Chapter 4

Basic Constraint Handling

With the definition of the supercube representation in the previous chapter, everything is now available to start optimising building spatial designs. However, it was also identified that numerous infeasible designs exist in this representation. To navigate the infeasible space, this chapter aims to evaluate constraint handling techniques to use during optimisation. This also ties in to answering RQ2, which asks for methods to effectively handle constraints, specifically to ensure feasible building spatial designs are discovered.

In order to improve the understanding of the supercube representation, the search space, and the objectives, this chapter approaches building spatial design as a single-objective problem. Based on the results it will be easier to extend to multi-objective building spatial design in the next chapter (Chapter 5). To this end an evolution strategy [82, 90] is used, and extended here with constraint handling mechanisms suitable to the considered problem.

This chapter continues in Section 4.1 with a discussion of the problem of building design, as well as a brief overview of work related to building design optimisation. Section 4.2 introduces evolution strategies, which will be used in the optimisation procedure. Then, in Section 4.3 the considered objective functions are discussed in more detail. In Section 4.4 the integration of evolution strategies with constraint handling techniques are discussed. The setup of experiments used to evaluate the approach are then described in Section 4.5. An in-depth discussion of results is then presented in Section 4.6. Finally, Section 4.7 provides a summary of the main results, and indicates directions for future work.

4.1 Building Design Optimisation

Building design is traditionally performed by architects and engineers who create solutions for discipline specific design problems. Nowadays these solutions are usually assessed by, and modified in accordance with, design analysis tools. Such tools include finite element methods (FEM) to simulate for instance structural performance or heat transfers, and computational fluid dynamics (CFD) to simulate e.g. heat, ventilation, and lighting problems. The division between the different disciplines within the field of building design also calls for tools that allow engineers from different disciplines to cooperate. An example of such a tool is computer aided design (CAD), which is used to create and share designs. However, currently building information modelling (BIM) [37] is on the rise. BIM is a method that uses data management in order to dynamically share information with other disciplines. This allows engineers to – among other things – take other disciplines into account in the early (also, conceptual) design phase. The early design phase is important for optimal building designs, because decisions in the conceptual design stage often affect performances across all disciplines. A design based on a single discipline may therefore lead to a suboptimal multi-disciplinary design.

Optimisation in the built environment is mostly performed by parametrising building components, e.g. installation type, construction type, material type, dimensions, or shapes. In [79] an overview of software tools for building optimisation is presented, followed by the introduction of a new tool. The new tool allows design variables of a building design to be selected for optimisation. Following that, an optimisation strategy can be selected. Although such tools can change and greatly improve a design, they cannot discover new designs (e.g. a new window cannot appear). Very recently, advances in early design optimisation have been made. For example, in [52] an optimisation approach inspired by the human design process is used to optimise a building spatial design for the structural performance of its related structural design. In the building physics discipline, the software tools discussed in [3, 102] are able to provide performance information for building designs. Statistical sensitivity analysis to predict the impact of design variables on the optimality of a building design is presented in [54]. This analysis is interesting for early design optimisation as the impact of each design variable in distinct design stages can be investigated.

In this chapter building optimisation for early stage building spatial design is performed using the previously defined supercube representation (Section 3.2). Although the supercube does not allow for completely free exploration of designs, it does permit significant changes in the shape of the building. Completely free exploration will be investigated in Chapter 9, where the supercube is used in conjunction with a corresponding superstructure free representation. Here, the supercube considers a layout of building spaces that can be rearranged and resized for optimal performance. Optimisation methods are investigated for two different objective functions: (a) Structural performance, for which the compliance is to be minimised, and (b) building physics, for which the outside surface area is to be minimised. These disciplines are selected because they are known to be dependent on the building spatial design. Later in this thesis (Chapter 6) a resistance/capacitance (RC) network will be employed to analyse heating and cooling energy, it will serve as a more accurate measure of building physics performance. For the development of the optimisation method presented here minimal surface area is used as objective function because it is cheaper to compute. Details on the objective functions follow in Section 4.3.

4.2 Evolution Strategies

As outlined in the introduction evolutionary algorithms (EAs, Section 2.3) subsume different algorithms that mimic natural evolution, in order to find improved or optimised technological designs [4]. Population-based evolutionary algorithms generally work according to a basic loop structure, the so-called generational loop. It starts after an initialisation phase where an initial parent population consisting of μ individuals (solution candidates) is generated and evaluated. Then the loop begins by establishing a ranking among the individuals according to their fitness (their performance according to some objective function). Next, parents are selected to generate an offspring population (also referred to as reproduction). In this step the ranking of the population might be taken into account, although in Evolution Strategies – an important EA variant – parent individuals are chosen randomly. From the selected parent individuals, λ offspring individuals are created. Recombination is applied to allow parts of the genomes (the decision variables, possibly encoded) from multiple parents to be combined into a new genome. In order to introduce new – possibly not previously considered – information into the genome, random perturbations are applied through mutation of some of the variables in the newly produced genome. When applicable, this is followed by constraint evaluation, where invalid individuals may either be repaired, penalised, or discarded. Finally, the offspring population is evaluated on the objective function, a new parent population is produced (note that here performance may be taken into account, unlike in the reproduction step), and the loop starts anew.

The specific type of evolutionary algorithm used in this chapter is the $(\mu+\lambda)$-Evolution Strategy. Evolution Strategies (ESs) were developed by Ingo Rechenberg and Hans-Paul Schwefel at the Technische Universität Berlin in the 1960s and are especially well suited for solving engineering design problems [82, 90]. They are interesting for this work because they are able to handle discrete as well as continuous decision variables, as outlined in [69].

In Algorithm 1 the main loop of a $(\mu+\lambda)$-ES is summarised. In short, the parent population (multiset) X_t, indexed by the number of iterations and consisting of μ individuals, is used as a template to generate the offspring population X'_t of size λ. Then, from the combination of parents and offspring the best individuals are selected as the parents of the next generation X_{t+1}. The initialisation, mutation, and recombination operators are chosen in a domain specific way, as will be discussed later in this section. For a more detailed discussion on evolution strategies and their properties the reader is referred to [4] and [12].

Algorithm 1 $(\mu+\lambda)$ Evolution Strategy [90]

```
1: $t \leftarrow 0$
2: $X_t \leftarrow$ init()                    ▷ $X_t \in \mathcal{S}^\mu$ : Set of individuals
3: while $t < t_{max}$ do    ▷ Generate $\lambda$ solutions by (stochastic) variation operators
4:     $X'_t \leftarrow$ generate($X_t$)
5:     evaluate($Q_t$)
6:     $X_{t+1} \leftarrow$ select($X'_t \cup X_t$)            ▷ Rank and select $\mu$ best
7:     $t \leftarrow t + 1$
8: end while
```

4.3 Objective Functions

Structural performance is optimised here by minimising the compliance. To compute the compliance corresponding to a building spatial design, a building structural design needs to be provided. This is carried out by applying a so-called structural grammar on each space of the building spatial design. The grammar that is used here adds four walls (slabs), with a roof (also a slab) on top. All of these are made of concrete with a thickness $t = 150\,\text{mm}$ (millimetre), Young's modulus $E = 30\,000\,\text{N}\,\text{mm}^{-2}$ (newton per square millimetre), and a Poisson's ratio $\nu = 0.3$. The building spatial design is then loaded with a live load of $1.8\,\text{kN}\,\text{m}^{-2}$ (kilo newton per square metre) on each floor/roof surface. Further, each outside surface is subjected to wind loads of $1.0\,\mathrm{kN\,m^{-2}}$ for pressure, $0.5\,\mathrm{kN\,m^{-2}}$ for suction, and $0.4\,\mathrm{kN\,m^{-2}}$ for shear from eight general directions (North, Northwest, West, etc.). After the transfer of the loads to the building structural design and meshing of the structure, finite element analyses are carried out to find the total compliance for all loads together. More detailed information about this procedure can be found in [52]. In summary, the first optimisation task is to minimise the total compliance (measured in newton metres), subject to the given constraints.

Next, building physics performance is optimised by minimising the total outside surface area. This objective can be computed for the supercube representation (Section 3.2) as follows. Firstly, it is required that the building spatial design contains no cantilevers, overhangs, or archways. This is achieved by Constraint C2, which ensures that no vertical gaps exist with Equation 3.2. Additionally, the computation requires a layer of cells with their binary variables equal to zero around the supercube (Equation 3.4).

To compute the outside surface area the surfaces of all outer walls, and the roof are considered. These may be found by considering all rows, columns, and pillars of the supercube. In width and depth the number of changes from zero to one are counted, and then multiplied by the area corresponding to the considered dimensioning indices. Finally, to consider both the entry and exit points, the outcome is multiplied by two. In case of the height direction the multiplication by two is omitted, because the connection with the ground layer is not counted as outside surface area. Since there are no vertical gaps in feasible designs, the height direction is essentially the sum of areas of all pillars with an active cell. The sum $S_A = S_w + S_d + S_h$ of Equations 4.1, 4.2, and 4.3 below is then the total outside surface area. Here S_w, S_d, and S_h are the total outside surface area of the width vectors (rows), depth vectors (columns), and height vectors (pillars), respectively. Note that once more, $b_{i,j,k}$ is taken as the result of a logical OR over all ℓ bits of a cell i, j, k. In summary, the second objective function is to minimise the outside surface area S_A (measured in square metres), subject to the given constraints.

$$S_w = \sum_{j=1}^{N_d} \sum_{k=1}^{N_h} \left(2 \left(\sum_{i=1}^{N_w+1} (1 - b_{i-1,j,k})\, b_{i,j,k} \right) d_j h_k \right) \tag{4.1}$$

$$S_d = \sum_{i=1}^{N_w} \sum_{k=1}^{N_h} \left(2 \left(\sum_{j=1}^{N_d+1} (1 - b_{i,j-1,k})\, b_{i,j,k} \right) w_i h_k \right) \tag{4.2}$$

$$S_h = \sum_{i=1}^{N_w} \sum_{j=1}^{N_d} \left(\left(\sum_{k=1}^{N_h+1} (1 - b_{i,j,k-1})\, b_{i,j,k} \right) w_i d_j \right) \tag{4.3}$$

4.4 Methods

This section provides a description of how the earlier introduced $(\mu + \lambda)$-ES is customised for building spatial design optimisation with the supercube representation, and how the different constraints are handled. To this end, first a general overview is given of the procedure that will be the subject of the later presented experiments. The outline is visualised in Figure 4.1.

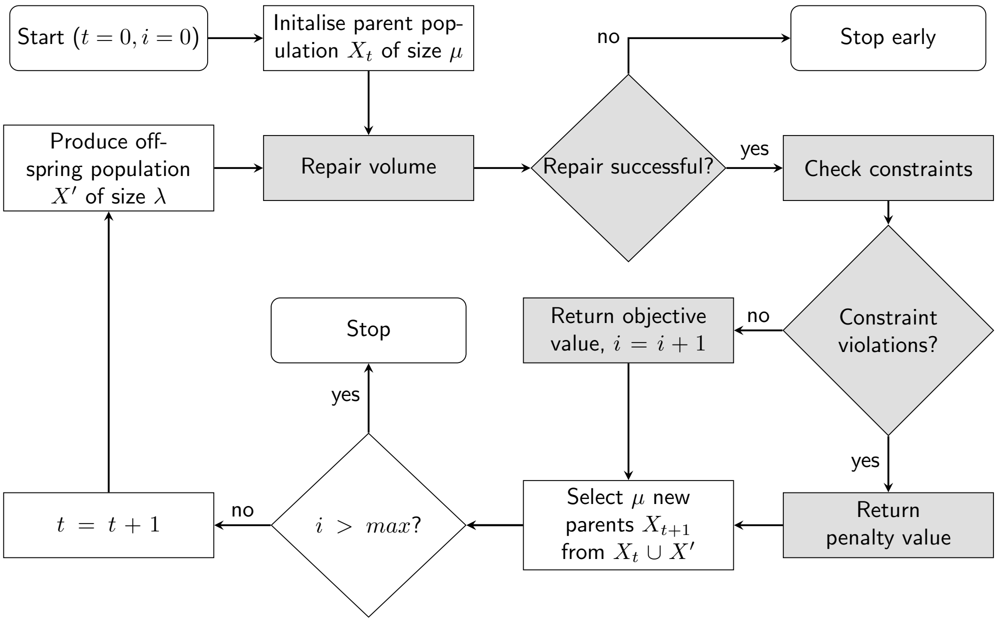

Start ($t = 0, i = 0$)
Initalise parent population X_t of size μ
no
Stop early
Produce offspring population X' of size λ
Repair volume
Repair successful?
yes
Check constraints
Stop
Return objective value, $i = i + 1$
no
Constraint violations?
yes
yes
$t = t + 1$
no
$i > max$?
Select μ new parents X_{t+1} from $X_t \cup X'$
Return penalty value

Figure 4.1: Optimisation outline. Nodes shaded in grey are performed for each individual.

The process starts by initialising the parent population X of size μ. Following this, the volume of all new individuals is repaired to be within a small margin of the desired volume V_0. If volume repair fails it is likely that incompatible settings were provided, and the process is stopped early. Next, the constraints are checked, and in case of constraint violations a penalty value is returned. If no constraints are violated, then the objective value is computed and returned instead. Moreover, the evaluation counter i is incremented. Based on these returned objective and penalty values, the

μ best individuals are selected from the parent and offspring populations. In the first iteration this will naturally always be the initial population. If the maximum number of evaluations is reached the process stops here, otherwise the iteration counter t is incremented. Finally, a new generation starts by producing an offspring population X' of size λ. The loop is then repeated by starting again from the volume repair step. This process continues until the desired number of evaluations is reached. Note that here evaluations are counted based on the number of valid (non-constraint violating) solutions.

In the following subsections a number of these processes are discussed in more detail. Specifically, domain specific ES operators (initialisation, selection, mutation, crossover), penalty functions, and repair functions will be introduced.

4.4.1 Domain Specific Operators

The ES starts by generating an initial parent population of $\mu = 20$ individuals. For continuous variables initial values are drawn uniformly at random from $[lb, \ldots, ub]$, with $lb = 3.0$, and $ub = 19.8$ (both in metres) being the lower, and respectively upper bounds for the continuous variables. Step sizes of the continuous variables are simply initialised to $\sigma = 0.1$. While binary variables are initialised to one with a probability $1/N_{cells}$, or zero otherwise. Recall that $N_{cells} = N_w \times N_d \times N_h$.

Parental selection is done by choosing two parents (possibly the same twice) uniformly at random, for each of the $\lambda = 100$ offspring individuals that are generated.

Next, intermediate crossover is applied to the continuous variables as well as their corresponding step sizes. That is, for each variable the arithmetic mean of the two parents is taken. For binary variables dominant crossover is applied by copying the value of the bit from one of the parents, chosen uniformly at random for each bit.

Mutation works as described in Algorithm 2. For convenience two definitions are introduced, the number of continuous variables $N_{cont} = N_w + N_d + N_h$, and the dimensionality of the search space $N_{dims} = N_{cont} + N_{cells} \times N_{spaces}$. Gaussian mutation with individual step sizes is applied to the continuous variables. To this end the number g_3 is drawn from a Gaussian distribution $G(\cdot, \cdot)$. Step sizes σ of the continuous variables are mutated using the Gaussian numbers g_1, g_2 (also drawn from $G(\cdot, \cdot)$), the local learning rate $\tau_1 = 1/\sqrt{2\sqrt{N_{dims}}}$, and the global learning rate $\tau_2 = 1/\sqrt{2 \times N_{dims}}$ as in [69]. Finally, binary variables are mutated by flipping each bit with a probability of $1/(N_{cells} \times N_{spaces})$, for which a number is drawn from the uniform distribution $U(\cdot, \cdot)$.

Algorithm 2 Mutate

```
 1: $\tau_1 \leftarrow 1/\sqrt{2\sqrt{N_{dims}}}$                    ▷ Local learning rate
 2: $\tau_2 \leftarrow 1/\sqrt{2 \times N_{dims}}$                  ▷ Global learning rate
 3: $g_1 \leftarrow G(0,1)$
 4: for all $i \in \{1, \ldots, N_{dims}\}$ do
 5:     $g_2 \leftarrow G(0,1)$
 6:     $g_3 \leftarrow G(0,1)$
 7:     if $i \leq N_{cont}$ then
 8:         $\sigma'_i \leftarrow \sigma_i \times \exp(g_1 \times \tau_2 + g_2 \times \tau_1)$      ▷ Mutate step size
 9:         $x'_i \leftarrow x_i + g_3 \times \sigma'_i$                 ▷ Mutate continuous variable
10:     else
11:         if $U(0,1) < 1/(N_{cells} \times N_{spaces})$ then
12:             $x'_i \leftarrow (x_i + 1) \% 2$                ▷ Mutate binary variable
13:         end if
14:     end if
15: end for
```

Following mutation, some variables may exceed their bounds. To repair them, modified interval bounds treatment is applied as in [69]. Decision variables consider the lower bounds *lb* and the upper bounds *ub*, while step size variables use the bounds $lb_s = 0.01$ and $ub_s = ub \times 0.1$.

Finally, survival selection works as is standard for evolution strategies. Namely, the μ best (lowest objective value) individuals from the $(\mu \cup \lambda)$ (parents and offspring) individuals are selected to be the parent population of the next generation, this is also referred to as elitist selection (the best/elite wins).

4.4.2 Penalties

The supercube representation considers a number of constraint functions that must hold to ensure feasible designs are produced. However, when many infeasible (constraint violating) designs exist (like in the supercube representation, see Section 3.2.4) an optimiser needs some technique to navigate towards feasible space. If no such technique is employed, in the worst case the optimiser might not find any feasible design at all.

Here two approaches are considered, which will be compared empirically in Section 4.5. First, a single fixed penalty value of $pen = 999\,999\,999$ is used whenever any constraint is violated. This value is chosen to be well beyond any realistically expected objective value to prevent any infeasible solution from being favoured over a feasible solution, but is otherwise arbitrary. The penalty provides the optimiser with the information that the considered design is of very low quality. However, if no feasible design is found, this does not help to steer the search closer to feasible space. Even so, this still provides a baseline when comparing with other methods. Second, a graded penalty approach is used where the penalty value depends on the number of constraint violations. The idea behind this approach is that the penalty value should decrease when fewer constraints are violated. By favouring solutions that violate fewer constraints, the search will be biased towards areas closer to feasible space. Given this property, the graded penalty approach should be more suited to navigate the constraint landscape than the single penalty approach. Specifically, an infeasible solution will receive a penalty value equal to $pen + CV - 1$. Here $CV \in \{1, \dots, 5\}$ represents the number of constraint violations, and pen is the same as before. Although the absolute differences between these penalty values are small, together with elitist selection this means the individual with the least constraint violations is always favoured. Note that five constraints are considered here. This concerns the four constraints described in Section 3.2.2, where Constraint C4 is split into two parts: (a) *cuboid shape* (Equations 3.5 and 3.6), and (b) *connected cuboid* (Equation 3.7).

4.4.3 Repair Functions

Given the constraint on the volume introduced by Equation 3.8, a mechanism is needed to maintain a constant volume during optimisation. Since this concerns an equality constraint on continuous variables, the use of penalty values would be less effective than for the previously discussed constraints. That is, finding solutions with exactly the right volume by the stochastic processes of an evolutionary algorithm is so unlikely that another technique is needed to handle this constraint. Here, a repair function is used to change any design that does not satisfy this constraint into one that does.

Repairing the volume works by scaling the continuous variables to satisfy a predefined desired total volume V_0 (in the following experiments $V_0 = 4^3 \times N_{cells}$. How these variables have to be scaled depends on the current total volume V_c, which is simply the outcome of the left-hand side of Equation 3.8. Given the current and desired volume, it is possible to compute the factor $\alpha = V_0/V_c$. The desired volume is then reached by multiplying the continuous variables by the cubic root of α, as shown in Equation 4.4. Note that this is only necessary for vectors (rows, columns, or pillars in the supercube) that contain at least one active cell (cells that belong to a space). Inactive cells do not influence the volume, and therefore can remain unchanged.

$$\forall_i : w'_i = \sqrt[3]{\alpha w_i} \qquad \forall_j : d'_j = \sqrt[3]{\alpha d_j} \qquad \forall_k : h'_k = \sqrt[3]{\alpha h_k} \tag{4.4}$$

Both the creation of new individuals (recombination and mutation), and the volume repair procedure may result in continuous variables that exceed their boundaries. Such boundaries result from the requirement for continuous variables to have positive values (Equation 3.9), but also from limits set for a specific design process. For instance, a lower bound lb may be set because a space that is less than half a metre wide is not practically useful. Likewise, an upper bound ub is used to avoid excessively wide/deep/high spaces. To correct for this, variables exceeding the lower bound are set to the lower bound, while variables exceeding the upper bound are multiplied by 0.95 until they are within the bound. Since these corrections affect the volume, volume repair is done iteratively together with bound corrections until both requirements are satisfied. Note that the volume requirement is considered satisfied as long as the volume remains within 1 % from the desired volume. This prevents an excessive amount of time from being spent solely on satisfying this constraint, while staying reasonably close to the requirement. The iterative process is repeated at most 26 times. If this number of repetitions is insufficient for any of the individuals the optimisation process is stopped and considered unsuccessful. The likelihood that this occurs is dependent on the chosen bounds and the desired volume, but this did not occur during any of the experiments presented in this chapter.

4.5 Experiments

Based on the described optimisation outline a number of experiments have been devised. Single objective optimisation is considered, concerning the surface area and the compliance objectives. By focussing on single-objective optimisation, and – for now – leaving out the additional complications involved in multi-objective optimisation, it is possible to purely focus on developing good constraint handling techniques. This is first done with single penalty values to investigate how well the proposed supercube representation functions in practice, and to set a baseline to compare against. Next, the same objectives are considered, but now with the graded penalty, based on the number of constraint violations. Through the comparison of these two approaches, some first insights are gained into constraint handling for this heavily constrained problem.

These experiments are all conducted with a budget of 1000 evaluations, and repeated five times. As mentioned before, only feasible candidates are evaluated, and thus only feasible candidates reduce the remaining evaluation budget. To prevent the optimisation process from going on forever (in case only infeasible solutions are found), the process is stopped after generating one million candidate solutions, even if the budget is not exhausted yet. Moreover, six different configurations of the supercube are considered to give insight into algorithm behaviour and problem characteristics for different numbers of cells and spaces. The considered configurations are: 2221, 2223, 2225, 3331, 3333, 3335. This notation has to be interpreted as follows. The first three numbers indicate the number of width, depth, and respectively height divisions in the supercube. Whereas the last number indicates how many spaces are considered. The 2221 configuration, for example, indicates a supercube that is two cells wide, deep, and high, and encodes a single space.

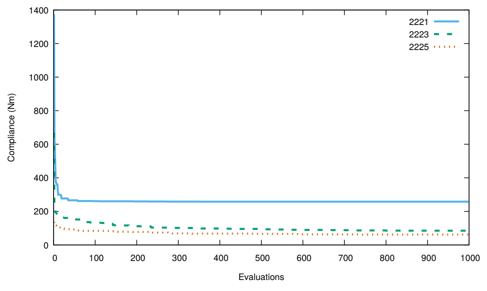
Settings as used in the experiments for the parameters introduced in the previous section are summarised in Table 4.1.

μ	λ	V_0	lb	ub	lb_s	ub_s
20	100	$4^3 \times N_{cells}$	3.0	19.8	0.01	$ub \times 0.1$

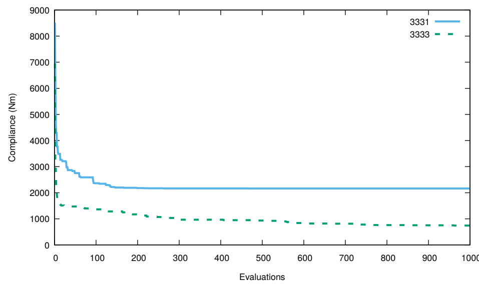
Table 4.1: Parameter settings used for the constraint handling evolution strategy.

4.6 Results

Figure 4.2 shows mean convergence plots for the compliance objective when using the single penalty approach. Note that this mean is only computed over the successful runs, i.e. runs that found 1000 feasible solutions. After a rapid decrease in the objective value during the first few hundred evaluations the optimisation process tends to stagnate. In Figure 4.2a the results for the 222x configurations are shown. Only three of the five runs for configuration 2225 were successful, so the mean of this configuration is based on only three runs. In fact, for these unsuccessful runs no feasible designs were found at all. This means that, for this problem size constraint handling is already a major problem with a single penalty value. With the larger supercube considered for the 333x configurations in Figure 4.2b this problem becomes even more apparent. In case of the 3333 configuration two out of five runs also failed to find any feasible solution, while for the 3335 configuration none of the runs found a feasible design. Evidently, a more sophisticated constraint handling technique is needed.



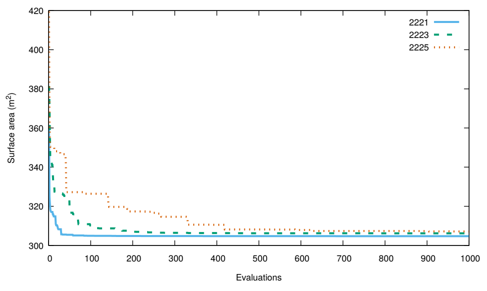
(a) 222x configurations.



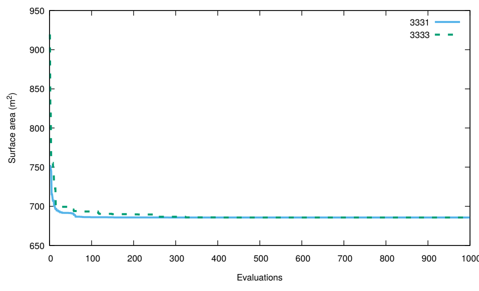
(b) 333x configurations.

Figure 4.2: Mean convergence of the compliance for all successful runs (maximum of five) using a single penalty value. Configurations 2225 and 3333 had 3 successful runs, while 3335 had none.

For the surface area objective similar results can be observed in Figure 4.3. Here too, a number of runs were not successful for various configurations. In Figure 4.3a configuration 2225 completed two of the five runs successfully. Further, the 3333 and 3335 configurations in Figures 4.3b respectively had three and zero successful runs. As was the case for the compliance objective, all unsuccessful runs here did not find any feasible solution at all. When comparing between the two objectives, it appears that the convergence speed is fairly similar.



(a) 222x configurations.



(b) 333x configurations.

Figure 4.3: Mean convergence of the surface area for all successful runs (maximum of five) using a single penalty. Configurations 2225, 3333, and 3335 had 2, 3, and no successful runs respectively.

For the ease of discussion the considered constraints are briefly summarised next.

- *No overlap* (Constraint C1, Equation 3.1), each cell belongs to no more than one space.
- *Ground connected* (Constraint C2, Equation 3.2), all cells are either on the ground level, or have an active cell on the level below them.
- *Existence* (Constraint C3, Equation 3.3), all spaces exist, i.e. they are described by at least one cell.
- *Cuboid shape* (part one of Constraint C4, Equations 3.5 and 3.6), the cells describing a space together form a cuboid shape, possibly with gaps.
- *Connected cuboid* (part two of Constraint C4, Equation 3.7), all cells in the same row, column, or pillar are connected. Given that the cuboid shape constraint also holds, this means that all cells forming a space are connected and form a cuboid without voids.

Note that for some problem configurations certain constraints are always satisfied, these are indicated as not applicable (N/A). For the 2221 and 3331 problem instances, a single space has nothing to overlap with and can therefore never violate Constraint C1. The 222x instances cannot have cuboids with gaps in them, because there are not enough cells for this to occur.

Given these constraints, Table 4.2[1] shows the ratios of constraint violations for every configuration and constraint type for the minimal compliance objective when using a single penalty value. These values are computed by taking the ratio of constraint violations for a single run, and then taking the mean over the five runs. For the small supercube sizes considered here, the existence constraint does not appear to be a major problem. Although for the 3335 configuration it is already violated frequently. The likelihood of violating the no overlap constraint naturally depends on the ratio between the number of spaces and the number of cells. I.e., more spaces per cell increases the likelihood of overlap. This is supported by the results for the 2223 and 2225 configurations. Meanwhile, the probability of constraint violation for the 3335 configuration is extremely high (0.999327413), making it difficult to search

[1]The values shown in the table here differ from those originally reported in [17], because the results there mistakenly showed numbers for a single run, rather than the average over five runs. However, the primary conclusions still hold. The same goes for Tables 4.3, 4.4, and 4.5.

at all. The remaining three constraints show a similar pattern to the no overlap constraint, increasing violation probability with more spaces for both the 222x and 333x configurations, and excessively many constraint violations for the 3335 configuration.

Config.	No overlap	Ground connected	Existence	Cuboid shape	Connected cuboid
2221	N/A	0.365825317	0.058657830	0.444417496	N/A
2223	0.360701373	0.370035639	0.140828056	0.515696088	N/A
2225	0.806357513	0.735303689	0.130767034	0.784227148	N/A
3331	N/A	0.502740860	0.017993546	0.587753423	0.115546192
3333	0.479018371	0.701397235	0.054896992	0.755925962	0.481095119
3335	0.999327413	0.999974401	0.816441071	0.999885802	0.998834023

Table 4.2: Mean constraint violation probability over five runs for minimal compliance optimisation with a single penalty value for various problem configurations.

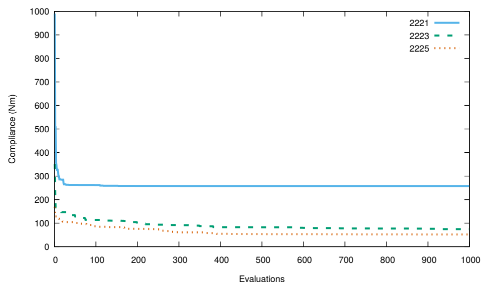
The constraint violations of the surface area in Table 4.3 show largely similar behaviour to those of the compliance objective, with a large portion of the constraints being violated with high probabilities. There are some minor variations between which constraint is violated more or less often between the two objectives. However, this can likely be attributed to chance, considering the small number of runs. Moreover, constraint violations in both cases frequently occur more than 50 % of the time. This further supports the findings from the convergence analysis, that the single penalty approach is unable to handle this heavily constrained problem.

Config.	No overlap	Ground connected	Existence	Cuboid shape	Connected cuboid
2221	N/A	0.283941972	0.058233004	0.460658365	N/A
2223	0.363531813	0.363550249	0.135668651	0.510627974	N/A
2225	0.865383477	0.817330126	0.089653027	0.850689870	N/A
3331	N/A	0.486303466	0.046760850	0.524354276	0.137718976
3333	0.532956461	0.744087185	0.049183401	0.795422495	0.500961661
3335	0.999399612	0.999975601	0.816310074	0.999910402	0.998905822

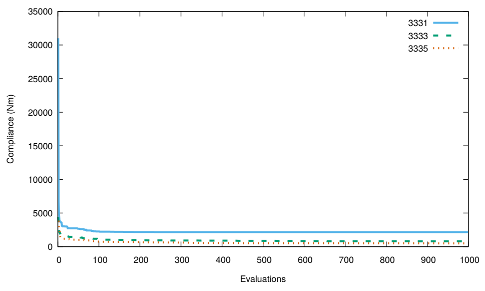
Table 4.3: Mean constraint violation probability over five runs for surface area optimisation with a single penalty value for various problem configurations.

Next, it is investigated whether a graded penalty approach is able to remedy this problem. The used approach penalises based on the number of constraints that are violated. This should allow evolutionary search to gradually correct violations, and move towards feasible solutions faster.

Figure 4.4a shows the mean convergence of the compliance on the 222x configurations when using the graded penalty method. Notably, all runs were completed successfully, which shows the advantage of this method over the single penalty approach. The convergence behaviour is largely similar to the single penalty method, with quick improvements early on and stabilisation after a few hundred evaluations. For the 333x configurations shown in Figure 4.4b the results are also largely the same. Despite the fact that these configurations are more challenging than their 222x counterparts, here too all runs were successful with the graded penalty method. Evidently, the graded penalty method shows the expected and desired result of being better equipped to deal with the constraint landscape than the single penalty approach.

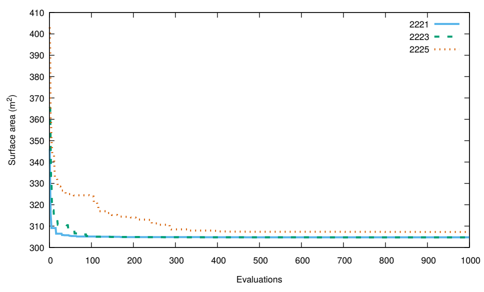


(a) 222x configurations

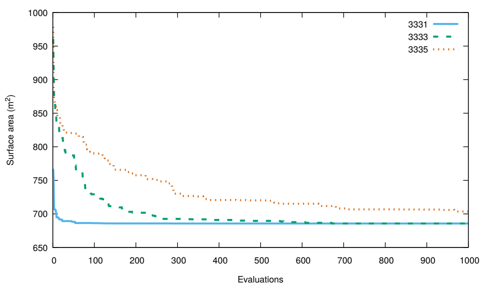


(b) 333x configurations

Figure 4.4: Mean convergence of the compliance over five runs using graded penalty values based on the number of constraint violations.



(a) 222x configurations



(b) 333x configurations

Figure 4.5: Mean convergence of the surface area over five runs using graded penalty values based on the number of constraint violations.

Mean convergence results of the graded penalty method for the surface area objective show similar improvements to those of the compliance objective. This holds for both the 222x configurations in Figure 4.5a, and the 333x configurations in Figure 4.5b. Furthermore, with all runs now being completed successfully, a more accurate analysis of the convergence behaviour is possible. Particularly for the 333x configuration it is now possible to clearly observe the difference in convergence speed as the number of spaces is increased from 1, to 3, and to 5, whereas this (expected) behaviour was not clear at all with the single penalty approach (Figure 4.3b).

In terms of constraint violation probability, the results of the graded penalty method are also an improvement over those of the single penalty method. This can be observed for both the compliance objective in Table 4.4, and the surface area objective in Table 4.5. Most notably, the probabilities around 0.99 observed in many cases with the single penalty method (e.g. for compliance in Table 4.2) have disappeared. Even so, many of the constraint violation probabilities are still around or above 0.5. Especially the number of violations of the existence constraint for the 3335 configuration are cause for concern, since this suggests that this will remain a problem with larger sized designs, which often have to be dealt with in real world applications.

Config.	No overlap	Ground connected	Existence	Cuboid shape	Connected cuboid
2221	N/A	0.361342915	0.058746357	0.453979888	N/A
2223	0.330288406	0.321899619	0.153398193	0.474461674	N/A
2225	0.475043604	0.345127277	0.305617513	0.443144107	N/A
3331	N/A	0.512244896	0.019280040	0.592070113	0.107721204
3333	0.114589803	0.476313711	0.131886344	0.564315490	0.100258658
3335	0.159904807	0.407218686	0.833920127	0.474255397	0.075510223

Table 4.4: Mean constraint violation probability over five runs for minimal compliance optimisation with a graded penalty value for various problem configurations.

Config.	No overlap	Ground connected	Existence	Cuboid shape	Connected cuboid
2221	N/A	0.294695023	0.065057210	0.449316221	N/A
2223	0.352604824	0.150746218	0.153169050	0.475316636	N/A
2225	0.527390468	0.389167427	0.285392221	0.455517944	N/A
3331	N/A	0.482476598	0.048915412	0.523626333	0.121148832
3333	0.130755993	0.477704137	0.144945542	0.573464583	0.106156523
3335	0.173648794	0.400757221	0.839106603	0.477907732	0.086331151

Table 4.5: Mean constraint violation probability over five runs for surface area optimisation with a graded penalty value for various problem configurations.

Visualisations of example results for some building spatial designs are analysed next. A first observation is that different configurations result in different spatial designs. This substantiates the need for optimisation since results from one configuration do not generalise for another configuration. There is a clear distinction between the results from minimal compliance optimisation (Figure 4.6) and those of surface area optimisation (Figure 4.7). Surface area optimisation leads to compact cuboid, or near cuboid, shapes, as might be expected. Minimal compliance optimisation on the other hand produces a variety of shapes. Little use is made of the extra space in the 333x configurations. For the 3333 configurations this may be explained by the availability of only three spaces. There is a limited number of valid building spatial designs that can make use of the larger number of cells with only three spaces. Note that while it is possible to produce spaces consisting of a large number of cells, reaching such a situation becomes increasingly difficult with more cells while also satisfying the constraints. For example the largest space in the 3335 configuration for surface area optimisation (Figure 4.7d) consists of just two cells. This is likely to play a role in the limited use of space for both the 3333 and 3335 configurations. These frequent issues with constraints complicate exploring all feasible solutions in the search space. In particular, transitions between different feasible parts of the search space are challenging when many moves end up in infeasible parts of the search space. Evidently, this is something that needs to be addressed in future work.

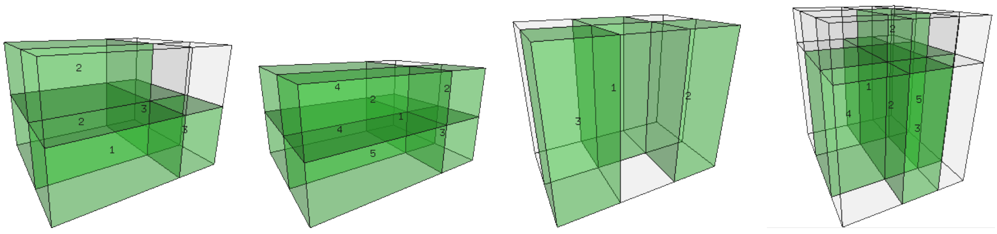

(a) 2223 **(b)** 2225 **(c)** 3333 **(d)** 3335

Figure 4.6: Examples of spatial designs optimised for minimal compliance.

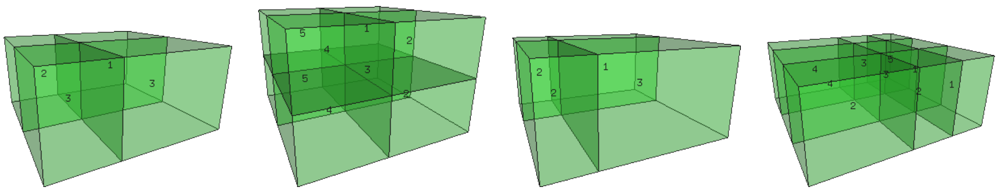

(a) 2223 **(b)** 2225 **(c)** 3333 **(d)** 3335

Figure 4.7: Examples of spatial designs optimised for minimal surface area.

4.7 Conclusions

4.7.1 Summary

With this chapter, the previously introduced supercube representation (Chapter 3) is subjected to first practical tests and assessed in combination with an optimisation algorithm. To this end, a brief overview of optimisation in building design, and the canonical Evolution Strategy (ES) have been presented. Furthermore, objective functions related to the two disciplines (structural, and building physics performance) considered in this thesis are also introduced.

Based on the canonical ES, and the knowledge about the number of infeasible (constraint violating) designs in the supercube representation (Chapter 3, Section 3.2.4), an ES variant has been developed to accommodate this situation. This algorithm considers variation operators (mutation, recombination) suitable to the mixed-integer nature of the supercube representation. Furthermore, penalty functions are used to deal with the constraints on the binary variables (Chapter 3, Section 3.2.2), while repair functions have been developed to satisfy the constraints on the continuous variables (Chapter 3, Section 3.2.3). With that, first steps have been taken towards answering RQ2, which calls for effective constraint handling.

This constraint handling ES is subjected to experiments comparing a single penalty method with a graded penalty method. In the single penalty method a fixed penalty value is returned regardless of the number of constraints that are violated, while the graded penalty method applies penalties that grow with the number of violated constraints. The experiments showed that, as expected, a graded penalty allows for a more effective search than a single penalty value. However, despite being more effective, constraint violations remain frequent. Due to this, it seems unlikely that larger supercube configurations than those considered here can be optimised effectively, even when using the graded penalty method. With this in mind, different constraint handling techniques have to be developed.

4.7.2 Future Work

As mentioned, although the improvements when using a graded penalty are promising, constraint violations still occur frequently, and likely prohibit the optimisation of larger building spatial designs. One direction in which the graded penalty approach could be improved is by increasing the granularity in which it penalises infeasible designs. In the version evaluated in this chapter a design that violates (for instance) the existence constraint once, would get the same penalty value as a design that violates the same constraint multiple times (for different spaces). By penalising designs based on a more fine-grained measure of constraint violation it should be possible to indicate a smoother path towards feasible space for the search process. This removes plateaus in the penalised part of the search space, but it might also introduce local optima.

Despite the possibilities to improve on the graded penalty approach, any penalty based approach has the inherent downside of spending time on infeasible solutions (except for the special – and unlikely – case were only feasible solutions are found). An alternative that does not suffer from this drawback is the use of specialised search operators that only navigate the feasible space. Naturally, developing such operators requires careful consideration of how to navigate the feasible search space. After all, every feasible solution should still be reachable, even if there are disconnected feasible regions. The development of specialised operators for the building spatial design problem will be investigated in the next chapter.

Another issue that was identified is that many optimised solutions seem to use a limited selection of the cells in the supercube. Although this may just be the end result of the optimisation process, it could also indicate that the search has difficulty moving from one part of the feasible space to another. Considering the number of constraint violations, and the number of changes that are often needed in the binary part of the search space to transition from one feasible solution to another, this is a serious concern. Fortunately, it should be possible to address the reachability of all feasible solutions with specialised operators, as will be addressed in Chapter 5. Additionally, in future work it may be worth investigating how global optimisation strategies such as niching [93] can improve the variety of the discovered solutions.