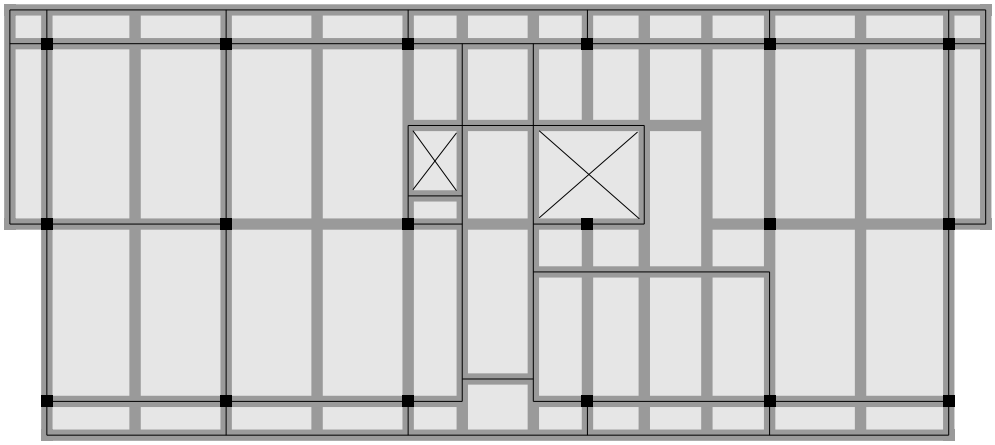


Gen-100

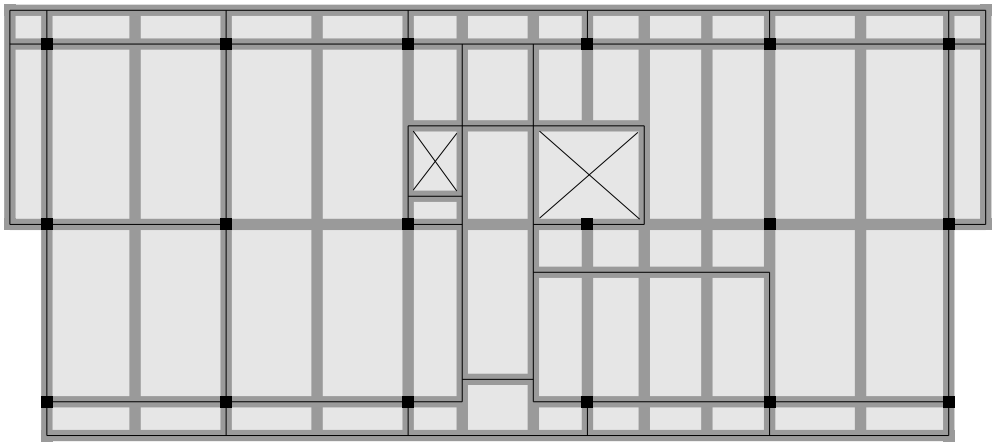


Gen-500

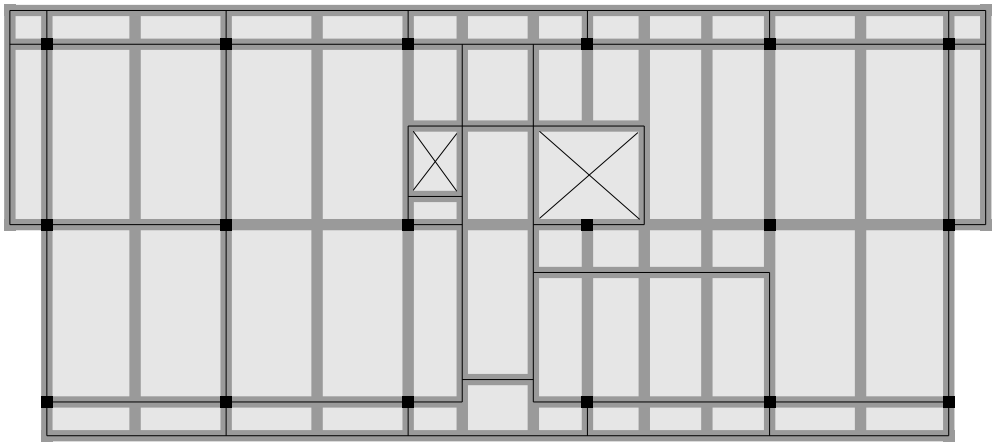
Fig. 5.22. Problem 7: a typical evolution of solutions.



Gen-700



Gen-725-Best



Gen-2000

Fig. 5.22. (*continued*)

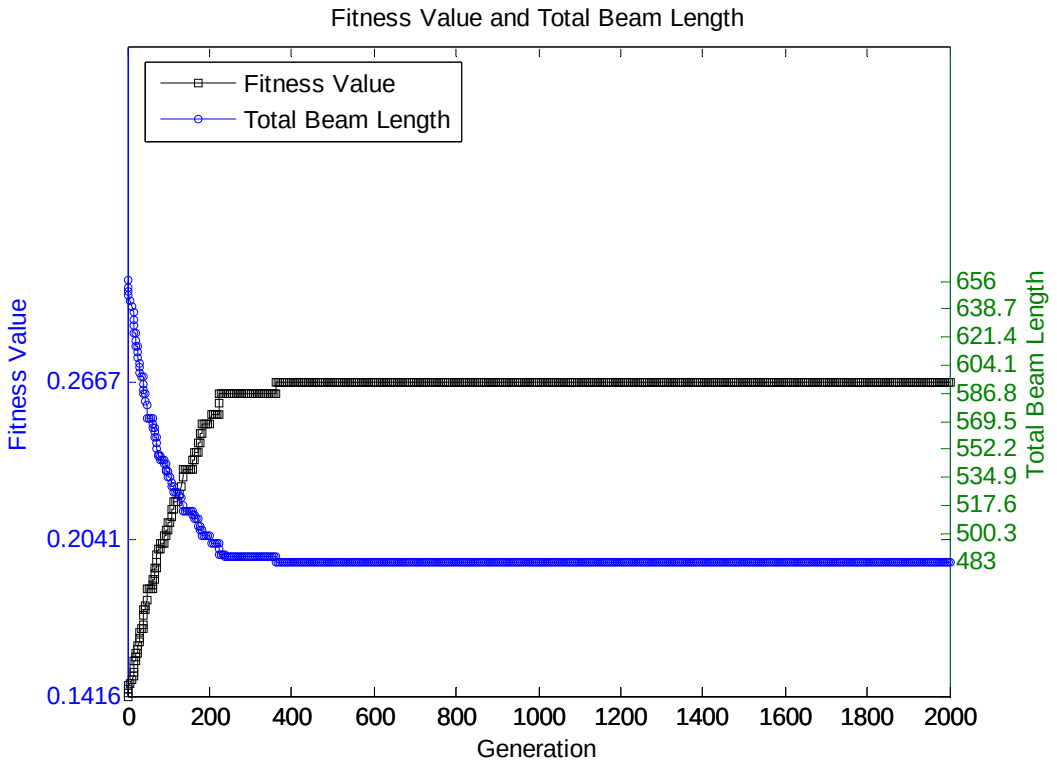
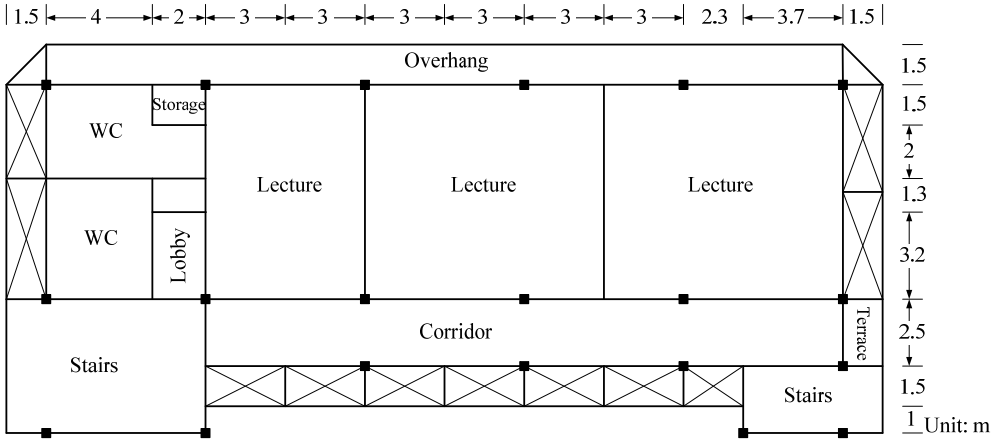


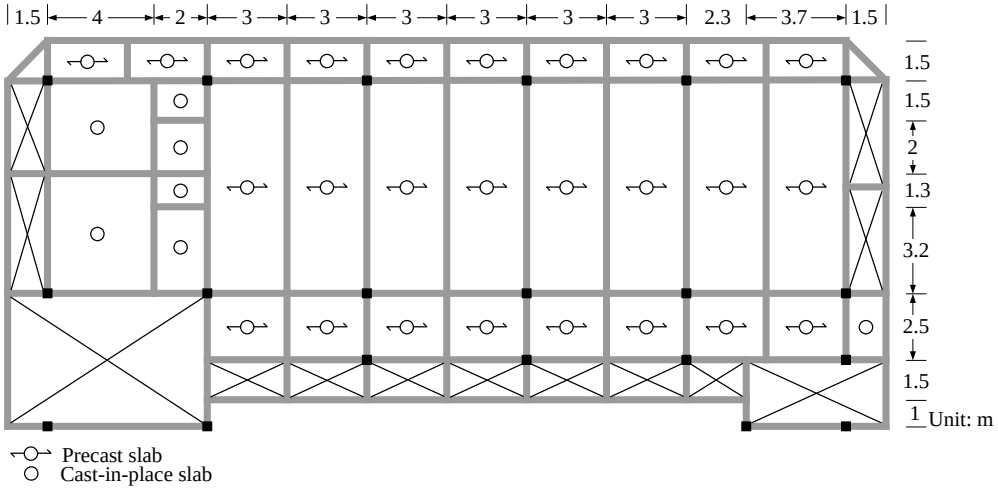
Fig. 5.23. Problem 7: a typical development of the fitness and the total beam length.

5.8 Problem 8

The eighth problem is also a real floor plan of an existing building. Fig. 5.24a shows the architectural floor plan of the building and Fig. 5.24b shows the real structural beam-slab layout. The angles of two floor corners are not right angles. As a result, they are changed to be right angles. The simplified floor plan is shown in Fig. 5.25a. This simplified plan is used as the input of the proposed algorithm. In the real structural layout in Fig. 5.24b, precast as well as cast-in-place slabs are used. The maximum length of the precast slabs is 3 m while the maximum length of the cast-in-place slabs is 4 m. In addition, the precast slabs are placed parallel to the *x* direction. To be able to compare the layout obtained from this study with the real structural layout, the maximum allowable slab length in the *x* direction is preset to 3 m. In addition, the maximum allowable slab length in the *y* direction is preset to 8 m, which is the maximum column spacing. In this problem, a non-uniform grid is used. The maximum



(a)



(b)

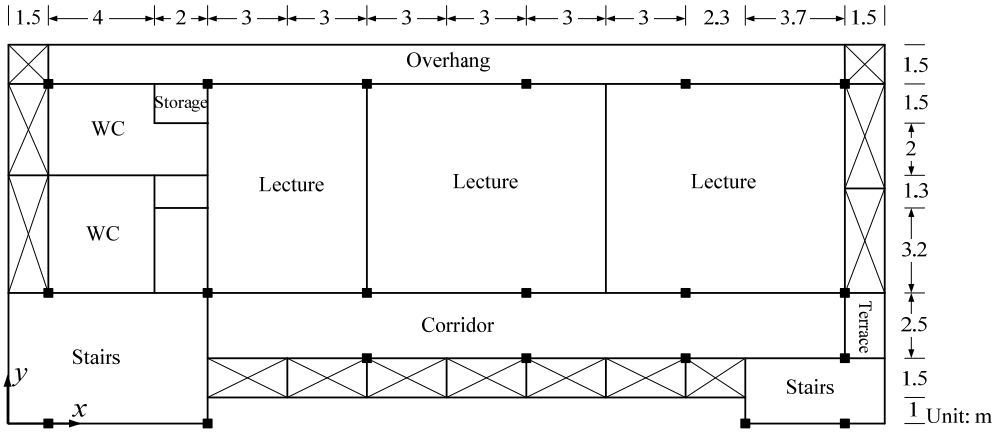
Fig. 5.24. Problem 8: (a) The real architectural floor plan. (b) The real structural floor plan.

grid spacing is set to 3 m in order that the slab constraint can be satisfied in both directions. Since there will be no beam in the stair opening areas and some other opening areas along the floor boundary, the line segments of the grid in these areas are removed. The obtained non-uniform grid is shown in Fig. 5.25b. With the outer-boundary beams placed in advance, this grid results in 199 design variables. For this problem, four calculation sets for four different population sizes of 20, 40, 60 and 80 individuals are analyzed. Each calculation set consists of 100 runs. In the algorithm, the openings are treated as part of the floor area, not as openings, and the calculation is performed as if there is no opening. However, a slab that fits any opening area exactly will not be penalized even if it violates the slab constraint.

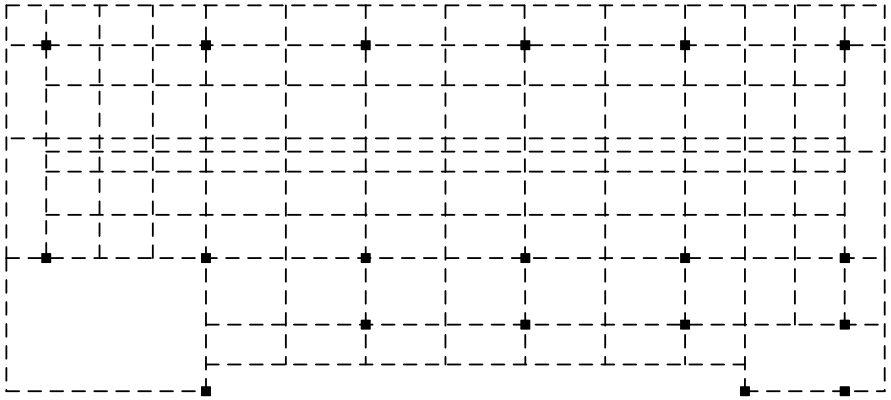
Table 5.9 shows the statistics of the obtained results. The best solutions of all calculation sets are found to be the same. The best layout from the algorithm is shown in Fig. 5.25c. This layout is found to be in good agreement with the real structural layout shown in Fig. 5.24b. The SD values in Table 5.9 for all calculation sets are zero. For this problem, the algorithm requires, on average, less than 430 generations to obtain the convergence. Fig. 5.26 shows a typical evolution of layouts from a run with 20 individuals. Fig. 5.27 shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.26.

Table 5.9. Problem 8: statistics of the results.

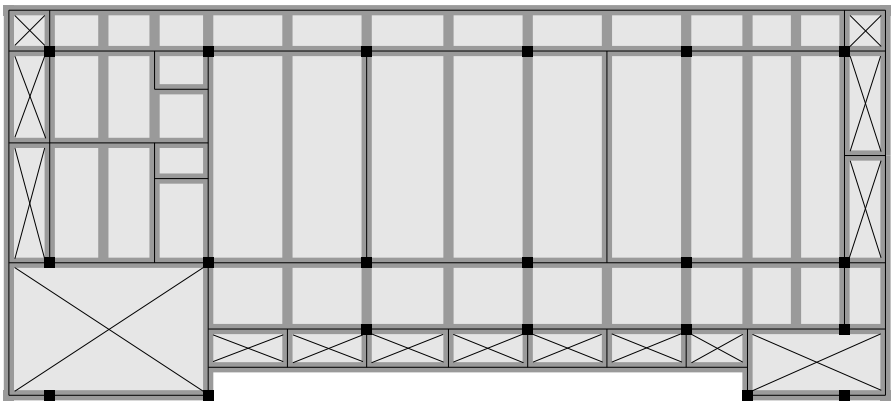
	Calculation set of 100 runs			
	Population Size			
	20	40	60	80
Maximum fitness	0.287	0.287	0.287	0.287
Average fitness	0.287	0.287	0.287	0.287
Minimum fitness	0.287	0.287	0.287	0.287
SD of fitness	0	0	0	0
Appearance percentage of the best solution of the calculation set (%)	100	100	100	100
Average required number of generations for the solution convergence	428.7	270.7	254.7	263.7



(a)

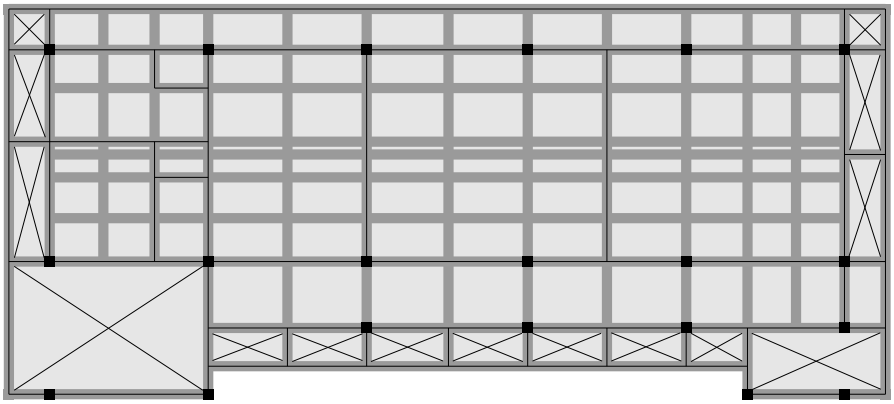


(b)

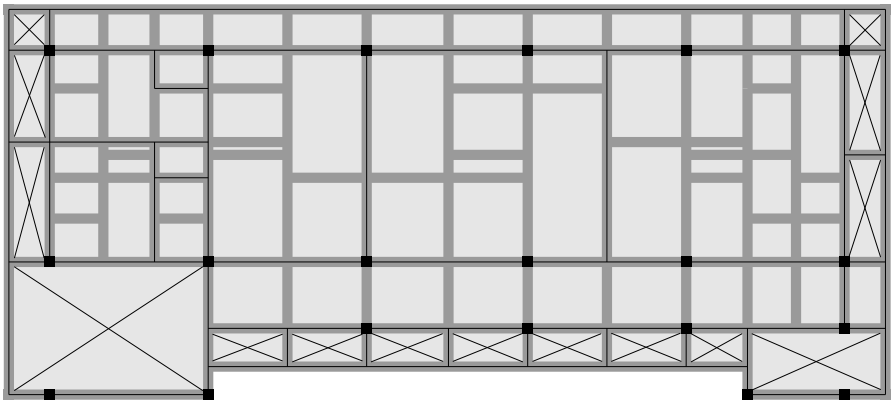


(c)

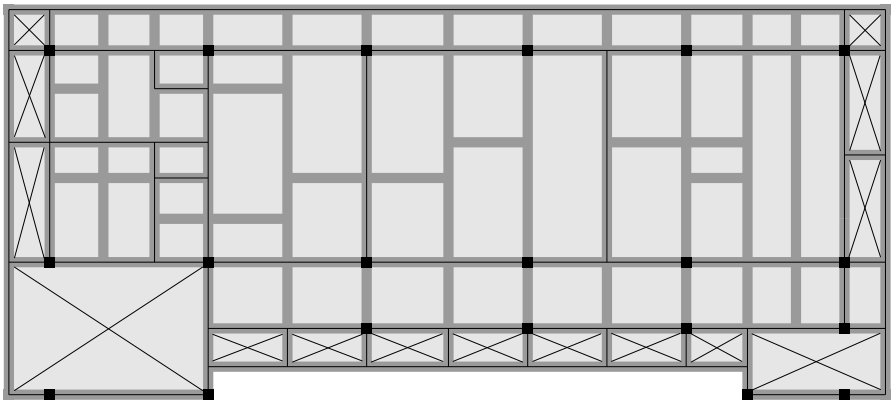
Fig. 5.25. Problem 8: (a) The simplified architectural floor plan. (b) The grid. (c) The best solution.



Gen-0

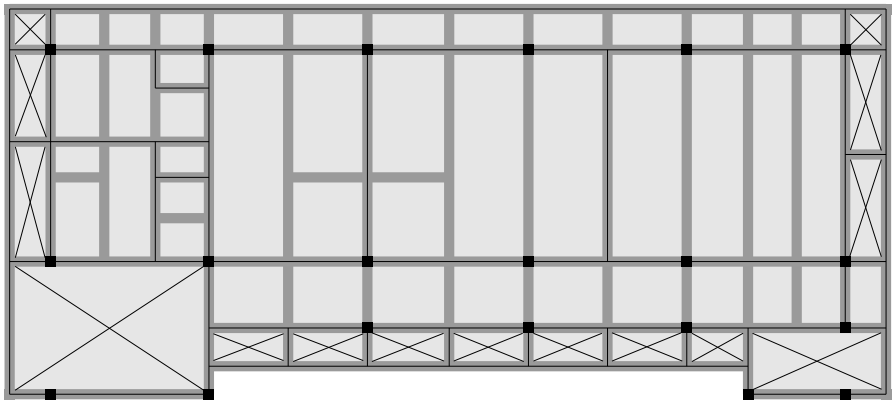


Gen-100

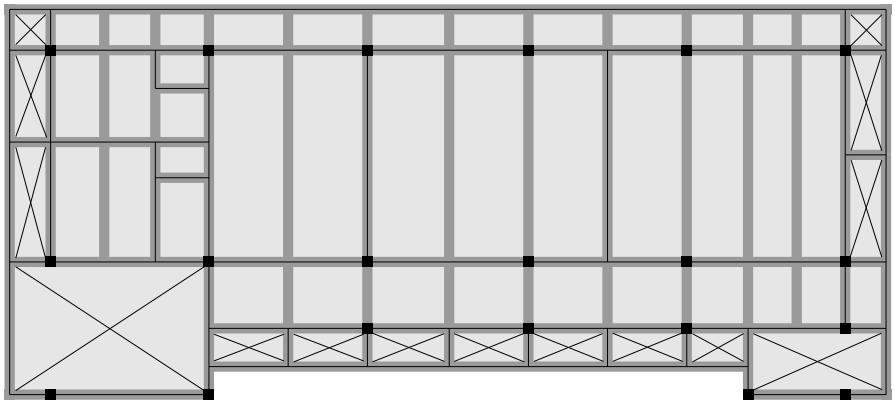


Gen-200

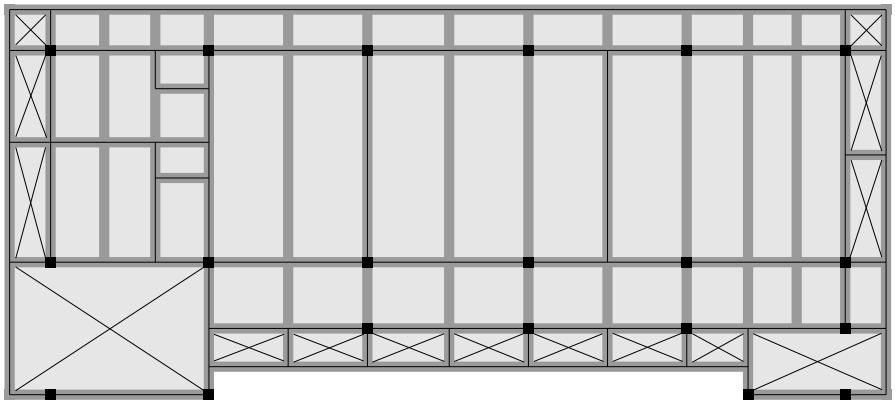
Fig. 5.26. Problem 8: a typical evolution of solutions.



Gen-300



Gen-425-Best



Gen-2000

Fig. 5.26. (*continued*)

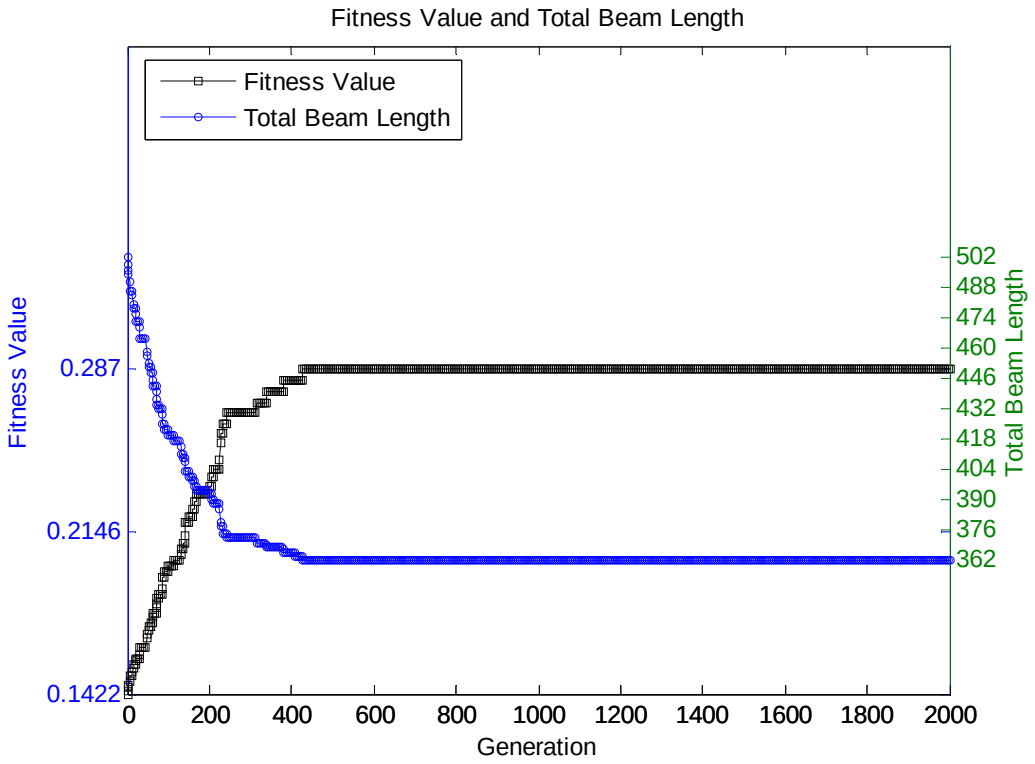
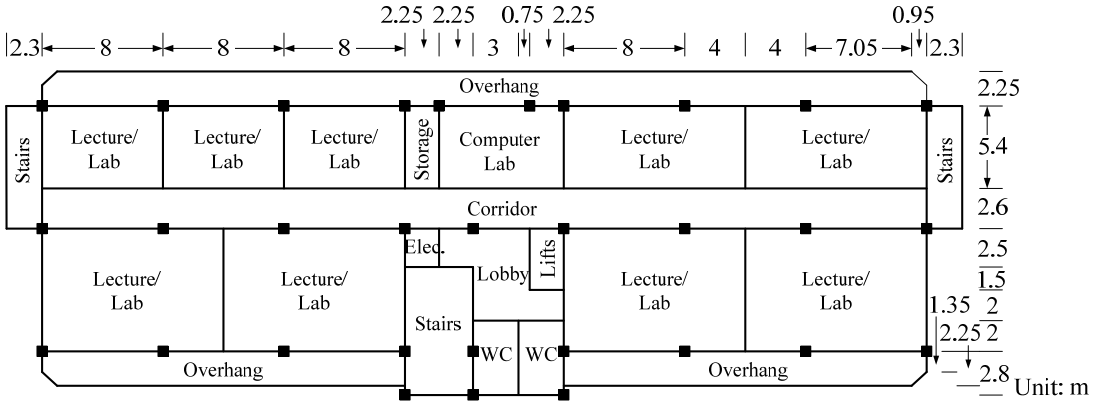


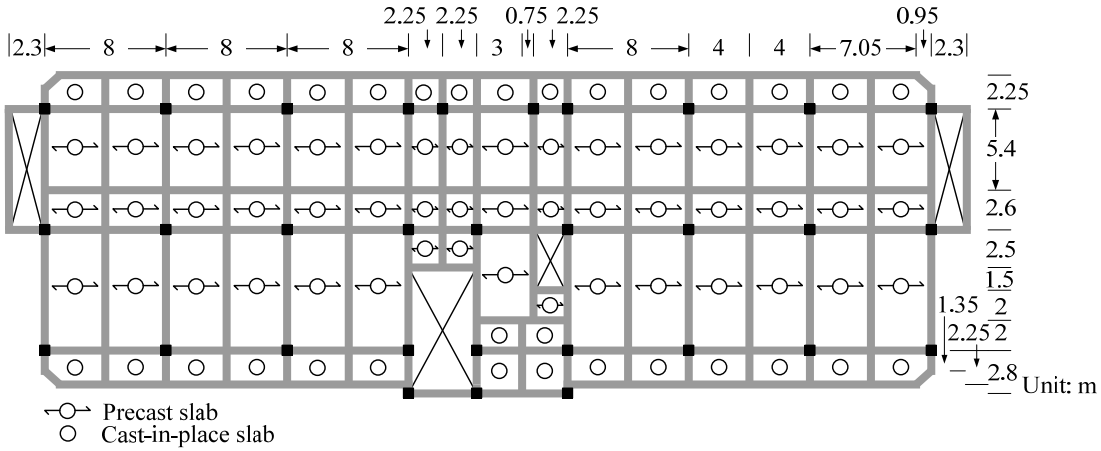
Fig. 5.27. Problem 8: a typical development of the fitness and the total beam length.

5.9 Problem 9

The last problem is the same floor plan as that of the fifth problem. The floor plan is slightly adjusted from the actual to a rectilinear one in contrast to a simplified rectangular floor plan adopted in the fifth problem. In fact, only the angles of the four floor corners are changed to be right angles. Fig. 5.28a shows the architectural floor plan of the building and Fig. 5.28b shows the real structural beam-slab layout. The simplified floor plan is shown in Fig. 5.29a. Similar to the fifth problem, the maximum allowable slab length in the *x* direction is preset to 4 m. In addition, the maximum allowable slab length in the *y* direction is preset to 8 m. In this problem, a non-uniform grid is used. The maximum grid spacing is set to 4 m in order that the slab constraint can be satisfied in both directions. The obtained non-uniform grid is shown in Fig. 5.29b. With the outer-boundary beams placed in advance, this grid results in 272 design variables. For this problem, four calculation sets for four different population sizes of 20, 40, 60 and 80 individuals are analyzed. Each calculation set consists of 100 runs.



(a)



(b)

Fig. 5.28. Problem 9: (a) The real architectural floor plan. (b) The real structural floor plan.

Table 5.10 shows the statistics of the obtained results. The best solutions of all calculation sets are found to be the same. The best layout from the algorithm is shown in Fig. 5.29c. This layout is found to be in good agreement with the real structural layout shown in Fig. 5.28b. In fact, if the simplified parts are disregarded, the two layouts are exactly the same. The SD values in Table 5.10 for all calculation sets are very small. The coefficients of variation of all calculation sets are found to be the same and equal to 0.01. For this problem, the algorithm requires, on average, less than 1,380 generations to obtain the convergence. Fig. 5.30 shows a typical evolution of layouts from a run with 20 individuals. Fig. 5.31shows the convergences of the fitness and the total beam length of the same run shown in Fig. 5.30.

Table 5.10. Problem 9: statistics of the results.

	Calculation set of 100 runs			
	Population Size			
	20	40	60	80
Maximum fitness	0.310	0.310	0.310	0.310
Average fitness	0.307	0.308	0.308	0.307
Minimum fitness	0.300	0.294	0.298	0.296
SD of fitness	0.003	0.003	0.003	0.003
Appearance percentage of the best solution of the calculation set (%)	40	45	42	38
Average required number of generations for the solution convergence	1147.5	1116.0	1222.1	1371.9

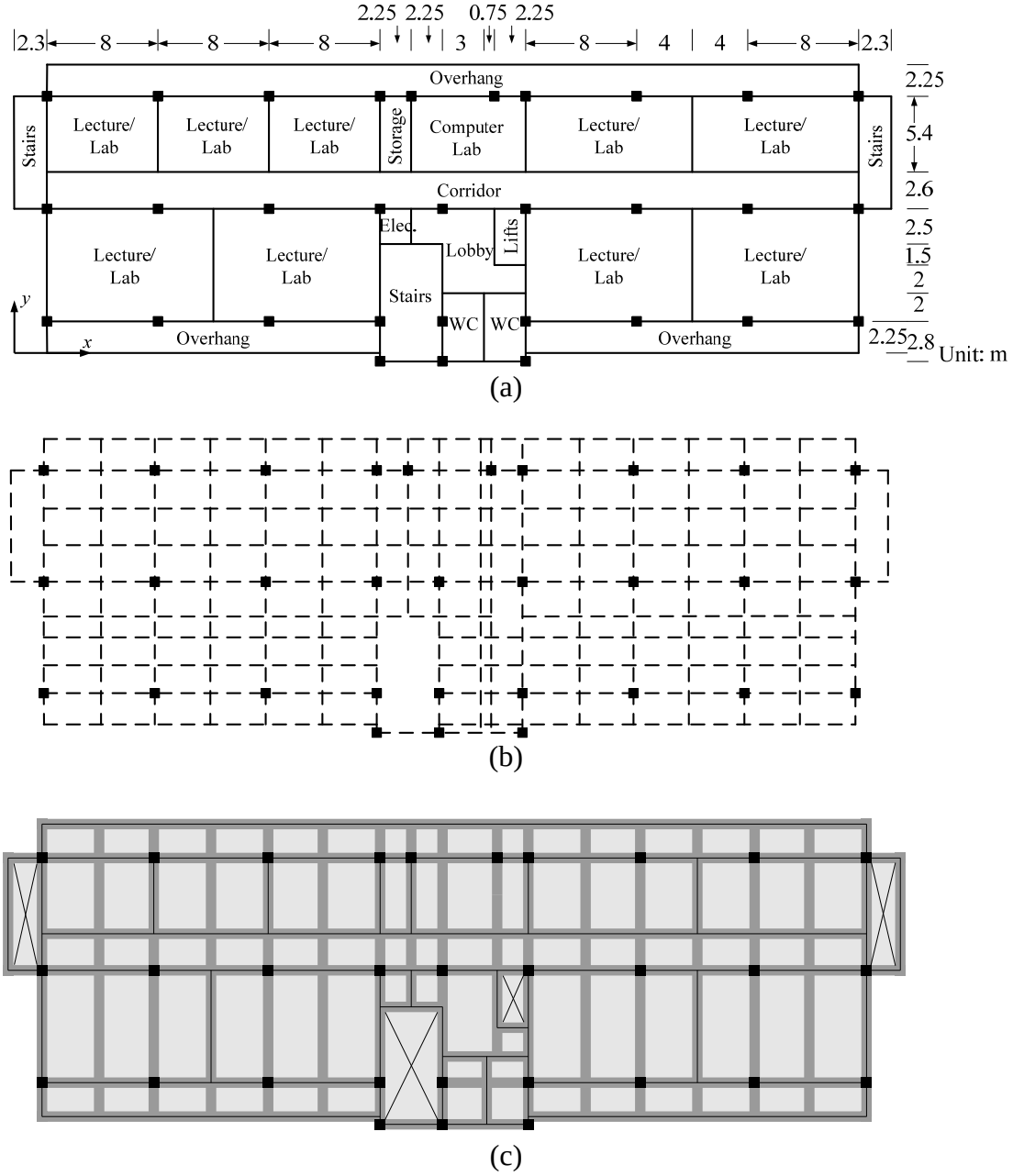
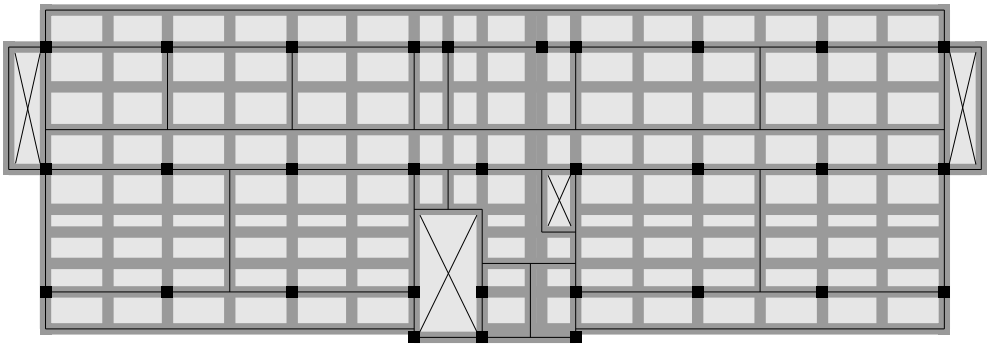
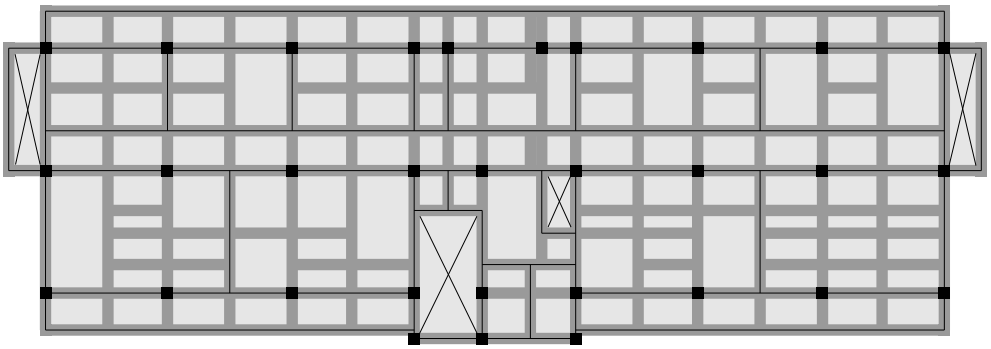


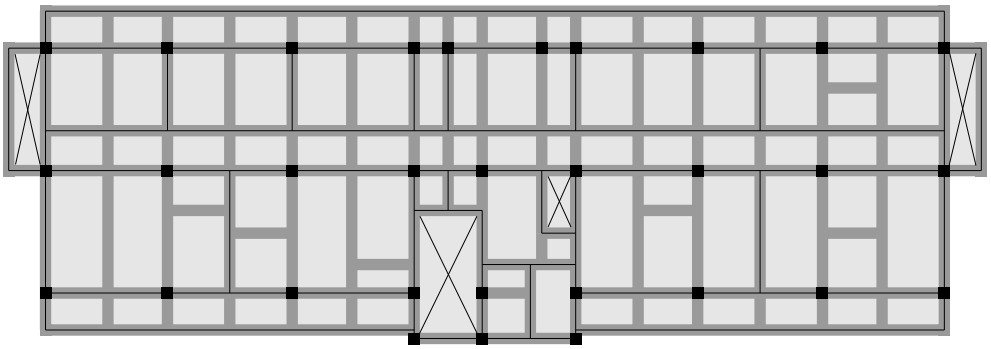
Fig. 5.29. Problem 9: (a) The simplified architectural floor plan. (b) The grid. (c) The best solution.



Gen-0

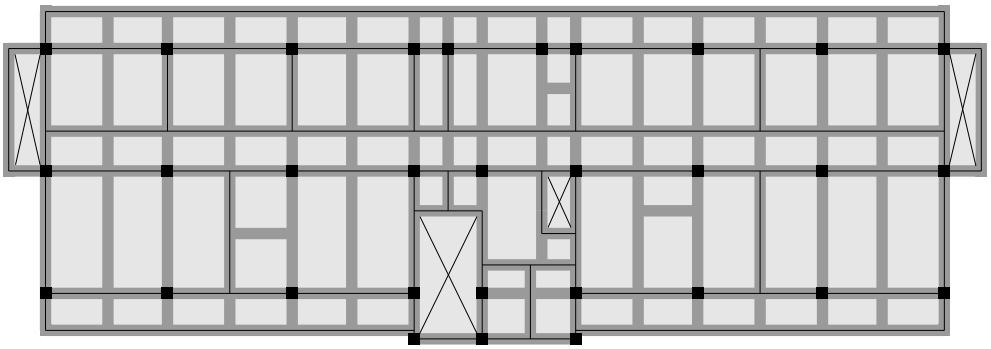


Gen-100

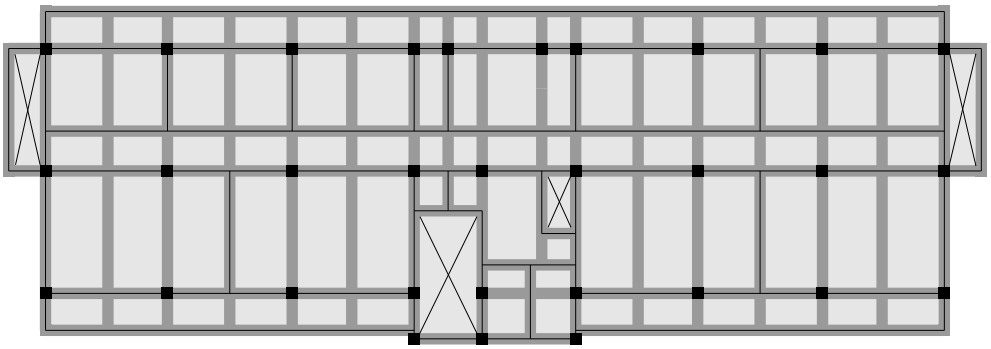


Gen-500

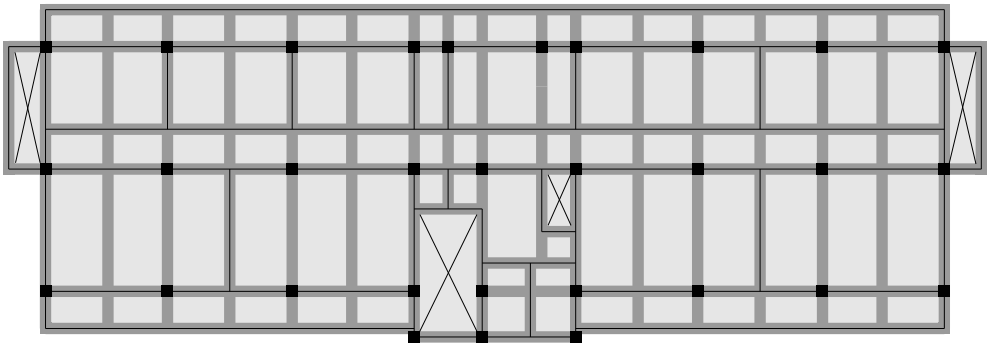
Fig. 5.30. Problem 9: a typical evolution of solutions.



Gen-700



Gen-926-Best



Gen-2000

Fig. 5.30. (*continued*)

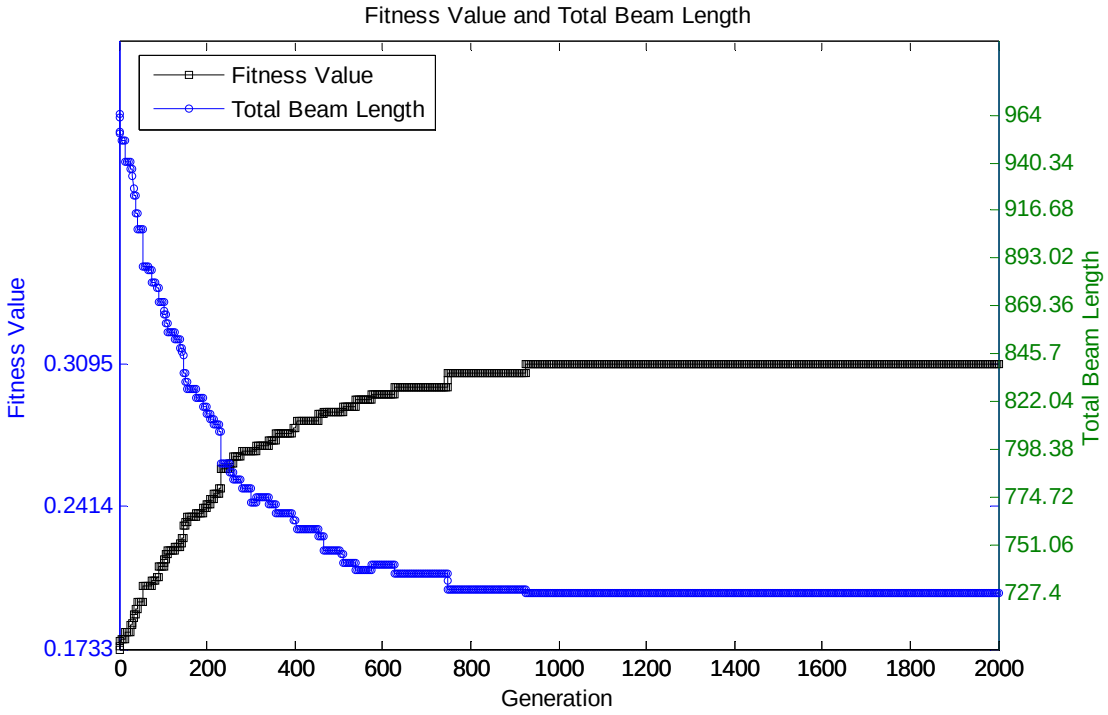


Fig. 5.31. Problem 9: a typical development of the fitness and the total beam length.

Chapter 6

Conclusions

In this study, a new genetic algorithm for beam-slab layout design of rectangular and rectilinear floors is successfully proposed. The input of the proposed algorithm is an architectural floor plan with given positions of columns and walls. Beam segments attached to line segments of a grid that is superimposed on the floor plan are used as the design variables. By using a newly proposed coding scheme for beam-slab layouts, any pattern of beam segments can always be interpreted as a geometrically valid beam-slab layout. In this study, the beam-slab layout design problem is written as an optimization problem by using an objective function that is written based on how well slabs are supported by columns. In addition, constraints based on positions of walls, the maximum slab dimensions as well as the total floor area are developed. The GA used in this study is derived from the simple GA by adding adaptive penalty and elitism processes.

From the example problems, it can be seen that the proposed GA successfully finds good layouts of beams and slabs for the given floor plans. The obtained beam-slab layouts in the example problems are found to be practical layouts that can really be used in the next structural design step. From Problems 1 to 4, the rectangular floor plans with the 0.5-m fine grids are used. The results show that the proposed GA can find good results even though the grids are fine. Problems 6 to 9 are rectilinear floors. Problem 6 is tested with the 0.5-m fine grid. The obtained results are also good. The floor plans of Problems 7 to 9 are simplified from some real existing floor plans. These Problems are solved by using the coarser grids. The obtained results agree quite well with the real structural floor plans. Although it may be argued that the beam-slab layouts of all example problems can be designed without much difficulty by humans, this study intends to demonstrate that this particular design task, which is highly heuristic, can be performed acceptably by computers. Finally, it can be concluded that the proposed GA, together with the proposed layout coding technique, can efficiently help automate design of beam-slab layouts.

In this study, all positions of columns are prescribed as part of the input data. The future study of this research may include those problems where the positions of columns are unknown and, as a result, become part of the design variables. Moreover, different search algorithms such as the particle swarm optimization may be tried in order to possibly increase the efficiency of the computation.

References

Adeli, H. and Cheng, N.-T., 1994. Integrated genetic algorithm for optimization of space structures. *Journal of Aerospace Engineering*. 6: 315-328.

Arora, J. S., 2004. *Introduction to optimum design*. Amsterdam. Academic Press.

Bailey, S. F. and Smith, I. F. C., 1994. Case-based preliminary building design. *Journal of Computing in Civil Engineering*. 8: 455-468.

Balachandran, B., 1993. *Knowledge-based optimum design*. Southampton. Computational Mechanics Publications.

Bausys, R. and Pankrasovaite, I., 2005. Optimization of architectural layout by the improved genetic algorithm. *Journal of Civil Engineering and Management*. 11: 13-21.

Bradshaw, J. and Miles, J. C., 1997. Using standard fitnesses with genetic algorithms. *Advances in Engineering Software*. 28: 425-435.

Burns, S., A., 2002. *Recent advances in optimal structural design*. Reston. American Society of Civil Engineers.

Camp, C., Pezeshk, S. and Cao, G., 1998. Optimized design of two-dimensional structures using a genetic algorithm. *Journal of Structural Engineering*. 124: 551-559.

Camp, C. V., Pezeshk, S. and Hansson, H., 2003. Flexural design of reinforced concrete frames using a genetic algorithm. *Journal of Structural Engineering*. 129: 105-115.

Coello, C. C., Hernandez, F. S. and Farrera, F. A., 1997. Optimal design of reinforced concrete beams using genetic algorithms. *Expert Systems with Applications*. 12: 101-108.

Deb, K., 1995. *Optimization for engineering design: Algorithms and examples*. New Delhi. Prentice-Hall.

Fenves, S. J., Rivard, H. and Gomez, N., 2000. SEED-config: A tool for conceptual structural design in a collaborative building design environment. *Artificial Intelligence in Engineering*. 14: 233-247.

Fenves, S. J., Rivard, H., Gomez, N. and Chiou, S.-C., 1995. Conceptual structural design in SEED. *Journal of Architectural Engineering*. 1: 179-186.

Foley, C. M. and Schinler, D., 2003. Automated design of steel frames using advanced analysis and object-oriented evolutionary computation. *Journal of Structural Engineering*. 129: 648-660.

Fuyama, H., Law, K. H. and Krawinkler, H., 1997. An interactive computer assisted system for conceptual structural design of steel buildings. *Computers & Structures*. 63: 647-662.

Galante, M., 1996. Genetic algorithms as an approach to optimize real-world trusses. *International Journal for Numerical Methods in Engineering*. 39: 361-382.

Gero, J. S. and Kazakov, V. A., 1998. Evolving design genes in space layout planning problems. *Artificial Intelligence in Engineering*. 12: 163-176.

Ghasemi, M. R. and Hinton, E., 1996. Truss optimization using genetic algorithms. *Advaced in Computational Structures Technology*. Edinburgh. Civil-Comp Press: 59-75.

Ghasemi, M. R., Hinton, E. and Wood, R., D., 1999. Optimization of trusses using genetic algorithms for discrete and continuous variables. *Engineering Computations*. 16: 272-301.

Goldberg, D. E., 1989. *Genetic algorithms in search, optimization, and machine learning*. Massachusetts. Addison-Wesley.

Grierson, D. E., 1996. Automated conceptual design of structural systems. *Advances in Computation Structures Technology*. Edinburgh. Civil-Comp Press: 157-161.

Grierson, D. E. and Khajehpour, S., 2002. Method for conceptual design applied to office buildings. *Journal of Computing in Civil Engineering*. 16: 83-103.

Grierson, D. E. and Park, K., -W., 1996. Optimal conceptual topological design. *Proceedings of the first ASCE U.S.-Japan joint seminar on structural optimization.* Chicago. American Society of Civil Engineers: 91-96.

Griffiths, D. R. and Miles, J. C., 2003. Determining the optimal cross-section of beams. *Advanced Engineering Informatics*. 17: 59-76.

Jo, J. H. and Gero, J. S., 1998. Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering*. 12: 149-162.

Kameshki, E. S. and Saka, M. P., 2001. Optimum design of nonlinear steel frames with semi-rigid connections using a genetic algorithm. *Computers and Structures*. 79: 1593-1604.

Kicinger, R., Arciszewski, T. and DeJong, K., 2005. Evolutionary design of steel structures in tall buildings. *Journal of Computing in Civil Engineering*. 19: 223-238.

Krishnamoorthy, C. S., Venkatesh, P. P. and Sudarshan, R., 2002. Object-oriented framework for genetic algorithms with application to space truss optimization. *Journal of Computing in Civil Engineering*. 16: 66-75.

Kumar, B. and Raphael, B., 1997. CADREM: A case-based system for conceptual structural design. *Engineering with Computers (Historical Archive)*. 13: 153-164.

Lin, T. Y. and Stotesbury, S. D., 1981. *Structural concepts and systems for architects and engineers*. New York. John Wiley & Sons.

Maher, M. L., 1984. *HI-RISE: A knowledge-based expert system for the preliminary structural design of high rise buildings*. Ph.D. Thesis, Department of Civil Engineering. Carnegie-Mellon University:

Maher, M. L. and Balachandran, B., 1994. Multimedia approach to case-based structural design. *Journal of Computing in Civil Engineering*. 3: 359-376.

Mathews, J. D. and Rafiq, M. Y., 1995. Adaptive search to assist in the conceptual design of concrete buildings. *Developments in Neural Networks and Evolutionary Computing for Civil and Structural Engineering.* Edinburgh. Civil-Comp Press: 179-187.

Michalek, J. J., Choudhary, R. and Papalambros, P. Y., 2002. Architectural layout design optimization. *Engineering Optimization*. 34: 461-484.

Michalek, J. J. and Papalambros, P. Y., 2002. Interactive design optimization of architectural layouts. *Engineering Optimization*. 34: 485-501.

Miles, J. C., Sisk, G. M. and Moore, C. J., 2001. The conceptual design of commercial buildings using a genetic algorithm. *Computers & Structures*. 79: 1583-1592.

Nanakorn, P. and Meesomklin, K., 2001. An adaptive penalty function in genetic algorithms for structural design optimization. *Computers & Structures*. 79: 2527-2539.

Nimityongskul, N., 2004. *An ant colony optimization algorithm for sizing optimization of structures*. Civil engineering. Sirindhorn International Institute of Technology, Thammasat University:

Park, K.-W. and Grierson, D. E., 1999. Pareto-optimal conceptual design of the structural layout of buildings using a multicriteria genetic algorithm. *Computer-Aided Civil and Infrastructure Engineering*. 14: 163-170.

Rafiq, M. Y., Mathews, J. D. and Bullock, G. N., 2003. Conceptual building design---evolutionary approach. *Journal of Computing in Civil Engineering*. 17: 150-158.

Rajan, S. D., 1995. Sizing, shape, and topology design optimization of trusses using genetic algorithm. *Journal of Structural Engineering*. 121: 1480-1487.

Rajeev, S. and Krishnamoorthy, C. S., 1992. Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering*. 118: 1233-1250.

Rajeev, S. and Krishnamoorthy, C. S., 1997. Genetic algorithms-based methodologies for design optimization of trusses. *Journal of Structural Engineering*. 123: 350-358.

Sabouni, A. R. and Al-Mourad, O. M., 1997. Quantitative knowledge based approach for preliminary design of tall buildings. *Artificial Intelligence in Engineering*. 11: 143-154.

Sacks, R. and Warszawski, A., 1997. A project model for an automated building system: Design and planning phases. *Automation in Construction*. 7: 21-34.

Sacks, R., Warszawski, A. and Kirsch, U., 2000. Structural design in an automated building system. *Automation in Construction*. 10: 181-197.

Schodek, D. L., 2001. *Structures*. New Jersey. Prentice-Hall.

Shaw, D., Miles, J. and Gray, A., 2008. Determining the structural layout of orthogonal framed buildings. *Computers & Structures*. 86: 1856-1864.

Sisk, G. M., Miles, J. C. and Moore, C. J., 2003. Designer centered development of GA-based DSS for conceptual design of buildings. *Journal of Computing in Civil Engineering*. 17: 159-166.

Soh, C.-K. and Yang, J., 1998. Optimal layout of bridge trusses by genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*. 13: 247-254.

Sriram, D., 1987. *Knowledge-based approaches for structural design*. Topics in engineering. Southampton. Computational Mechanics Publications.

Syrmakezis, C. A., Mikroudis, C. A. and Rouva, S., 1996. Development of the VK.Expert system for computer-aided preliminary design of reinforced concrete buildings. *Information Processing in Civil and Structural Engineering Design.* Civil-Comp Press: 45-51.

Syrmakezis, C. A. and Mikroudis, G. K., 1997. ERDES--an expert system for the aseismic design of buildings. *Computers & Structures*. 63: 669-684.

Tsakalias, G. E., 1994. KTISMA: A blackboard system for structural model synthesis of asymmetrical skeletal reinforced concrete buildings. *Artificial Intelligence and Object Oriented Approaches for Structural Engineering.* Edinbugh. Civil-Comp Press: 15-21.