Chapter 3

Design Space Representations

This chapter aims to answer RQ1, which asks for a problem representation for building spatial design. This is essential since, in order to optimise anything, a problem representation is needed. That is, an encoding of the problem in decision variables. Broadly, two classes of representations are considered here: Superstructure based representations, and superstructure free representations. This chapter first discusses the differences between these two types. As will become clear in this chapter, the superstructure representation is preferred here for evolutionary computation. As such, a superstructure is defined for spatial design representation, termed the supercube. Following that, limitations of the chosen representation are analysed and compared to an alternative representation. Finally, the chapter is summarised and directions for future work are discussed.

3.1 Superstructure or Superstructure Free

Algorithmic optimisation – like any optimisation process – requires a problem representation. This representation formalises the design space. In other words, the space of possible solutions. If the representation is poorly defined the optimal solution may be excluded from the design space. Or there may be so many infeasible solutions that it becomes difficult to find any acceptable solution at all. A good representation is therefore paramount to finding high quality solutions in an efficient manner. Even so, different goals may benefit from different representations. Here two classes are investigated: superstructure representations, and their counterpart, superstructure free representations.

A superstructure prescribes the possible solutions by encoding each of them in a vector of constant length. This results in a preselection of solutions that can be found by the optimiser, and which cannot, because they exist outside the superstructure. By limiting the design space in this manner, the optimiser can focus its search, instead of having to consider every possible alternative. On the other hand, if the global optimum is not contained in the superstructure, it will never be found. Given the fixed size of the superstructure, it may be possible to describe it as a mathematical program. If this is done the problem can be solved by standard solvers (such as described in e.g. [47]). The superstructure terminology originates from the process industry, where configurations of chemical engineering plants are optimised. For example, Jackson [56] described flow configurations in chemical reactors with a superstructure, although without explicitly mentioning the term. Various more recent works [8, 11, 92] also employ superstructures in other engineering fields.

To give an example of a superstructure, let us consider the optimisation of sunlight illumination in a building. Using binary variables, windows can be in- or excluded from the building design (where each binary variable is associated with a specific window). Assume also a fixed number of variables per window that indicate their location in the building. The maximal number of windows is fixed at the start of the optimisation process, resulting in a fixed number of binary and positioning variables. If the optimiser starts with four windows that can be turned on or off, it will find a solution with somewhere between zero and four windows. However, it will never find a solution with five windows, even if that would outperform all other solutions.

Since superstructures limit the optimisation to a predefined region of the design space, superstructure free optimisation has been suggested to be able to reach all possible solutions. Conversely to superstructures, the optimiser is now faced with the task of searching through all possibilities, but there is no risk of excluding the global optimum. Emmerich et al. [43] propose the use of replacement, insertion, and deletion rules to modify (mutate, recombine) chemical process configuration designs in evolutionary algorithms. However, the development of these modification operators requires domain knowledge. Voll et al. [96] suggest a more general framework that uses generic replacement rules in evolutionary algorithms. A similar strategy is followed in [36], where it is exemplified for the optimisation of decision diagrams. Other examples of superstructure free design spaces include the work found in [6, 62].

Many optimisation methods rely on the number of variables remaining fixed during optimisation, which is not the case for a superstructure free representation. Due to this, many optimisation methods can not handle superstructure free representations. However, algorithms such as simulated annealing, evolutionary algorithms, and heuristic local search can handle superstructure free representations. Simulated annealing has been used in the design of processes, e.g. in [35]. In the field of structural design, [59] describes a superstructure free approach in the optimisation of structural topologies. Moreover, in [52] simulations of a co-evolutionary design process (these simulations can also be interpreted as asymmetric subspace optimisation [75]) are used to find a building spatial design for which a structural design created by certain design rules shows minimal strain energy.

To provide an example of a superstructure free representation, consider again the optimisation of sunlight illumination in a building. With a superstructure free representation it is possible to only consider the fixed number of positioning variables per window. Now, the optimiser can freely add and remove sets of positioning variables to reduce or increase the number of windows. There are pitfalls, however. When a window is removed its positioning information is lost. Then, when the optimiser later adds a window, no information is retained on which positions have been tried before. With the superstructure representation some information is saved (although still only the last known position), even when a window is temporarily excluded from the design. Furthermore, an optimiser may attempt to evaluate designs with excessively many windows, whereas a human designer might realise that there is no chance that this particular design will perform well.

3.2 Supercube

Based on the definitions of superstructure and superstructure free representations it becomes clear that with a superstructure a more focused search is possible, whereas a superstructure free representation allows for a more exploratory search. A focused search makes it possible to cover a significant part of the focus area, and possibly guarantee local optimality (RQ3). Given extensive knowledge on this focus area, it may be possible to analyse the data to gain new design insights (RQ4). As such, this section proposes a superstructure representation for building spatial design, namely: the supercube.

The supercube is a superstructure that aims to encode the spaces making up a spatial design, as well as the shapes of these spaces. This is achieved by mapping each space to a 3D rectangular grid, as visualised in Figure 3.1. By using binary variables to turn different cells in this grid on or off, the shape of a space is carved out. Further, each row, column, and pillar of the 3D grid can be stretched or shrunk with continuous variables, increasing the diversity of possible shapes. Using this 3D grid limits the spaces to be composed out of cuboid (3D rectangles) shapes. However, this also means the optimiser does not have to search through a massive number of irregular shapes until it finds something that works reasonably well.

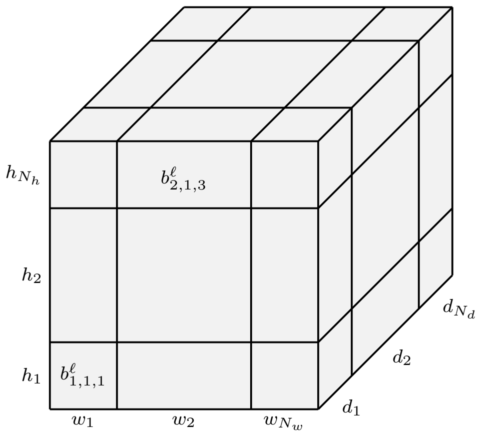


Figure 3.1: The grid used in the supercube representation, consisting of continuous width, depth and height divisions denoted by w_i, d_j, h_k, and for every space ℓ a collection of binary cells $b^{\ell}_{i,j,k}$.

More formally, the supercube is delimited by the parameters N_{spaces}, N_w, N_d, and N_h. These parameters indicate the number of spaces encoded in the supercube, and the number of segmentations in width, depth, and height respectively. Each of these is indexed as follows: $\ell \in \{1, \ldots, N_{spaces}\}$, $i \in \{1, \ldots, N_w\}$, $j \in \{1, \ldots, N_d\}$, $k \in \{1, \ldots, N_h\}$. Given these parameters, the decision variables $b^{\ell}_{i,j,k} \in \{0, 1\}$, $w_i \in \mathbb{R}^+$, $d_j \in \mathbb{R}^+$, and $h_k \in \mathbb{R}^+$ are to be optimised. A binary variable $b^{\ell}_{i,j,k}$ indicates whether the cell with indices i, j, k is active for the space with index ℓ. The continuous variables w_i, d_j, h_k denote the length of the grid segments.

3.2.1 Examples

To illustrate how the variables in the supercube can be used to encode building spatial designs a few visual examples are introduced. Note that although these examples are in 2D, everything discussed here extends to 3D.

First, Figure 3.2 shows how changes to the continuous variables of the supercube affect the building spatial design. On the left, a supercube representation is shown for which all dimensioning variables (w_i, d_j) have a value of 1. On the right, w_1 is changed to 2, increasing the width of the whole column of cells. The dashed lines in the figure indicate that the cells in the depicted supercube are active ($b^{\ell}_{i,j,k} = 1$). Given that in this case all cells are active, and belong to a single space ($\ell = 1$, for every cell), the corresponding building spatial design has the same exterior shape as these visualisations of the supercube. However, the internal divisions are not present in the spatial design. Those merely serve to indicate the division of the supercube into cells.

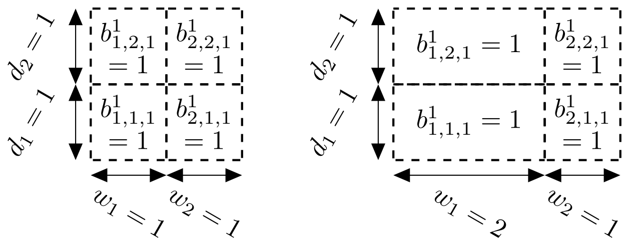


Figure 3.2: Example of continuous variation in the supercube representation.

In Figure 3.3 an example of variation resulting from the discrete variables is shown. On the left, the binary variable $b^1_{2,1,1}$ – associated with the top-right cell – is set to 1 (indicating an active cell), while on the right it is set to 0 (i.e. it is inactivate). Here, the dotted lines indicate an inactive cell ($b^{\ell}_{i,j,k} = 0$). As a result of this, the building spatial design is changed from a square-shaped form on the left, to an L-shaped form on the right. Like before, the outer contour of dashed lines correlates with the shape of the single space encoded for this particular building spatial design.

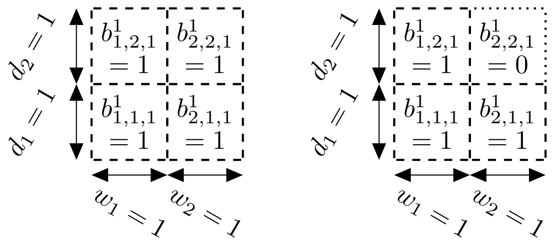


Figure 3.3: Example of discrete variation in the supercube representation.

The binary variables from the example in Figure 3.3 can be described equivalently by a 2×2 matrix (extending to 3D building spatial design naturally requires a 3D matrix). Recall that a separate bit-mask is used for each space. Here each bit-mask is denoted by $\mathbf{B}^{\ell}$, with $\ell \in 1, \ldots, N_{spaces}$. When three spaces are considered, this results in three matrices, this is depicted in Figure 3.4. The building spatial design is drawn with solid lines, to differentiate it from the dashed and dotted lines previously used in the visualisations of the supercube. Note that, unlike in the previous figures, not all cell separators are shown. This is because those separators merely serve to visualise the cells of the supercube, whereas here a building spatial design is shown. In the building spatial design, separators only exist on their exterior. That is, only between spaces, and to the outside. Notably, there is no divisor between the left half of space 3 and its right half.

$$\mathbf{B}^1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad \mathbf{B}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{B}^3 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

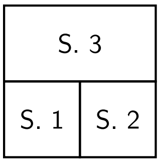


Figure 3.4: Example of multiple spaces (abbreviated by 'S.') encoded in the supercube representation.

3.2.2 Constraints on the Binary Variables

When it is not restricted by constraints, the supercube representation as described so far may lead to infeasible building spatial designs, for example floating spaces could occur. Therefore the following constraints are defined on the binary variables:

C1 There should be no overlap between spaces.

C2 All components of the building should be connected to the ground.

C3 Each space must exist, i.e. it must have at least one active cell.

C4 Each space forms a cuboid (3D rectangle) out of the cells assigned to it.

These four topology constraints are described here in mixed-integer nonlinear programming (MINLP) form, see also [17, 21, 25]. By describing the constraints in MINLP form, it is possible to use standard solvers (e.g. [47]) in combination with the supercube representation, conditional on all objective functions also being defined in MINLP form.

Constraint C1 (No overlap). Overlap between building spaces is not allowed, because it is impractical, and may lead to erroneous results in subsequent design analysis. A constraint is needed for this because every space is represented by a separate bit-mask (enumerated by ℓ) of all cells in the supercube. Thus, overlap is not automatically prevented in the representation. Equation 3.1 achieves this by taking the sum of each cell over all masks. As a result of the binary representation, only if such a sum is smaller or equal to one, no overlap exists at that position. If the bits corresponding to the same cell were active in different bit masks, the sum would be greater than one, and the overlap would be detected.

$$\forall_{i,j,k} \sum_{\ell=1}^{N_{spaces}} b^{\ell}_{i,j,k} \leq 1 \tag{3.1}$$

Constraint C2 (Ground connected). Building spatial designs normally stand on the ground. Since this is difficult to check by a simple equation if vertical gaps are allowed in the spatial design, the constraint considered here also enforces that there are no vertical gaps. As a result of this constraint it is not possible to describe structures with cantilevers, overhangs or archways. The no vertical gaps restriction could be abandoned if one is willing to use more complex procedures to check ground connectedness constraints, but that would complicate the use of standard mathematical programming solvers. In order to enforce that every space is ground connected, and no vertical gaps exists, transitions from 0 to 1 for i, j beams are disallowed. This ensures that for every pillar (i.e. a vertical column in the supercube) all active cells have consecutive k indices, such that no vertical gaps exist. Moreover, if $b^{\ell}_{i,j,1} = 0$ no cells can be active in this beam without inducing a 0 to 1 transition, and thus violating the constraint. As such, ground connectedness is also ensured.

To check this based on the supercube variables, let $b_{i,j,k}$ be the outcome of a logical OR of all ℓ bits belonging to cell i, j, k. In equations: $\forall_{i,j,k} : b_{i,j,k} = sgn(\sum_{\ell=1}^{N_{spaces}} b^{\ell}_{i,j,k})$, where the $sgn()$ may be omitted if the previously defined no overlap constraint (Equation 3.1) is satisfied. Using $b_{i,j,k}$, if Equation 3.2 holds, the building spatial design has no vertical gaps and stands on the ground. In short, it checks that no change from 0 to 1 occurs when moving upwards in a vertical beam of cells.

$$\forall_{i,j} : \left(\sum_{k=1}^{N_h - 1} (1 - b_{i,j,k})\, b_{i,j,k+1} \right) = 0 \tag{3.2}$$

Constraint C3 (Existence). The number of described spaces is kept constant. That is, all spaces should exist in the resulting spatial design. A building is usually designed with a certain purpose in mind, resulting in a requirement for a specific number of spaces. With the supercube the desired number of spaces is easily specified, but that every space exists (has an active cell) requires a constraint. Checking the existence of each space is achieved by ensuring that every space is described by at least one cell. In Equation 3.3, this is achieved by simply taking the sum over all combinations of indices of a space, and checking whether this sum is at least one. I.e., at least one active cell exists for this space.

$$\forall_\ell : \sum_{i=1}^{N_w} \sum_{j=1}^{N_d} \sum_{k=1}^{N_h} b_{i,j,k}^{\ell} \geq 1 \tag{3.3}$$

Constraint C4 (Cuboid). Spaces are constrained to cuboid shapes for practicality. The restriction to cuboid spaces also helps to keep the size of the design space manageable. Consider, for instance, how the number of – especially infeasible – designs would increase when slanted structures are allowed. Searching through these to find reasonable designs would be prohibitively expensive.

To check this condition, pairwise comparisons can be made between rows, columns, and pillars of cells. In this comparison the number and position of transitions from 0 to 1, and from 1 to 0 have to be checked for equivalence. However, due to the nature of the bit-masks used in the supercube, these checks would have to be done in two directions. When comparing two rows with each other, for instance, transitions have to be checked from left to right, as well as from right to left.

In order to prevent having to check in two directions, first each bit-mask in the supercube will be extended with a single layer of cells all around it. These new cells are set to have no relation to any space ($b_{i,j,k}^{\ell} = 0$), this extension is described by Equation 3.4:

$$\begin{gathered} \forall_\ell : \forall_{i,j,k} \in \{0, \ldots, N_w + 1\} \times \{0, \ldots, N_d + 1\} \times \{0, \ldots, N_h + 1\} : \\ i = 0 \lor j = 0 \lor k = 0 \lor i = N_w + 1 \lor j = N_d + 1 \lor k = N_h + 1 \Rightarrow b_{i,j,k}^{\ell} = 0 \end{gathered} \tag{3.4}$$

Now it is possible to identify the position, and number of transitions from 0 to 1, and from 1 to 0 by checking a single direction, as shown in Figure 3.5. This process is formulated in Equation 3.5 for transitions from 0 to 1, and in Equation 3.6 for transitions from 1 to 0 (where transitions are checked from lowest to highest index value). Note that these equations serve as example for transition checks in the height, and similar equations are used for width and depth.

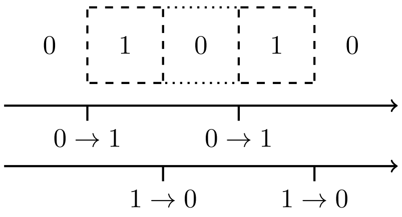
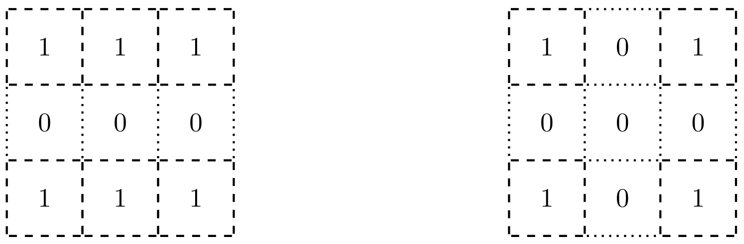


Figure 3.5: Checking transitions from 0 to 1, and from 1 to 0. Zeros without border indicate cells added by Equation 3.4.

$$\forall_\ell : \forall_{i_1,j_1,i_2,j_2} : \left(\left(\sum_{k=1}^{N_h} k\left(1-b^\ell_{i_1,j_1,k-1}\right)b^\ell_{i_1,j_1,k}\right) - \left(\sum_{k=1}^{N_h} k\left(1-b^\ell_{i_2,j_2,k-1}\right)b^\ell_{i_2,j_2,k}\right)\right)\left(\sum_{k=1}^{N_h} b^\ell_{i_1,j_1,k}\right)\left(\sum_{k=1}^{N_h} b^\ell_{i_2,j_2,k}\right) = 0 \quad (3.5)$$

$$\forall_\ell : \forall_{i_1,j_1,i_2,j_2} : \left(\left(\sum_{k=1}^{N_h} k\left(1-b^\ell_{i_1,j_1,k+1}\right)b^\ell_{i_1,j_1,k}\right) - \left(\sum_{k=1}^{N_h} k\left(1-b^\ell_{i_2,j_2,k+1}\right)b^\ell_{i_2,j_2,k}\right)\right)\left(\sum_{k=1}^{N_h} b^\ell_{i_1,j_1,k}\right)\left(\sum_{k=1}^{N_h} b^\ell_{i_2,j_2,k}\right) = 0 \quad (3.6)$$

Unfortunately, this still allows for some designs that ought to be infeasible. For example, Figure 3.6a shows how a row of zeros is allowed. This, however, makes it possible for a space to consist of multiple disconnected parts, which is not desirable. At the same time, rows of zeros have to be allowed, to ensure parts of the bit-mask can remain empty, leaving room for other spaces. In Figure 3.6b it is shown how taking this further can lead to many disconnected components of a space.



(a) A row of zeros is allowed. **(b)** A cross of zeros is allowed.

Figure 3.6: Designs that satisfy Equations 3.5 and 3.6, but should be infeasible.

To prevent these voids, and to ensure spaces are truly cuboid, Equation 3.7 is introduced. This equation enforces spaces to have a connected, ortho-convex shape. As before, this also relies on the layer of zero cells as added by Equation 3.4. With Equation 3.7 the number of changes from zero to one are counted for each space, and for each axis. Any space where there are multiple changes from zero to one is not fully connected and therefore infeasible. In conjunction with the previous equations (3.5 and 3.6) this ensures that every space has a fully occupied cuboid shape.

$$\begin{gathered}\forall_\ell : \\ \forall_{i,j} : \sum_{k=0}^{N_h} \left(1 - b^\ell_{i,j,k}\right) b^\ell_{i,j,k+1} \leq 1 \\ \forall_{i,k} : \sum_{j=0}^{N_d} \left(1 - b^\ell_{i,j,k}\right) b^\ell_{i,j+1,k} \leq 1 \\ \forall_{j,k} : \sum_{i=0}^{N_w} \left(1 - b^\ell_{i,j,k}\right) b^\ell_{i+1,j,k} \leq 1\end{gathered} \tag{3.7}$$

3.2.3 Constraints on the Continuous Variables

In order to compare between different building spatial designs the total volume V_0 is kept constant during optimisation. This can be achieved by a constraint on the continuous variables. Since only the active cells (i.e. cells that are encoding part of a space) are relevant to the volume computation, inactive cells have to be excluded. To do this, here (as in Constraint C2) $b_{i,j,k}$ is again taken to be the result of a logical OR over all ℓ bits of a cell i, j, k. By multiplying with $b_{i,j,k}$, the volume contribution of any cell that is inactive for all spaces ($b_{i,j,k} = 0$) will be zero. For active cells, the volume contribution is simply the product of the relevant width, depth, and height variables. Note that $b_{i,j,k}$ is used here instead of summing over all spaces with $b^\ell_{i,j,k}$ to ensure that the volume is still correct even if a cell is active for multiple spaces (and thus violates Constraint C1). Together, this yields the equality constraint in Equation 3.8 below:

$$\sum_{i=1}^{N_w} \sum_{j=1}^{N_d} \sum_{k=1}^{N_h} b_{i,j,k} w_i d_j h_k = V_0 \tag{3.8}$$

In addition, all continuous variables should be positive or taken from a range of positive values:

$$\forall_i : w_i > 0, \qquad \forall_j : d_j > 0, \qquad \forall_k : h_k > 0 \tag{3.9}$$

3.2.4 Analysis of the Supercube Search Space

To make optimal use of the supercube representation, understanding of its workings and limitations is essential. To this end, here the representation is analysed with a focus on duplicates and feasibility. Duplication is important, because having duplicates in a representation may hamper the search process. For one, evaluating the same solution multiple times is undesirable, but there may also be a problem with bias if some some solutions are encoded more frequently than others. How many designs are (in)feasible in the representation is indicative of how much time a search algorithm might spend on actually useful (feasible) solutions, versus infeasible solutions.

Duplicate Designs

Within the previously described supercube representation, duplicate building spatial designs can occur. Here a number of duplication types are discussed.

Stretch duplicates indicate duplicate designs that have different binary representations for a space, but still result in the same building spatial design due to variation of the continuous variables. An example of this is shown in Figure 3.7. By reducing the width of the two active cells in Figure 3.7a, and activating the other two cells, the representation in Figure 3.7b is reached, with an equivalent spatial design associated to it.

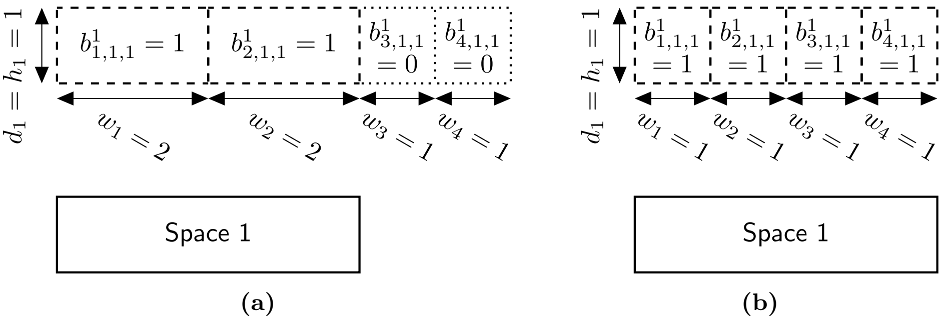


Figure 3.7: Example of stretch duplicates. Top: Supercube representations; Bottom: Spatial designs.

Swap duplicates result in equivalent building spatial designs by swapping the binary assignments of one space with those of another. In Figure 3.8 this is exemplified by swapping space 1 and 2 with each other. Although the spaces now have different identifiers, these identifiers carry no meaning, they merely serve to differentiate between spaces. As such, since the shapes of the spaces and the building stay the same, the design is equivalent. Note that duplicates of this type are not affected by the continuous variables at all.

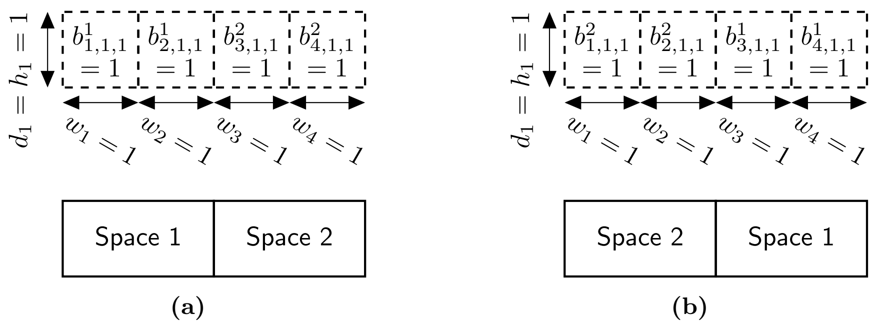


Figure 3.8: Example of swap duplicates. Top: Supercube representations; Bottom: Spatial designs.

Shift duplicates are equivalently shaped building spatial designs, encoded in a different selection of cells in the supercube. The example in Figure 3.9 shows how this can occur. Although in this case the continuous variables influence whether the design is equivalent, both binary representations can be stretched in the same way by the continuous variables (assuming all continuous variables of a dimension have the same bounds).

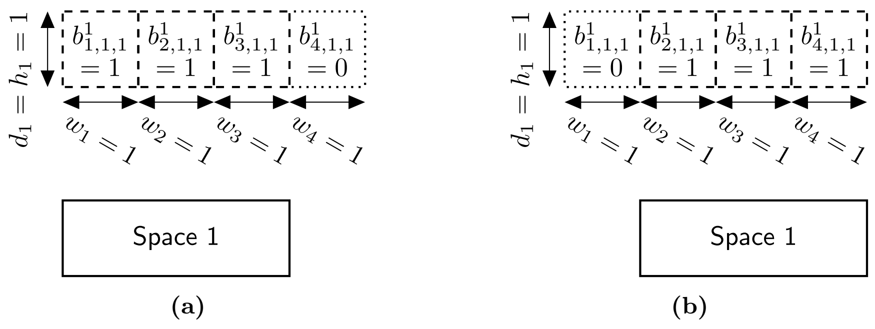


Figure 3.9: Example of shift duplicates. Top: Supercube representations; Bottom: Spatial designs.

Other types of duplication, besides those mentioned up till now, may be considered. So far all considered duplication types produce the same building spatial design. However, rotated and mirrored designs may be considered duplicates as well under certain conditions. These are duplicates in the sense that they are structurally the same, while their building spatial design differs. It is important to note that what is or is not a duplicate is largely determined by the considered objective function(s). Rotations of a design, for example, may be equivalent in a structural sense, but differ in terms of sunlight illumination. This is also the case for wind, which may have different intensities for each direction. Since rotations and mirror images of designs will not always be considered duplicates, these duplication types are not investigated further here.

Duplicates and Feasibility

The binary representation considered for the supercube has a combinatorial search space that encodes $2^{N_{spaces} \times N_{cells}}$ designs. Here $N_{cells} := N_w \times N_d \times N_h$, i.e. the number of cells in a bit-mask of the supercube. Feasible designs (that is, designs satisfying all constraints) are duplicated exactly $N_{spaces}!$ times,[1] in the swap duplication sense. This is because for a design to be feasible, every space must consist of at least one cell (Constraint C3). As a result, cells assigned to space A may be swapped with those of space B (without losing any structure , such that space A gets the cells of space B and vice versa. Since every space has at least one cell assigned (Constraint C3), $N_{spaces}!$ such configurations of the same feasible building spatial design can be reached by swaps.

For infeasible designs the number of swap duplicates is *at most* $N_{spaces}!$. The exact number of times a specific design is duplicated depends on how many spaces have equivalent bit-masks. When the bit-masks of all spaces are distinct, $N_{spaces}!$ duplicates exist. If there are equivalent bit-masks, there are fewer duplicates. This is because swaps between equivalent bit-masks result in equivalent representations, and thus do not lead to duplicates. For example, when two spaces both have no active cells (their bit-masks are all zeros), swapping between them results in the same representation.

Due to their dependence on the continuous variables, stretch and shift duplicates are much less likely to occur, and as such not analysed in detail here.

Alternative Representation

It is clear now that the proposed supercube representation has a number of limitations. Large numbers of infeasible, as well as duplicate designs are represented. Here an alternative is investigated that replaces the binary variables with integers. This is visualised in Figure 3.10. Instead of using a binary matrix $\mathbf{B}$ for each space, a single

[1]Assuming $N_{cells} \geq N_{spaces}$. If this does not hold, no feasible designs exist.

integer matrix **I** is used. In the integer matrix every element holds the index of the space to which the corresponding cell belongs, while 0 still indicates an inactive cell.

This representation has a combinatorial search space with $(N_{spaces} + 1)^{N_{cells}}$ designs, resulting in a smaller search space than the binary supercube. Notably, only designs that are infeasible because they violate Constraint C1 (no overlap) are not represented anymore.

$$\mathbf{B}^1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad \mathbf{B}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{B}^3 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{I} = \begin{bmatrix} 3 & 3 \\ 1 & 2 \end{bmatrix}$$

Figure 3.10: Example of integer instead of binary encoding for the supercube representation.

Although this alternative representation reduces the size of the search space, it cannot make use of the previously defined MINLP constraints. Having to find new MINLP formulations, or discarding the idea of mathematical programming altogether, makes this an alternative of limited interest. Furthermore, procedures to deal with the remaining infeasible regions are needed, regardless of using the binary or integer encoding. Moreover, all the discussed duplication types remain present in the considered alternative. A downside of the binary encoding, however, is that the space complexity grows from $O(\log_2 N_{spaces} N_w N_d N_h)$ to $O(N_{spaces} N_w N_d N_h)$, which limits the size of the buildings that can be represented.

Given these facts, the remainder of this work makes use of the binary representation. Nevertheless, studying alternative representations may make it possible to optimise more efficiently, and should be considered in the future.

3.3 Conclusions

3.3.1 Summary

In this chapter the differences between superstructure and superstructure free representations have been investigated. Superstructures allow for an extensive search for the optimum with a great variety of algorithms, within a predefined region. On the other hand, superstructure free representations make it possible to explore all options and potentially discover surprising solutions. As such, a superstructure is more suitable for pure optimisation, while free representations allow more extensive exploration.

Based on this analysis, and the goal considered in RQ1, a superstructure representation has been defined. This representation, termed the supercube representation, allows for the description of building spatial designs with cuboid spaces. In addition, constraints on the supercube variables were defined to ensure feasibility of the designs. Here feasibility was considered both in a practical sense (what can the simulator handle), and in a realistic sense (what can actually be built).

The supercube representation was analysed to verify its suitability to be used in practice, and to understand its limitations. Based on the analysis, limiting factors to the supercube representation are the large number of infeasible designs that are represented, and the number of duplicate designs, both in the feasible and infeasible space. A considered alternative somewhat reduced the number of infeasible designs represented. However, this advantage was found to be very limited, and the alternative would also require new constraint formulations. In addition, regardless of using the supercube representation or the alternative, sophisticated constraint handling techniques would have to be developed. Due to the limited advantage and the required time investment for new constraint definitions, the supercube representation was regarded to be of greater practical use.

3.3.2 Future Work

A number of interesting research directions to follow up on the work presented here remain. For instance, there is the question whether better representations exist than those proposed here. It would be of particular interest to reduce the number of designs that need to be searched through. Defining a representation that excludes the infeasible designs from the search space would naturally greatly simplify the optimisation process. At the same time, care needs to be taken when defining such a representation to ensure none of the feasible design alternatives are lost.

Another improvement to the existing representation would be one that does not contain duplicate designs. However, beyond excluding exact duplicates, this requires careful consideration of what constitutes an equivalent design for the considered problem. Rotated designs, for instance, may be equivalent in a structural sense, but not in terms of solar illumination.

In the real world, spatial designs are not restricted to spaces with cuboid shapes as considered here. Another follow up question is thus how a representation can be defined that removes this restriction. Moreover, can such a representation provide a toggle mechanism to choose whether it is restricted to cuboid shapes or not? This mechanism would make it possible to investigate only cuboid designs when a broader analysis is not needed, while still providing the freedom to search through all designs for use cases that require it.

An ideal representation would combine all of the aforementioned aspects. Furthermore, it could be a generic spatial design representation. One aspect in that would be the option to include or exclude certain designs, based on what is considered a duplicate for the optimisation problem under consideration. Naturally, a representation that solves all of these issues is more challenging, but each individual challenge can serve as a milestone towards it.

Given any of these suggested representations it would be valuable to still be able to define mathematical programming constraints. Although this is yet another challenge, it is a desirable property of a design space representation. Even so, depending on the representation it may simply not be possible, in which case it would be good to prove that this goal is unattainable.