A Model of Co-evolutionary Design

MARY LOU MAHER
Department of Architectural and Design Science
The University of Sydney, NSW 2006, Australia

Abstract *Computational evolution provides a mechanism for searching a space of potential solutions according to a specified fitness function. In design, the search for potential solutions is often interleaved with changes in the specifications of the solution. We have developed a model of design that uses a co-evolutionary process. Specifically, design is modeled as a parallel search for both design requirements and design solutions. Further, we have developed a co-evolutionary process in which the interaction between requirements and solution redefine the current fitness function. The concepts of fitness and convergence in computational evolution do not necessarily have the same meanings in a co-evolutionary process in which the fitness function changes. An additional consideration not present in computational evolution is the interaction between the parallel search spaces. We demonstrate how the design model can be implemEnted for structural system layout.*

Keywords: Co-evolutionary computation, Design, Evolutionary computation, GAs, fitness, convergence

1. Introduction

Computational evolution provides a mechanism for searching a space of potential solutions according to a specified fitness function. However, these evolutionary models typically require that the metric by which a solution is measured is defined before the solution space is searched. A characteristic of designing is a reconsideration of the requirements when a design solution is suggested. To achieve the effect of a changing metric, several different fitness functions should be used in different runs of evolutionary design processes. An alternative to changing the metric manually as the evolutionary process searches for solutions is to consider a co-evolutionary process in which the problem space is searched in parallel to the solution space. The contribution of this paper is the elaboration of a model of co-evolutionary design as an algorithm and its further elaboration through the application of co-evolutionary design to a structural system layout problem.

Co-evolution is the term used to identify the process in nature in which two or more species interact so intimately that their evolutionary fitness depends on each other. Biological co-evolution has been the inspiration for a class of computational algorithms called co-evolutionary computing (for example, see Hillis, 1991; Paredis, 1995; and Richard, 1995). Co-evolutionary design, as introduced in Maher (1994), is an approach to design problem solving in which the requirements and solutions of design evolve separately and affect each other. This paper firstly identifies co-evolutionary design by comparing it to design as search and design as exploration. Then an algorithm for co-evolutionary design is presented to highlight the process and parameters. A reconsideration of the purpose of the fitness function and its affect on convergence is necessary since the fitness function changes through the co-evolutionary cycles. The interactions between requirements and solutions of design may possibly add some new variables to both aspects of design, which may not only redefine the search space for requirements and solutions but also the fitness function. Based on the idea of mutualism, which is one of the three types of coevolution in nature (Smith, 1989), the interacting populations raise the level of fitness in both, rather than the two populations competing with each other or one population living off the other.

2. Design as search, exploration and coevolution

Simon (1969) characterises design as a search process, allowing the design process to be understood as one of the "sciences of the artificial". Since then the design research community has embraced the "design as search" model by formulating the goals, state spaces, and operators for various design domains and design problems. Although the search paradigm is very strong and still underpins much of problem solving, other models of design have been proposed that address the formalisation of design knowledge and design goals. Here we compare three related models: search, exploration, and co-evolution. The comparison highlights the difference between the three models in order to clarify co-evolutionary design.

2.1 DESIGN AS SEARCH

Design can be formalised as search when the goals of the design are well-defined before search commences and the focuses of design are not changed until a solution is found. In this model, ρ is the focus of the design which is derived from the problem requirements $\boldsymbol{P}$. A solution $\mathbf{S}_\rho$ is selected based on how well it satisfies $\boldsymbol{\rho}$. In computational evolution terms, ρ is the fitness function. The fitness function is defined once and convergence occurs when the solution with the best fitness function is found. Design as search can be modelled as:

$\boldsymbol{S}_\rho = \{S_{\rho 1}(\boldsymbol{P}), S_{\rho 2}(\boldsymbol{P}), S_{\rho 3}(\boldsymbol{P}), \ldots\}$
$S_f = \text{best}\{\boldsymbol{S}_\rho\}$

where
ρ is a given focus of design
$\boldsymbol{P}$ is the set of problem requirements
$\boldsymbol{S}$ is the search space of design solutions
S_ρ is a set of best design solutions for the focus ρ
$S_{\rho i}(\boldsymbol{P})$ is the *i*th best design solution for the focus ρ, $S_{\rho i}(\boldsymbol{P})\quad \boldsymbol{S}$
S_f is the final selected design solution

2.2 DESIGN AS EXPLORATION

Design becomes exploration when the focus of the design changes as the process continues. A focus, derived from the requirement space, determines which solution in a solution space is considered best. Design as exploration can be modelled as follows:

$\boldsymbol{S}_{\rho i} = \{S_{\rho i 1}(\boldsymbol{P}), S_{\rho i 2}(\boldsymbol{P}), S_{\rho i 3}(\boldsymbol{P}), \ldots\}$

$S_{f\rho i} = \text{best}\ \{S_{\rho i}\}$

$S_f \quad \{S_{f\rho 1}, S_{f\rho 2}, S_{f\rho 3}, \ldots, S_{f\rho n}\}$

where,
ρ_i is a given focus of design for i, i=1,2,…,n
$\boldsymbol{P}$ is a space of all possible problem requirements
$S_{\rho ij}(\boldsymbol{P})$ is the *j*th design solution corresponding to the focus ρ_i
$\boldsymbol{S}_{f\rho i}$ is the best design solution of $S_{\rho i}$ corresponding to the space of $\boldsymbol{P}$ with ρ_i
S_f is the final selected design solution

The idea of exploration can be illustrated using Figure 1, showing that a designer will see different parts of a solution space when he considers a different focus. Assuming that ρ is the focus, exploration allows the fitness function to change, but does not specific how it changes.

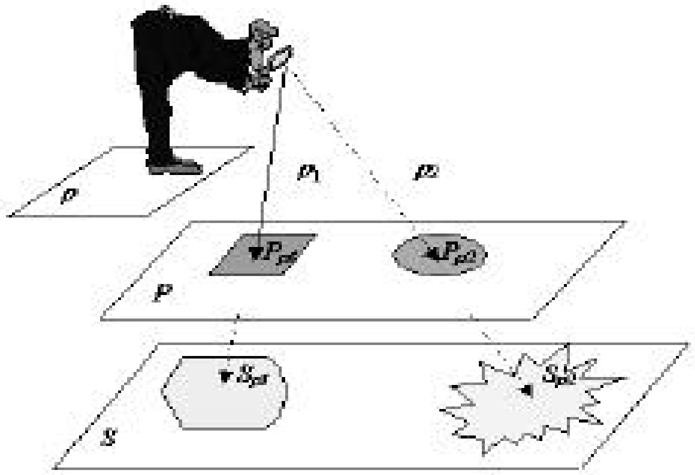


Figure 1. The idea of design as exploration

2.3 DESIGN AS CO-EVOLUTION

Design becomes co-evolution when the focus of the design can change, and the requirements space and solution space change through mutual interaction. The focus of the search is based on the requirements when searching the solution space, and based on the solutions when searching the requirement space. Design as co-evolution can be modeled as:

$\rho_i = f(P_i)$
$\boldsymbol{S}_{\rho i} = \{S_{\rho i 1}(\boldsymbol{P_i}), S_{\rho i 2}(\boldsymbol{P_i}), S_{\rho i 3}(\boldsymbol{P_i}), \ldots\}$
$S_{f\rho i} = \text{best}\{S_{\rho i}\}$
$S_f \quad \{S_{f\rho 1}, S_{f\rho 2}, S_{f\rho 3}, \ldots, S_{f\rho n}\}$

$\rho'_i = f(S_i)$
$\boldsymbol{P}_{\rho' i} = \{P_{\rho' 1}(\boldsymbol{S_i}), P_{\rho' 2}(\boldsymbol{S_i}), P_{\rho' 3}(\boldsymbol{S_i}), \ldots\}$
$P_{f\rho' i} = \text{best}\{P_{\rho' i}\}$
$P_f \quad \{P_{f\rho' 1}, P_{f\rho' 2}, P_{f\rho' 3}, \ldots, P_{f\rho' n}\}$

where,
ρ_i is a given focus for the design solution at time i
ρ'_i is a given focus for the design requirements at time i
$\boldsymbol{P}_i$ is the *i*th search space of problem requirements
$\boldsymbol{S}_i$ is the *i*th search space of design solutions
$P_{\rho' ij}$ $(\boldsymbol{S_i})$ is the *j*th design requirement corresponding to $\boldsymbol{S_i}$ and ρ'_i
$S_{\rho ij}$ $(\boldsymbol{P_i})$ is the *j*th design solution corresponding to $\boldsymbol{P_i}$ and ρ_i
$\boldsymbol{S}_{\rho i}$ is a set of design solution corresponding to the space of $\boldsymbol{P_i}$ with ρ_i
$\boldsymbol{P}_{\rho' i}$ is a set of problem requirement corresponding to the space of S_i with ρ'_i
$P_{f\rho' i}$ is the best design requirement of $P_{\rho' i}$
$S_{f\rho i}$ is the best design solution of $S_{\rho i}$
S_f is the final selected design solution
P_f is the final selected design requirements

The model of design as coevolution is illustrated in Figure 2. The interactions between two sets of spaces can occur through the change in focus of the search and by passing variables from one space to another. The downward arrow corresponds to the evaluation of ***S*** using ***P*** as the source of the focus. If no satisfactory solution is found with the stated design requirements, the solution space becomes the basis to derive the focus for searching the problem space. The upward arrow corresponds to the evaluation of ***P*** using ***S*** as the source of the focus. After searching for relevant requirements in the problem space, the new requirements space is the basis of the focus for the search for a solution (Maher and Poon, 1996). The interaction between the spaces can include transferring variables from one space to the other.

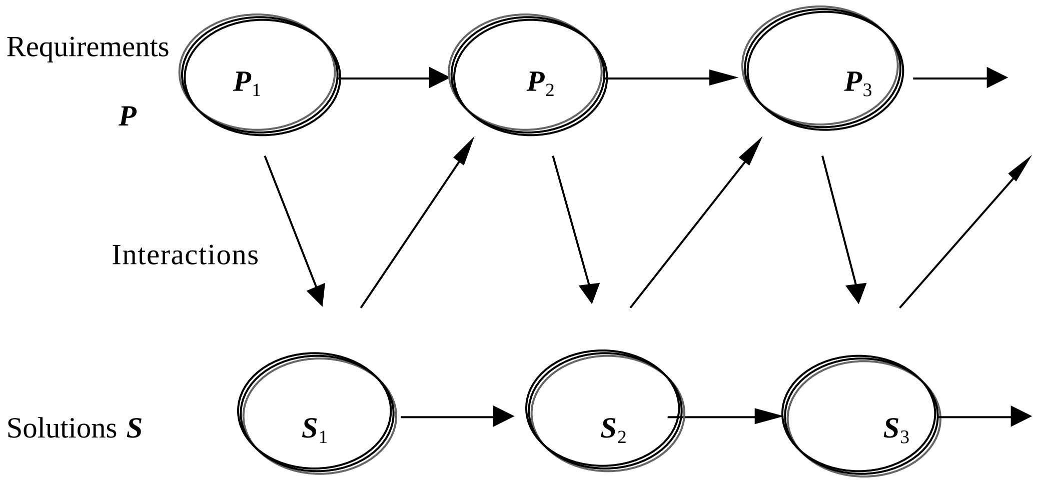


Figure 2. A model of design as exploration (after Maher, 1994)

2.4 COMPARISON OF SEARCH, EXPLORATION AND COEVOLUTION

Design as search finds the best design solutions from a part of the design space that satisfies the pre-defined requirements. Since design requirements are not changed during the design process, the model of design as search is suitable for well-defined design problems.

Design as exploration searches different parts of the solution space by changing the design focus. Viewed from different perspectives, different parts of the solution space become relevant. The difficulty in design as exploration is determining a way to change the focus of design. For instance conventional evolutionary computation which assumes a fixed fitness function does not simply apply to this design model.

Design as co-evolution explores the spaces of design requirement and design solution iteratively. Interactions between the spaces may add new variables into both design spaces. Usually, the requirements and solution of an ill-defined design problem can not be understood well enough to define a fixed focus. The search for potential solutions is often interleaved with changes in the requirements for the solution. Therefore, the model of design as co-evolution is appropriately used in conceptual design, where relatively little is known about the design problem. Figure 3 illustrates the input and output of the co-evolutionary model of design.

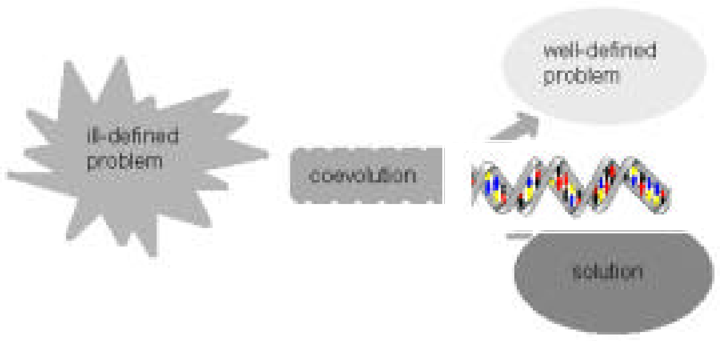


Figure 3. Input and output of co-evolutionary design.

3. Co-evolutionary Design Algorithm

Co-evolutionary design is characterised by having a search space of problem requirements and a search space of problem solutions. The algorithm has two phases, each one corresponding to a simple genetic algorithm. In the first phase, the problem space provides the basis for a fitness function used to evaluate alternatives in the design space. This corresponds to the downward arrow in Figure 2. In the second phase, the solution space provides the basis for a fitness function used to evaluate the problem space. This corresponds to the upward arrow in Figure 2. Each phase is essentially a goal-directed search process with a fixed goal.

First, the requirements space is used to derive a focus, as illustrated in Figure 4. The solution space is a population of alternative designs. Applying the evolution operators, the members of the population in the solution space go through repeated cycles of selection, reproduction, crossover, and mutation. At each generation, the solutions are evaluated against a fitness function. When the process has converged, and if the process has not yet reached the termination condition, the focus shifts and the search changes to the problem space.

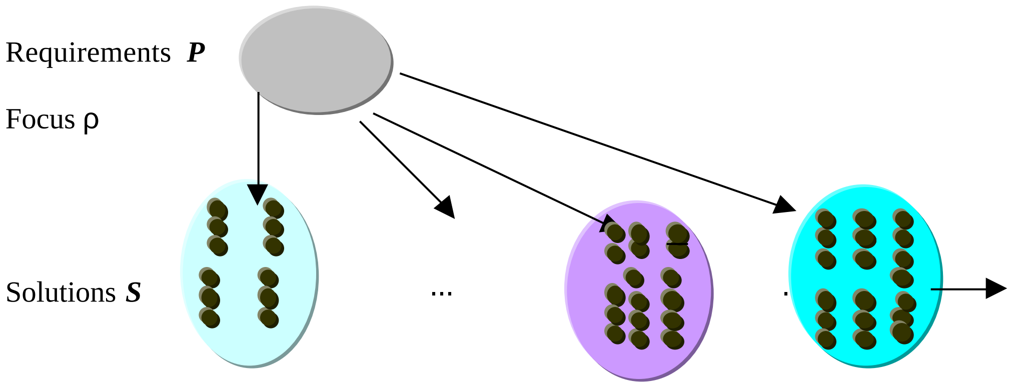


Figure 4. Evolutionary search for design solutions.

After convergence, the solution space is used to develop a focus for searching the problem space, as illustrated in Figure 5. The search space is a population of alternative requirements. Applying the evolution operators, the members of the population in the problem space go through repeated cycles of selection, reproduction, crossover, and mutation. At each generation, the requirements are checked for convergence. When the process has converged, the focus shifts and the search changes to the solution space. Logically, termination is not possible at this phase because this phase results in a new problem definition and does not necessarily have a corresponding design solution.

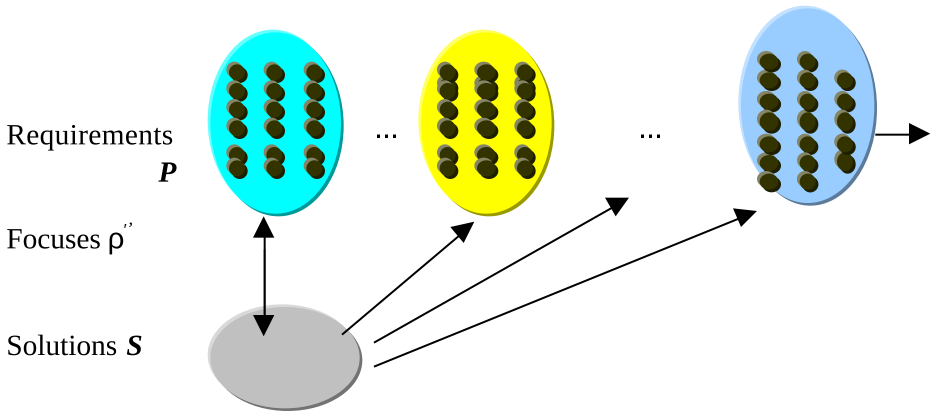


Figure 5. Evolutionary search for design requirements.

Figure 6 presents a general algorithm of coevolutionary design. Each of the two phases of co-evolutionary design is a search process using a simple GA and unchanging fitness function, denoted ρ. Therefore, each phase corresponds to one design focus and a change in phase indicates a change in focus. Using this algorithm for co-evolutionary design we need to reconsider concepts of evolution, their counterpart in GAs and their meaning in the co-evolutionary design process. Fitness is used as the criteria for determining which members of the population are selected to participate in the next generation. Convergence is defined within each phase as the criteria for changing the focus of the search. This distinction ensures that fitness is used to measure and compare the performance of individuals with the same fitness function, and convergence indicates that the current search space is no longer going to be searched. Termination is not tied to fitness or convergence and may be determined by an analysis of the changing focus as recorded in the set Ω, by a fixed amount of time for which co-evolutionary design is allowed to continue, or by the human designer.

```
/* CoDESIGN  algorithm */
T =1;                          /*co-evolutionary cycle counter */
v_p=1;                         /*local evolutionary counter of problem space*/
v_s=1;                         /*local evolutionary counter of solution space*/
u =1;                          /*co-evolutionary phase counter*/
t  =1;                         /* generation counter */
Ω ={};                         /* set of fitness functions*/
initialize genotypes of requirements P_t and solutions S_t;
While termination conditions, f(T,Ω ), are not satisfied
{
T=T+1;                      /*update the evolutionary cycle counter*/
u=1                         /*update the evolutionary phase counter*/
while u< 3                  /*two phase  coevolutionary design process*/
switch (u)
{
case u=1:
```

```
/* Phase 1: determine focus for new problem requirements, */
/* redefine problem requirement space, and search for best problem requirement*/
if $T \neq 1$                          /*no evolution  in problem space at the first cycle*/
{
$\rho'_{t,T} = f(\boldsymbol{S}_t)$;                      /*fitness function*/
$\Omega = \Omega \ \{\rho'_{t,T}\}$;          /*search focus*/
$\boldsymbol{P}_t = g(\boldsymbol{P}_t, \rho'_{t,T})$;       /*revised problem requirement space*/
t=1;                                   /*reset the evolutionary counter*/
While convergency conditions, $c(t,\Omega)$, are not satisfied
{
$\boldsymbol{P}_t$ ::= select genotypes in $\boldsymbol{P}_{t-1}$;
reproduce, crossover and mutate genotypes in $\boldsymbol{P}_t$;
calculate fitness of phenotypes in $\boldsymbol{P}_t$ using $\rho_T$';
convergence using DNA testing, $h(v, \boldsymbol{P}_t, \boldsymbol{P}_{t-1})$;
t=t+1;                                 /*update the evolutionary counter*/
$v_p = v_p+1$;                 /*update the local evolutionary counter of problem space */
}      /*end of evolution search in problem space*/
}      /*end of the phase one*/

case u=2:
/* Phase 2:determine focus for new design solution, redefine design solution space*/
/*search for best design solution */
$\rho_{t,T} = f(\boldsymbol{P}_t)$;                        /*fitness function*/
$\Omega = \Omega \ \{\rho_{t,T}\}$;                     /*search focus*/
$\boldsymbol{S}_t = g(\boldsymbol{S}_t, \rho_{t,T})$;        /*revised design solution space*/
t=1;                                      /*reset the evolutionary counter*/
While convergency conditions, $c(t,\Omega)$, are not satisfied
{
$\boldsymbol{S}_t$ ::= select genotypes in $\boldsymbol{S}_{t-1}$;
reproduce, crossover and mutate genotypes in $\boldsymbol{S}_t$;
calculate fitness of phenotypes in $\boldsymbol{S}_t$ using $\rho_T$ ;
convergence using DNA testing, $h(v, \boldsymbol{S}_t, \boldsymbol{S}_{t-1})$;
t=t+1;                                 /*update the evolutionary counter*/
$v_s = v_s+1$;                 /*update the local evolutionary counter of problem space */
}      /*end of evolution search in solution space*/
break;                                 /*end of the phase two*/
}      /*end of a coevolutionary cycle*/
}      /*end of coevolutionary design */
```

Figure 6. CoDESIGN: an algorithm for co-evolutionary design.

The algorithm is illustrated in Figure 7 graphically. The concept of a "generation" or "cycle" in co-evolution computing has different meanings from the simple GA since there is more than one population that is evolving. Therefore, we use three counters for time: T, t, v and u. The co-evolutionary cycle counter, T, indicates the change in generation from one full cycle of co-evolution to another. T is updated after the solution space and the problem space have been searched and converged. The generation counter, t, indicates a new generation in the simple GA sense. t is updated after each generation of search in one of the spaces; that is, after a cycle of selection, crossover, and mutation. The local evolutionary counter, v, indicates the number of generations in each search of the problem or solution space. Its value is reset once a full cycle of coevolution is completed so that v is a local counter for evolutionary cycles. The co-evolutionary phase counter, u, indicates the current status of coevolution process. Its value is reset at the beginning of every covolutionary cycle.

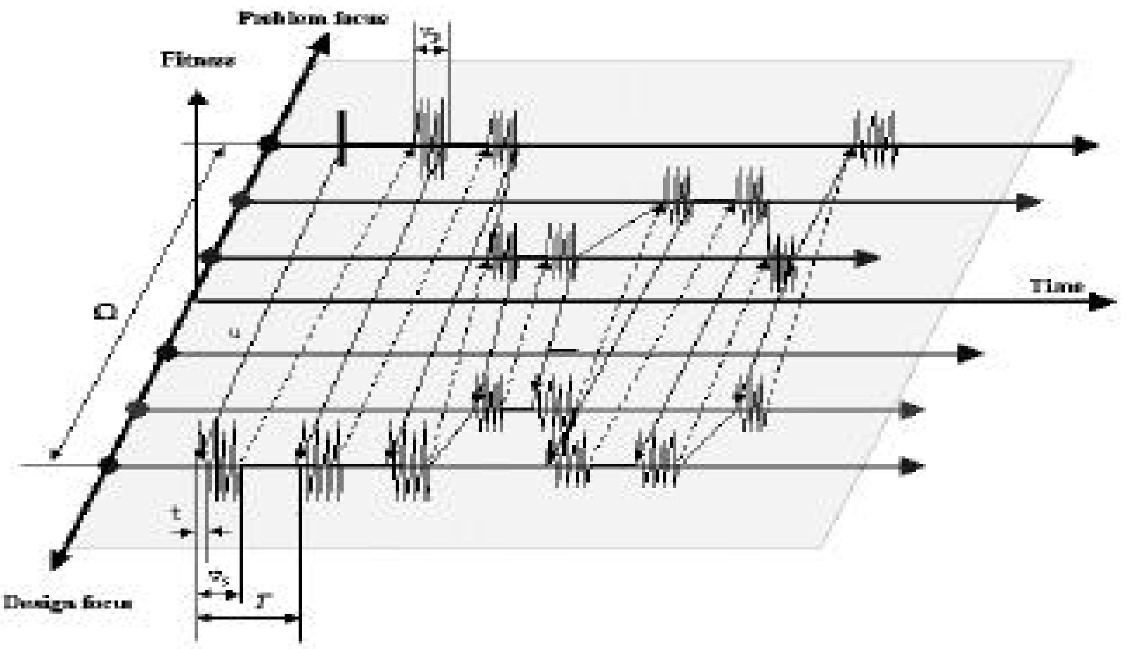


Figure 7. Illustration of the co-evolutionary design algorithm.

The concepts of fitness, convergence and termination are discussed below.

Fitness: Survival of the fittest in evolution has been translated as a fitness function in simple GAs. This fitness function is the basis for the comparison of alternative solutions. In design, when we let the definition of the fitness function change, the value of the fitness function can no longer serve as the basis for comparison for all alternative designs. The performance of individuals in the solution space can only be compared when they are evaluated using the same fitness function. This makes it difficult to compare the performance of solutions across different phases of the co-evolutionary design process. The performance of individuals is used to determine which members of the population "survive", or are selected to participate in the next generation of search. When searching a space, either the problem space ***P*** or the solutions space ***S***, the performance is measured by how well the alternatives satisfy the focus. A focus for the search for a design solution is a function of the set of possible design requirements, and a focus for the design requirements is based on the current set of design solution alternatives. In the CoDESIGN algorithm, we represent the set of possible design requirements in the space ***P*** and the corresponding focus, or fitness function, as ρ. We represent the current set of design solution alternatives in the space S and the corresponding focus as ρ**'**. The fitness function, ρ, is defined as a function of ***P***, or f(***P***). The subscripts of ρ are t and T, where t indicates which generation of ***P*** was used to derive ρ and T indicates which generation the ρ is used. There will be a new ρ for each T, but not for each t, since each phase may result in multiple generations of ***P*** or
but the focus stays the same.

Convergence: Convergence in evolutionary algorithms means that the search process has led to the "best" design in terms of the specified fitness function. Convergence is typically the criteria for termination of the evolutionary search process. Since the fitness function in co-evolutionary design changes from one phase to another, the idea of convergence needs to be reconsidered. This requires a consideration of the purpose of coevolutionary design as compared to evolutionary search. The purpose of evolutionary search is to find the best solution based on a given environment, where the environment is effectively represented by the fitness function. The purpose of coevolutionary design is to explore both the problem and solution spaces, allowing both to change in reaction to each other, until a satisfactory combination of a problem statement and solution state is found. The exploratory nature of the coevolutionary process implies that the process should search until the potential for new ideas is reduced. We propose that convergence is not related to fitness, but to the similarity of the members of the population. A population in which there is little change in the genotypes of the members when compared to the previous population indicates that the search process has converged.

Termination: The link between convergence and termination in evolutionary algorithms occurs because the convergence to the "best" solution indicates that the search should be terminated. In co-evolutionary design, convergence is determined for each phase of the search, that is, for a given focus, and following the convergence for one focus, another focus is determined and search commences in the other space. This indicates a separation of termination and convergence. We use termination to indicate when the co-evolutionary process should stop, and convergence to indicate when the search in a given space for a given focus should stop. One criterion for termination is the number of cycles of the co-evolution process. This is indicated in the algorithm as T. This criterion is equivalent to setting a time limit for the design process. Often, the time limit is a major criterion for signalling when exploration of changes in problem and solution should stop. Another criterion for termination is similar to the convergence criterion above – there are no new fitness functions being found. With each change in phase, the fitness function is appended to a list of fitness functions, labelled Ω in the algorithm. A criterion for termination is when there are no new fitness functions added to Ω. The significance of this criterion is that the algorithm is not able to identify a different focus for the design and therefore, new ideas have been exhausted.

Interaction. Interaction is relevant only in the context of co-evolution, since there are two search spaces. Interaction provides a mechanism for transferring knowledge from one space to another, with the potential to expand the boundaries of a search space during the design process. Interaction between the requirements and solution spaces can be modeled as passing variables from one space to the other.

4. Structural System Layout

The design of a structural system layout is used to demonstrate the model of co-evolutionary design. For co-evolutionary design we have a separate search space for the problem space specification and the solution. The design problem is specified as a set of loads in space and rectangular regions of constrained free space, or space in which structural elements cannot be placed. The design solution is specified by horizontal, vertical, and diagonal elements in a plane. For instance, a horizontal element can represent a beam, a vertical element can represent a column, a diagonal element can represent a bracing member.

Structural system layouts are specified by combining elements. For example, a frame can be represented as a combination of a horizontal and two vertical elements. Elements are connected to each other to form a structural system. Each element has two end-points, as shown in Figure 8.

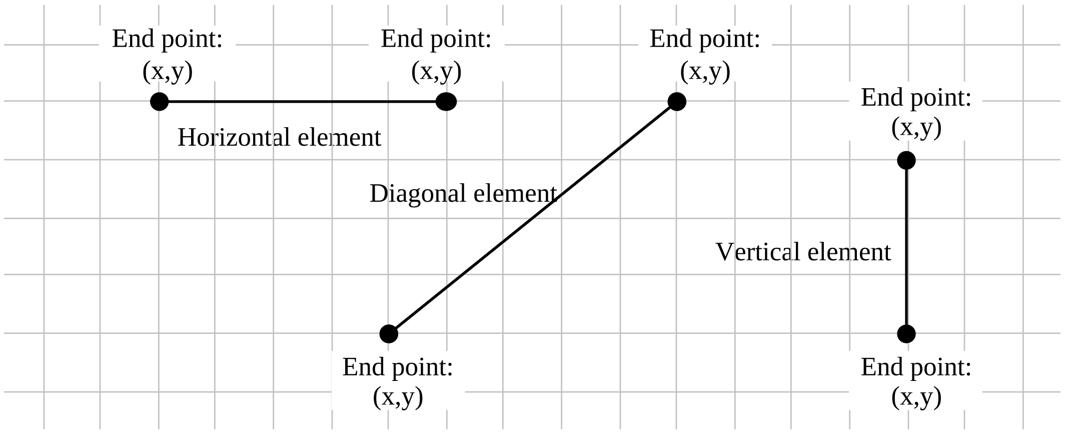


Figure 8. Horizontal, diagonal, and vertical element with end points

It is only possible to have connections to other elements at the first and/or at the second end-point. The connection types are not considered in this implementation. An example of a possible structural system as a solution for a design problem is shown in Figure 9.

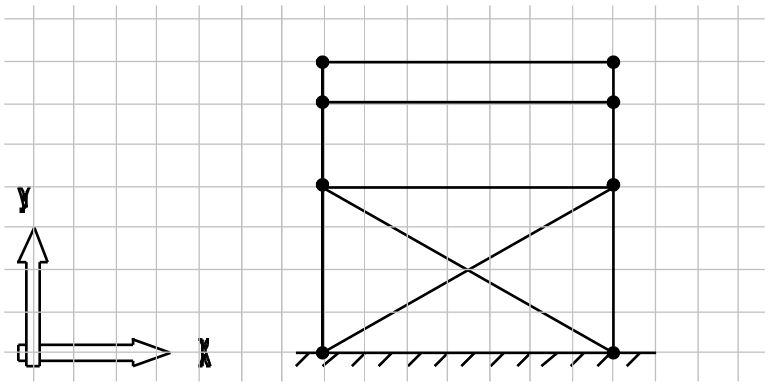


Figure 9. A possible structural system as a solution

Loads are classified into two load types, point loads and distributed loads, for a planar problem. Moreover, a load is described by its magnitude and units, its location, its direction, and its length of distribution (zero for point loads). The direction of all loads in this demonstration is fixed: the loads are perpendicular to the earth surface, pointing downwards. The magnitude and the units of a load are not considered in this implementation.

The constrained free spaces of a problem represent openings, such as doors, windows, tunnels, and required areas for technical equipment like heating or lights. Neither loads nor structural members can be placed within a constrained free space. A constrained free space is described by its geometry (length and height) and its location ((x,y)-coordinates of the lower left point). An example of a problem specification consisting of loads and constrained free spaces is shown in Figure 10.

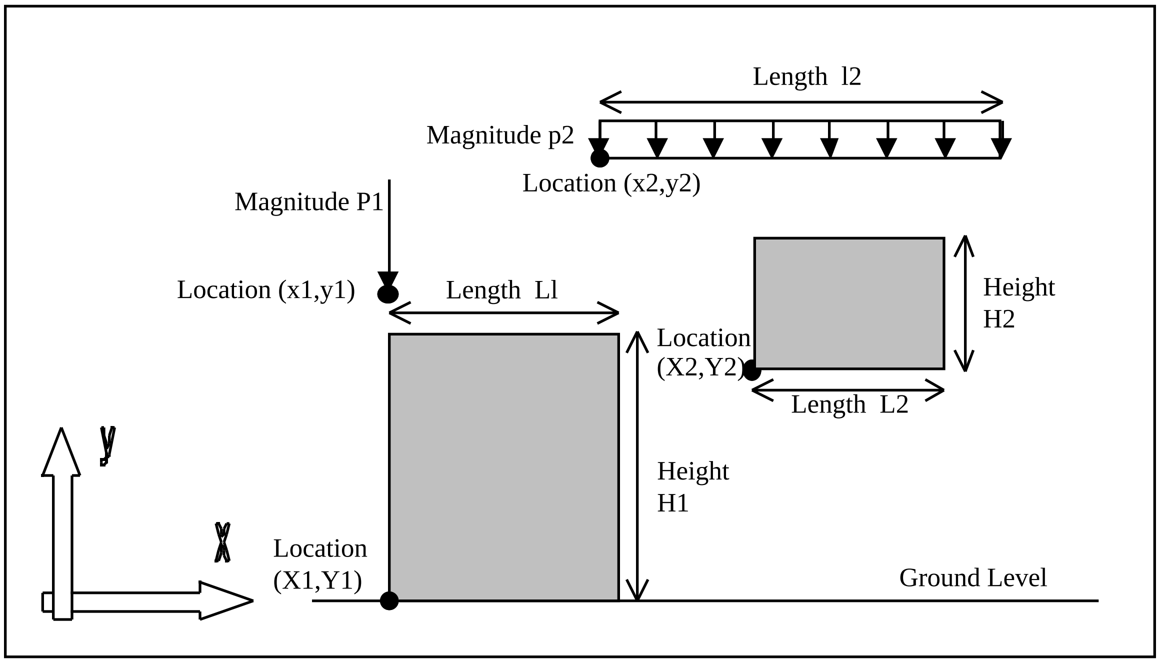


Figure 10. A specification of a structural system layout problem

4.1 PROBLEM SPACE REPRESENTATION

The problem space contains a *population* of individuals, where each individual represents a problem. As shown in Figure 11, each *individual* has a name, a phenotype, a genotype, and (after it has been evaluated) a fitness value. Additional information about its parents and the way it was created is also included. The method of creation can be crossover, mutation, or problem space initialisation.

Individual					
name	created-by	parents	fitness	phenotype	genotype

Figure 11. Individual problem object

The *problem phenotype* of an individual problem is a list of problem phenes. A *problem phene* can represent a load or a constrained free space. It is possible for a phenotype to have either only loads or only constrained free spaces, but tyically loads and constrained free spaces are combined. The phenes within a phenotype do not have to be ordered in any particular way, and there can be any number of phenes within a phenotype. The representation of an example of a phenotype is shown in Figure 12. For this demonstration, the *genotype* of a problem is exactly the same as the phenotype of the same individual.

Problem phenotype					
Load 1	Space 1	Space 2	Load 2	Load 3	. . .

Figure 12. Representation of a possible problem phenotype

A load is represented by its location, and its length of distribution. If the length of distribution is zero the load is a point load, otherwise the load is a distributed load. The content of a load phene structure object is shown in Figure 13. The location describes the grid point where a point load acts or the leftmost grid point of application of a distributed load.

Figure 13. Load phene structure object

The representation of a constrained free space phene includes its location, its length, and its height. The content of a free space phene object is shown in Figure 14.

Constrained Free space phene		
Location : grid point	Length	Height

Figure 14. Constrained free space phene object

4.2 SOLUTION SPACE REPRESENTATION

The solution space also contains a *population* of individuals. The *solution phenotype* of an individual solution is a list of elements containing any number of solution phenes as shown in Figure 15.

Solution phenotype				
Element 1	Element 2	Element 3	Element 4	. . .

Figure 15. Solution phenotype

A *solution phene* represents a horizontal, a vertical, or diagonal design line element. Each element is characterised by the locations of its end-points, using their absolute grid coordinates as shown in Figure 16.

Element phene	
Location of first end point	Location of second end point

Figure 16. Element phene structure object

The type of the element can be determined by the location of both end-points. If the two end-points have the same x-coordinate value, the element is a vertical element, if the two end-points have the same y-coordinate value, the element is a horizontal element, and if the two end-points have different x- and y-coordinate values, the element is a diagonal element. Absolute coordinates are preferred to relative locations because this makes it easier to identify if the problem loads and spaces match the solution design elements, and to compare the locations of design elements with each other when evaluating structural integrity.

The *genotype* of a solution individual is not equivalent to the phenotype of the individual. The solution genotype is a list of points, each of which is a *gene*. The first point is an absolute location on the ground (y=0), and each of the following points is described relative to the previous point. Each pair of adjacent points represents an element which is placed between them. It can happen that an element appears two or more times in a genotype because of the use of relative coordinates, but this repetition can be eliminated during genotype-to-phenotype mapping. Relative grid points are needed to avoid producing a lot of useless or meaningless solutions when the crossover mechanism is applied. The representation of a solution genotype is shown in Figure 17.

Solution genotype				
Absolute grid point	Rel. grid point 1	Rel. grid point 2	Rel. grid point 3	. . .

Figure 17. Representation of a solution genotype

4.3 FITNESS EVALUATION

Fitness functions are needed for the *evaluation* of individuals in both the problem and the solution space. The fitness of a solution is based on how well it solves the current best problem. The fitness of a problem is based on how well the current best solution solves it. The fitness value may be influenced by the internal consistency of an individual, by common sense constraints, and by design principles particular to the domain.

In this application, we define the fitness function as a set of constraints on the combined problem/solution pair. When evaluating the fitness of alternatives in the problem space, we combine each alternative problem individual with the current best solution. When evaluating the fitness of alternatives in the solution space, we combine each alternative solution with the current best problem.

Some constraints are only valid for evaluating problems, some are only used for evaluating solutions, and some are equally applicable to both problems and solutions. The constraints used in this demonstration are listed in Appendix B. An example of a constraint is: “each vertical element must have a support beneath it, either another vertical element or the ground”. Each constraint returns a value between 0 and 1 reflecting the quality of the individual. If the constraint is fully satisfied the returned value is 1, if the constraint is not satisfied at all the returned value is 0, and if the constraint is satisfied by only some of the components of the individual the returned value is between 1 and 0 according to the degree of satisfaction. The degree of satisfaction depends on how many phenes of the individual’s phenotype satisfy the constraint. The *final constraint value* for the phenotype is the average of the constraint values for each phene in the phenotype being evaluated.

The *fitness value* for an individual is the average of the weighted constraint values returned by each of the constraints that is relevant to that individual. The *weight* of a constraint indicates the importance of a constraint. A constraint with a weight of 3 is very important, a constraint with a weight of 2 is still important, and a constraint with weight 1 is less important. Constraints c12 and c13 have a weight of 5 when they are used to evaluate how well a solution fits a problem, but they also serve to evaluate how well a problem fits a solution, and in this situation their weight is 1. The formulas for the fitness values of a solution individual and a problem individual are shown in Figure 18. The higher the fitness value the better the solution individual or the problem individual.

Solution Individual:

$$\text{Fitness value} = \frac{1}{w0+w1+w2+w3+w4+w5+w6+w7+w8+w9+w12+w13+w14} *$$

$$* \left[\frac{\Sigma\ \text{c1-value}}{h} * (w1) + \frac{\Sigma\ \text{c2-value}}{h} * (w2) \right.$$

$$+ \frac{\Sigma\ \text{c3-value}}{v} * (w3) + = \frac{\text{c4-value}}{v} * (w4) + = \frac{\text{c5-value}}{v} * (w5) +$$

$$+ \frac{\Sigma\ \text{c6-value}}{d} * (w6) + \frac{\Sigma\ \text{c7-value}}{d} * (w7) + \frac{\Sigma\ \text{c8-value}}{d} * (w8) +$$

$$+ \frac{\Sigma\ \text{c9-value}}{h+v+d} * (w9) + \frac{\Sigma\ \text{c12-value}}{h+v+d} * (w12) + \frac{\Sigma\ \text{c13-value}}{L} * (w13) +$$

$$\left. \right]$$

Problem Individual:

$$\text{Fitness value} = \frac{1}{w10+w11+w12+w13+w15+w16} *$$

$$* \left[\frac{\Sigma\ \text{c10-value}}{L} * (w10) + \frac{\Sigma\ \text{c11-value}}{L+S} * (w11) + \frac{\Sigma\ \text{c12-value}}{S} * (w12) + \right.$$

$$\left. + \frac{\Sigma\ \text{c13-value}}{L} * (w13) + \frac{\Sigma\ \text{c14-value}}{1} * (w14) + \frac{\Sigma\ \text{c15-value}}{1} * (w15) \right]$$

wi: weight of constraint i
h: number of horizontal elements
v: number of vertical elements
d: number of diagonal elements
L: number of loads
S: number of constrained free spaces

Figure 18. Formulas for evaluating a solution individual and a problem individual

4.4 IMPLEMENTING THE CO-EVOLUTIONARY PROCESS

Initialisation: A single problem individual and a population of solution individuals initialise the problem space and solution space, respectively. In this demonstration, the initial solution space has 20 different solutions. The phenotype of each individual solution represents a possible structural system. Some of the individuals in the initial population of solutions are shown in Appendix A.

Selection: Each GA cycle starts with 20 individual members of the population. The selection of individuals as parents for the crossover operation is determined by the parent selection parameter. In this demonstration we select the parents randomly from the 20 individuals in the population. The population size is controlled by the reproduction percentage and the selection percentage parameters. In this demonstration, the reproduction percentage parameter is 100% and the selection percentage is 50%. These values mean that in each cycle, 20 new individuals are generated through crossover and mutation and 20 of the 40 individuals are selected to be in the next GA cycle.

Convergence: Convergence can be determined genetically, optimally, or by number of generations. Convergence is determined genetically by keeping track of which genotypes have been encountered before and during each GA cycle. The search converges or ends when most genotypes generated in the current cycle have already been encountered. The threshold for ending the search is the repetition percentage parameter. Convergence is determined optimally by ending the search when an individual has a fitness value of 1. Convergence is determined by number of generations when the v parameter reaches a specified value.

Interaction: Interaction in this demonstration occurs after convergence in the solution space. The 20 individuals in the solution space are the basis for contributing alternative problems in the problem space. Each of the best solutions produce one alternative problem. A new problem is defined by a point load for each vertical element, and distributed load for each horizontal element, and a constrained free space for each open space below a horizontal element. This interaction provides a mechanism for the best solutions to influence the problem specification.

Termination: Termination occurs when the co-evolution process ends. Where convergence determines when the search in one space ends so that search in the other space can begin, termination determines when search in both spaces ends. Termination can be determined using optimality criteria, by no change from one phase to another, or by the number of co-GA cycles. In this demonstration, we terminate the co-evolutionary process after a fixed number of co-GA cycles, indicated by the parameter T.

4.5 RESULTS

Run 1: In the first run of the co-evolutionary algorithm, we only let the GA run for one cycle in each space before searching the next space. The parameters for this run are:

Initial solutions: 20
Initial problems: 1
Reproduction percentage: 100%
Selection percentage: 50%
Selection: random
Convergence at v=1
Termination at T=10

The initial problem and the best solution and problem at the end of each evolutionary search is shown in Figure 19. The initial set of requirements and the first solution found are shown at the bottom of the Figure. The coevolutionary process produces a variety of solutions and alternative requirements. The final solution and requirements does not have much in common with the initial. This is an appropriate process, where each phase converges after one generation, when the goal of the design process is to identify a broad range of requirements and solutions.

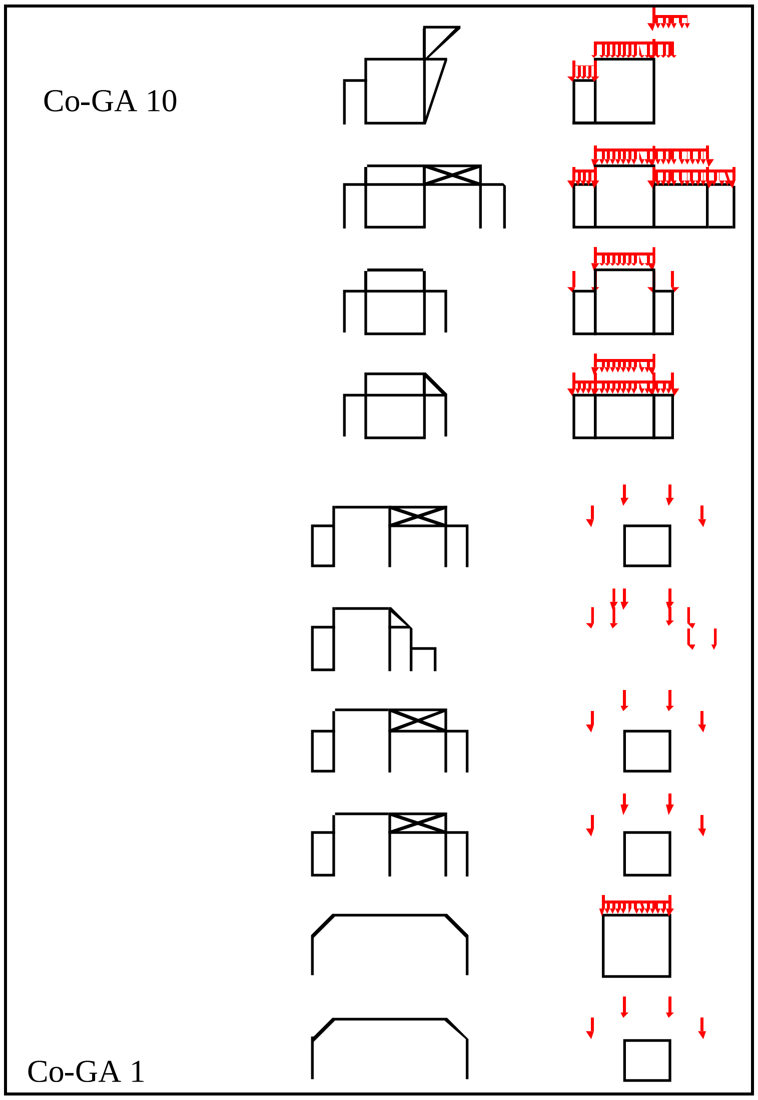


Figure 19. Results of 10 co-evolutionary cycles for run 1.

Run 2: In the second run, we used the same parameters and let the search in each space continue for 20 generations. The parameters for this run are:

Initial solutions: 20
Initial problems: 1
Reproduction percentage: 100%
Selection percentage: 50%
Selection: random
Convergence at v=20
Termination at T=6

The initial problem and the best solution and problem at the end of each co-evolutionary cycle are shown in Figure 20. Again, the initial set of requirements and solution are shown at the bottom of the Figure. The major difference between this run and the previous is the number of generations in each phase, where this run allows the search to continue for 20 generations. The process shows early convergence, although not to the initial requirements. The requirements and solution change between the first and second coevolutionary cycles and then there is little change over the next five cycles.

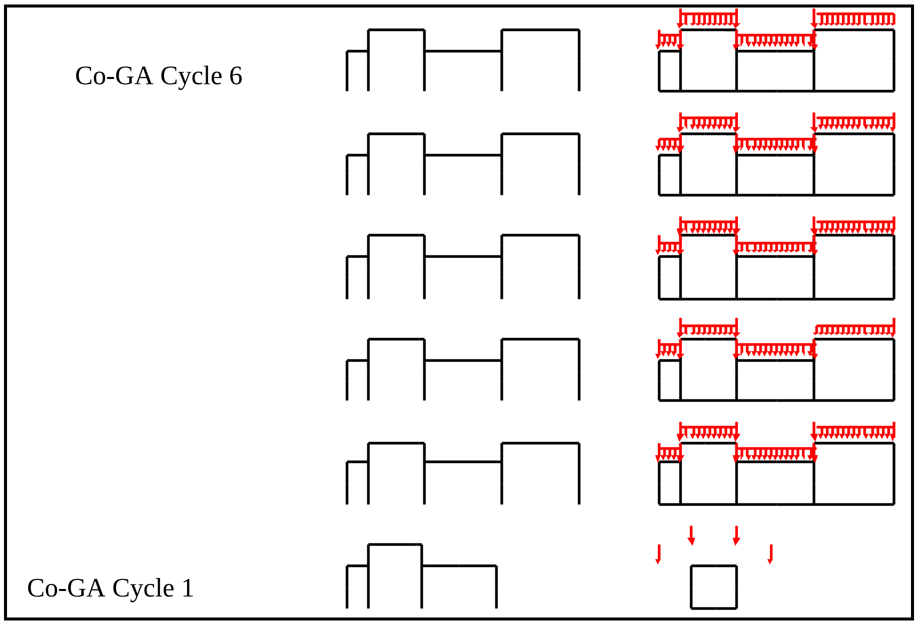


Figure 20. Results of 6 co-evolutionary cycles for run 2.

4.6 DISCUSSION

In this demonstration we were not considering whether the co-evolutionary algorithm found the best solution to the initial problem. We are observing the behaviour of the algorithm to determine whether alternative problem specifications are formulated and if the search behaviour can be loosely described as a search in two spaces in parallel with some interaction between the spaces. The results from these two runs show that a co-evolutionary model of design does allow the problem specification to change in response to the solutions. In both runs, the problem continues to change to better fit the current best solution. In the first run there is more exploration of alternative solutions because the GA did not optimise the solution space. In the second run, with 20 GA cycles in each space before searching the other space, the solution and problem specification do not change after the first cycle.

5. Summary

This paper formalises the model of co-evolutionary design using the terminology of search and exploration. Traditional evolutionary computing can be considered as a case of coevolution. Coevolutionary design has the following features:

- Co-evolutionary design consists of two spaces/populations: design specifications and design solutions.
- Co-evolutionary design consists of two iterative phases: search for design specifications and search for design solutions.
- The focus of search in one space is determined by the current population in the other space.
- Interactions may add new variables to either space, which may lead to unexpected design requirements or solutions.
- Genetic changes in co-evolutionary design consist of two aspects: changes in design focus and changes in genotype.
- Fitness is local and changes in each phase.
- The fitness value is not comparable across different phases.

- There is no relationship between convergence and fitness: fitness is used to determine which individuals survive and convergence occurs when new ideas can not be found.
- The termination conditions do not rely on the fitness of individuals.

The model of evolutionary design has been used to explain real design projects such as the Sydney Opera House [Maher and Poon, 1996] and as the basis for developing a computational model of the design of braced frames [Maher, et al, 1995] and floor plans [Poon and Maher, 1997].

Acknowledgments
This work is supported by a grant from the Australian Research Council. Peter Wu contributed to the development of the general CoDESIGN algorithm. The development of the structural system layout problem was done with the collaboration of Anja Wetzel and Andres Gomez de Silva Garza. Andres implemented and executed the co-evolutionary algorithm and produced the results.

References

Hillis, D. W.: 1991, Co-evolving parasites improve simulated evolution as an optimization procedure. In: Langton, C., C. Taylor, J. D. Farmer, & S. Rasmussen (eds) Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, vol. X, Addison-Wesley, pp 313-324.

Gero, J. S.: 1992, *Creativity, emergence and evolution in design* in Gero J. and F. Sudweeks (eds) Preprints Computational Models of Creative Design, University of Sydney, pp 1-28.

Maher, M.L.: 1994, Creative Design Using A Genetic Algorithm, *Computing in Civil Engineering*, ASCE, pp 2014-2021.

Maher, M.L., Poon, J., Boulanger, S.:1995, Formalising design exploration as co-evolution: A combined gene approach, in J.S.Gero and F. Sudweeks (eds), *Advances in Formal Design Methods for CAD*, Chapman & Hall, pp 1-28.

Maher, M.L. and Poon, J.:1996, Modelling design exploration as co-evolution, *Microcomputers in Civil Engineering*, 11: pp 195-210.

Mendelson,:1963,: *Introduction to Topology*, Blackie & Son Limited.

Paredis J.: 1995, *Articial Coevolution, Explorations in Artificial Life*, AI Expert Presents, Miller Freeman Inc

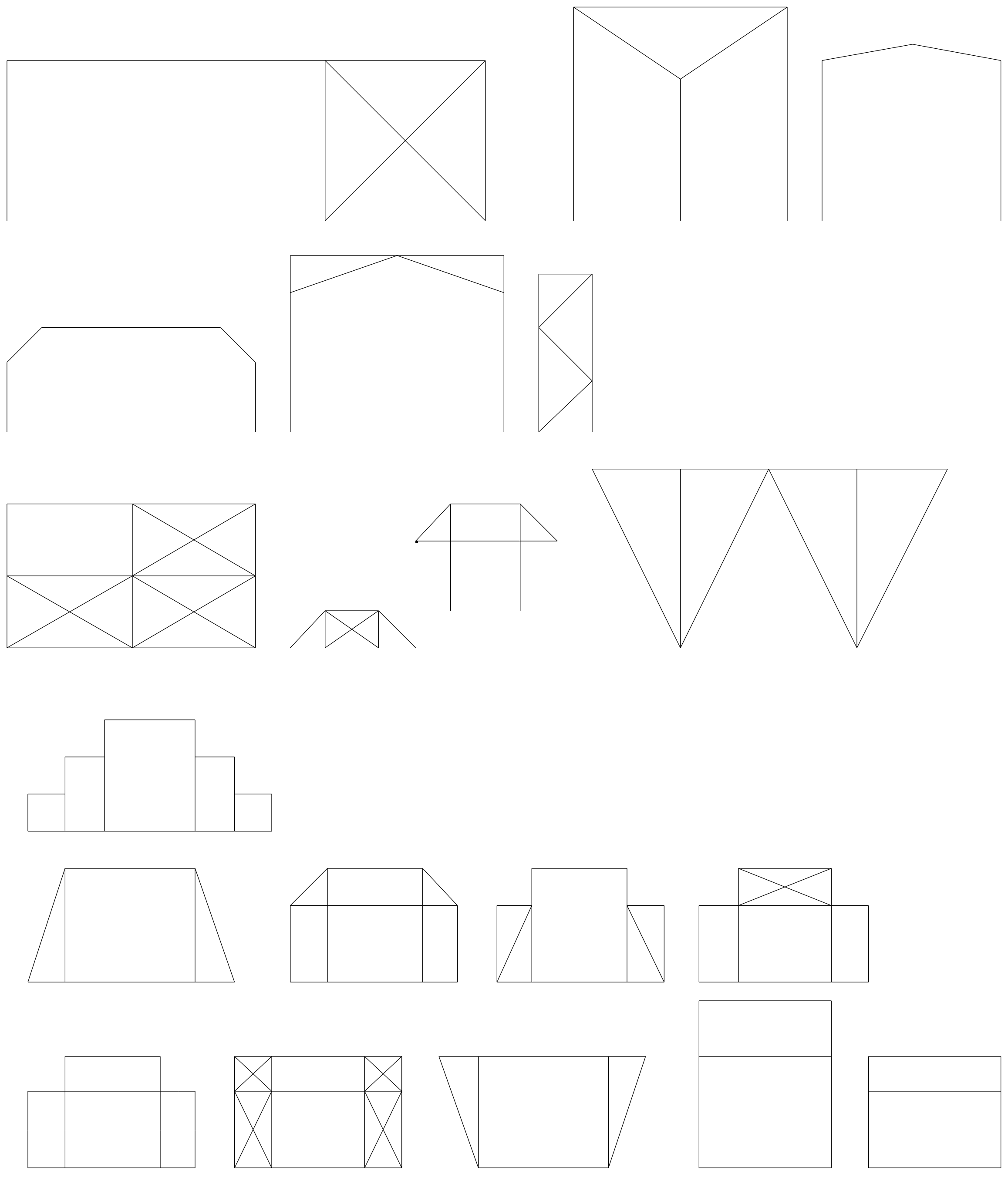
Richard N. L.,; 1995, *The Coevolution of Technology and Organization in the Transition to the Factory System*, Report-no: 95-153, Department of Economics, University of Connecticut, 1995

Simon, H.A.,:1969, *The Sciences of the Artificial*, MIT Press.

Smith M.J.: 1989, Evolutionary Genetics, Oxford University Press.

Appendix A. Examples of Design Solutions in the Initial Solution Space

The initial solution space was defined by manually creating a range of structural system layouts. These layouts are intended to provide a broad range of layout types as the basis for a search space. As a demonstration of co-evolution, this initial space is sufficient. A more detailed analysis of the search space and the boundaries implied by the initial population would be necessary for a full scale implementation of co-evolutionary design.



Appendix B Constraints for Structural System Layout Problem

B.1 Constraints applicable to solution evaluation:

Constraint c1 (weight:2): For economic reasons **horizontal members** should never be longer than 20m. Examples are shown in Figure B.1.

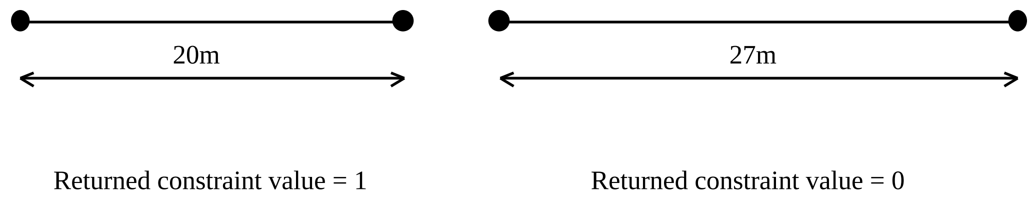


Figure B.1 Examples for constraint c1

Constraint c2 (weight:3): Every **horizontal element** needs another element or other elements as support. Most of the time a horizontal element needs to have two supports, one at each end. Cantilevers are allowed for horizontal elements with a length equal to or less than 2m. These elements need only one support at either one of the two ends. Supporting elements can be vertical or diagonal elements. It is not relevant whether the supporting element is under or above the horizontal element. Examples are shown in Figure B.2.

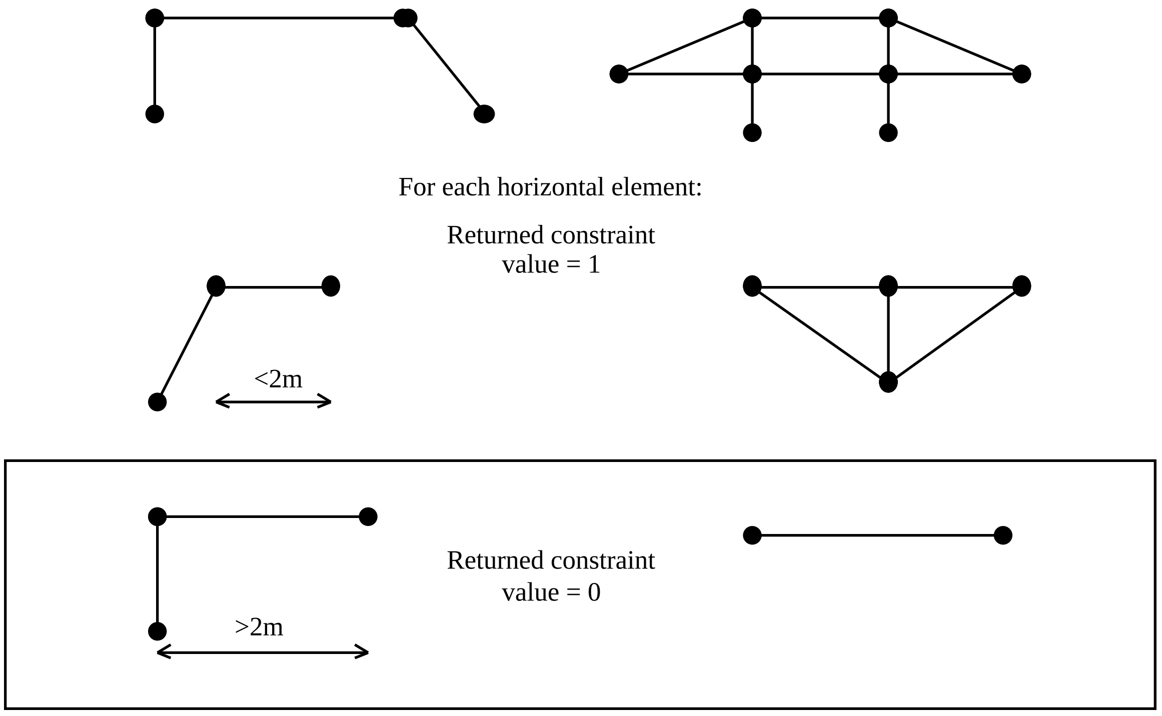

For each horizontal element:
Returned constraint value = 1
<2m
Returned constraint value = 0
>2m

Figure B.2 Examples for constraint c2

Constraint c3 (weight:2): A **vertical element** should not be bigger than 12m. Examples for constraint c3 are shown in Figure B.3.

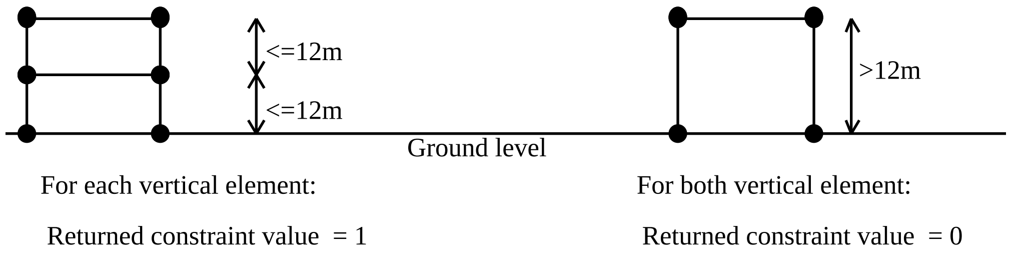

<=12m
<=12m
>12m
Ground level
For each vertical element:
Returned constraint value = 1
For both vertical element:
Returned constraint value = 0

Figure B.3 Examples for constraint c3

Constraint c4 (weight:3): Every **vertical element** needs a support beneath it. The support can be another vertical element or the ground. Examples for constraint c4 are shown in Figure B.4.

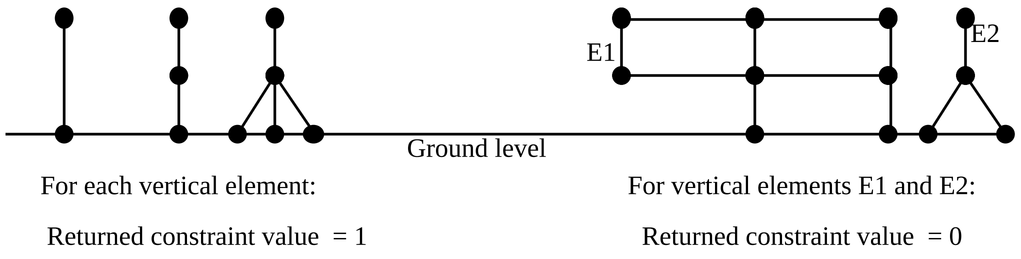

E1
E2
Ground level
For each vertical element:
Returned constraint value = 1
For vertical elements E1 and E2:
Returned constraint value = 0

Figure B.4 Examples for constraint c4

Constraint c5 (weight:3): Every **vertical element** more than 3m long needs lateral restraints to increase the possible buckling load and to prevent collapsing of the structure. Constraint c5 checks if the top end of each vertical element is connected to a horizontal element or a diagonal element. This is necessary for the vertical element to be laterally restrained, though not sufficient (other constraints help identify other necessary conditions for the existence of lateral restraints). Examples are shown in Figure B.5.

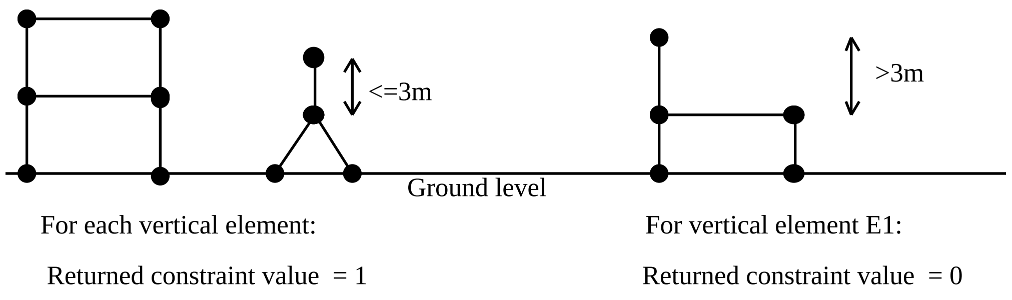

<=3m
>3m
Ground level
For each vertical element:
Returned constraint value = 1
For vertical element E1:
Returned constraint value = 0

Figure B.5 Examples for constraint c5

Constraint c6 (weight:3): . We don't want to allow diagonal cantilevers in our current implementation. Therefore, every **diagonal element** needs to be connected to at least one vertical, one horizontal element, or another diagonal element at both ends. Examples for constraint c6 are shown in Figure B.6.

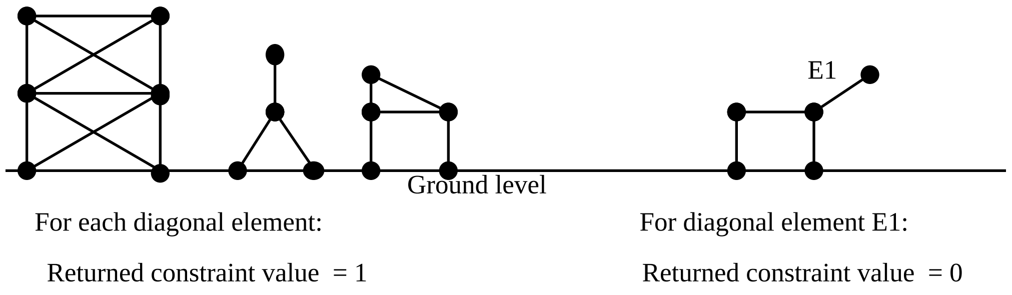

E1
Ground level
For each diagonal element:
Returned constraint value = 1
For diagonal element E1:
Returned constraint value = 0

Figure B.6 Examples for constraint c6

Constraint c7 (weight:2): Diagonal elements should not be too long for economical reasons. We restrict the length of diagonal members to 12m in our implementation. Especially for compression members the length of a diagonal element is critical, since buckling problems can occur. The structural analysis will show later on if the design elements are possible for specific load combinations. Examples for constraint c7 are shown in Figure B.7.

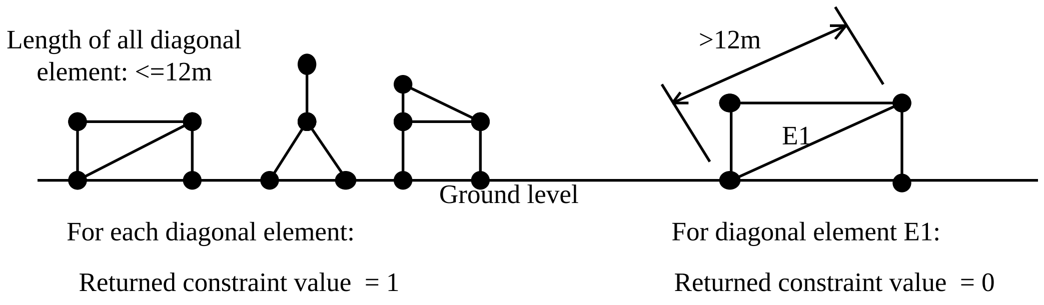

Length of all diagonal element: <=12m
>12m
E1
Ground level
For each diagonal element:
Returned constraint value = 1
For diagonal element E1:
Returned constraint value = 0

Figure B.7 Examples for constraint c7

Constraint c8 (weight:3):. None of the **design elements** should be under the ground or outside the design space area. The design space area has a length of 50m and a height of 50m in our current implementation, which means higher or longer buildings are not possible in our current implementation. Examples for constraint c9 are shown in Figure B.8.

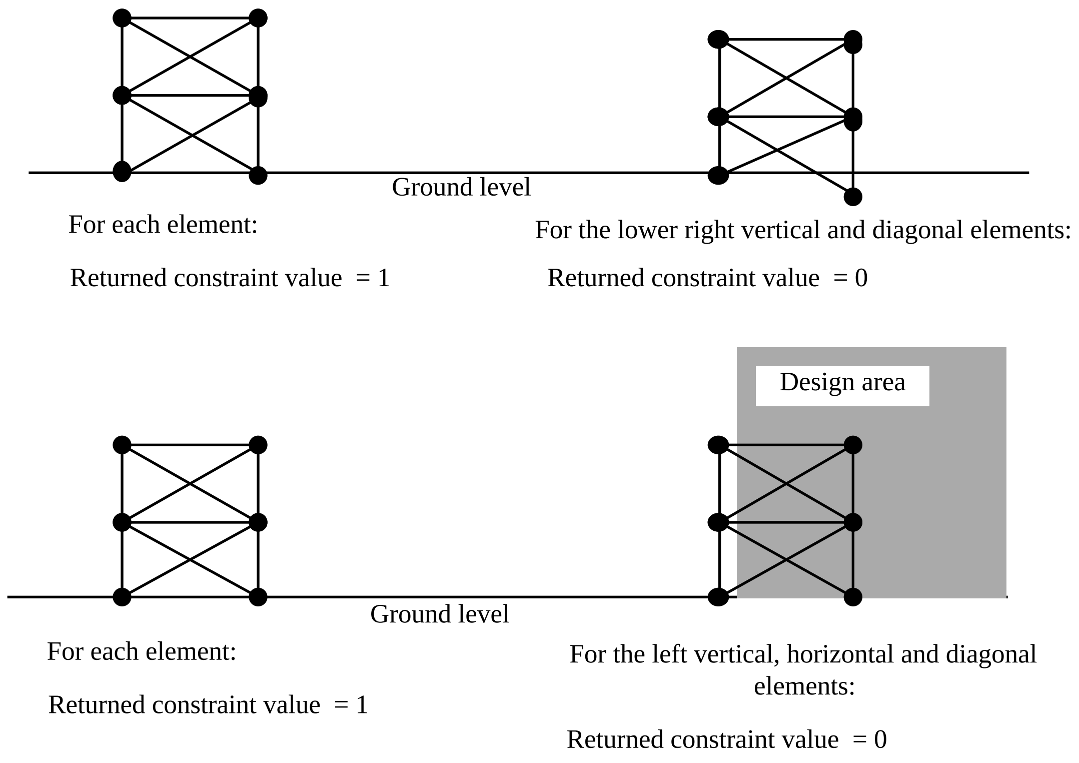

Ground level
For each element:
Returned constraint value = 1
For the lower right vertical and diagonal elements:
Returned constraint value = 0
Design area
Ground level
For each element:
Returned constraint value = 1
For the left vertical, horizontal and diagonal elements:
Returned constraint value = 0

Figure B.8 Examples for constraint c8

Constraint c9 (weight:3):. The number of **design elements** in a structural system has to be limited. For our current implementation the maximum number of allowed elements is 15. The constraint value for c14 of a structure is 1 if the number of design elements is less than or equal to 15; otherwise, it is 0.

B.2 Constraints applicable to evaluating problems

Constraint c10 (weight:3): Loads should never be within constrained free spaces. Examples for constraint c10 are shown in Figure B.10.

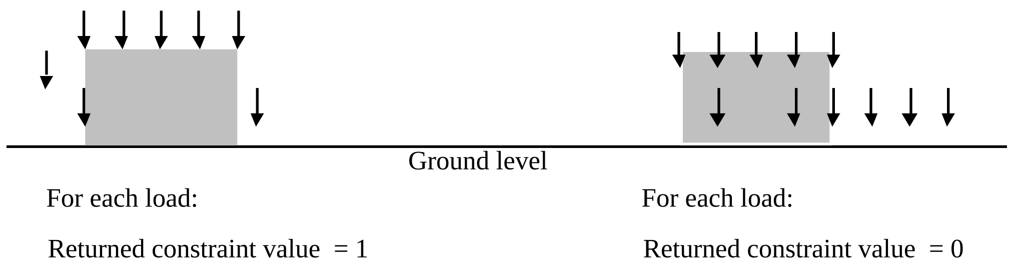


Figure B.10 Examples for constraint c10

Constraint c11 (weight:3):. None of the **loads and constrained free spaces** should be under the ground or outside the design space area. The design space area has a length of 50m and a height of 50m in our current implementation. Examples for constraint c11 are shown in Figure B.11.

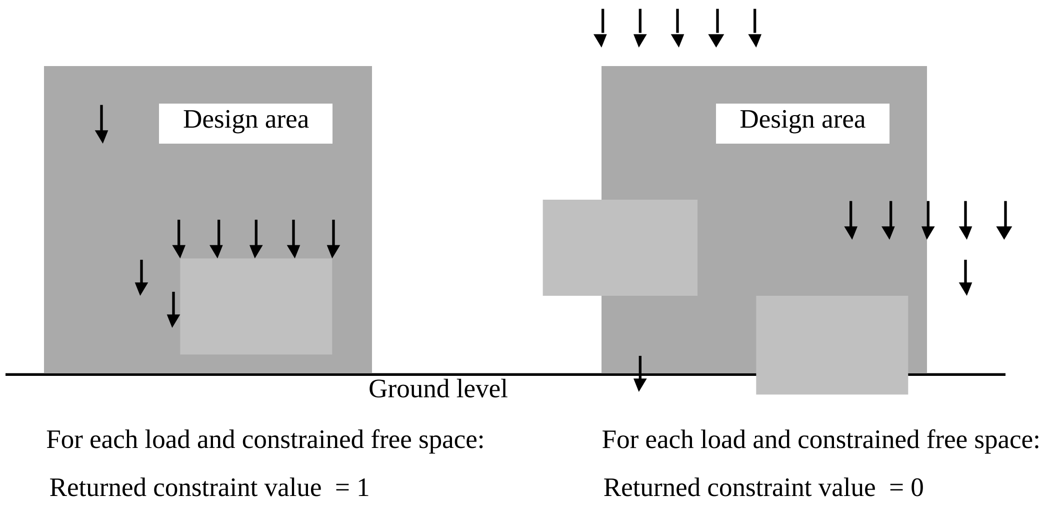

Design area
Design area
Ground level
For each load and constrained free space:
Returned constraint value = 1
For each load and constrained free space:
Returned constraint value = 0

Figure B.11 Examples for constraint c11

Constraint c14 (weight:3): The new evolved problems should be similar to the initial problem. If there is any match between the problem being evaluated and the initial problem the constraint value for c15 will be 1; otherwise, it will be 0.

Constraint c15 (weight:3): To prevent the problem evolving to a reduced problem with less of the original requirements the number of loads and constrained free spaces of the problem being evaluated has to be compared to the number of loads and constrained free spaces of the initial problem. If the number of loads is greater than the numbers of initial loads minus 2, and the number of constrained free spaces is greater than the number of initial spaces minus 2, the constraint value of the problem being evaluated for c16 will be 1. If only one of these conditions is satisfied the constraint value will be 0.5, and if none of the conditions is satisfied the constraint value will be 0.

B.3 Constraints applicable to evaluating problems and solutions

Constraint c12 (weight:5 for solution evaluation and weight:1 for problem evaluation): Any design element must not be in a constrained free space, since by definition these spaces need to be empty to be used for other purposes. If this constraint is being used to evaluate a solution phenotype, each design element in the phenes is checked to see whether it intrudes into a constrained free space from the problem or not. The final constraint value is the total number of non-intruding design elements divided by the total number of design elements in the solution. If the constraint is being used to evaluate a problem phenotype, every constrained free space in its phenes is checked to see whether a design element from the solution intrudes in it or not. The final constraint value is the number of non-intruded constrained free space divided by the total number of constrained free spaces in the problem. Examples for constraint c12 are shown in Figure B.12.

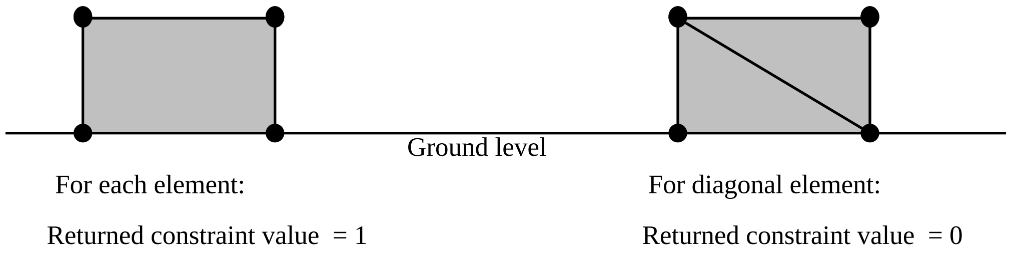


Figure B.12 Examples for constraint c12

Constraint c13 (weight:5 for solution evaluation and weight:1 for problem evaluation): Every **load** needs to be applied to a structural element. Since diagonal members should transfer mainly axial loads, we do not allow diagonal members as direct supports for loads. Distributed loads can only be transferred to horizontal elements whereas point loads can be applied to both vertical and horizontal members. Examples for constraint c13 are shown in Figure B.13.

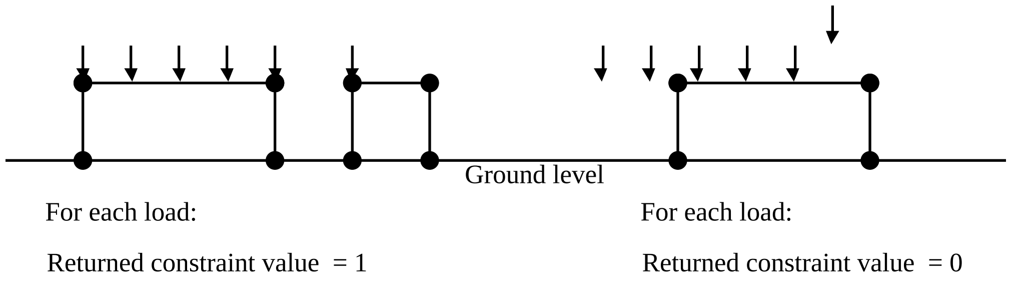


Figure B.13 Examples for constraint c13

B.4 Ranking constraints for Solutions

Ranking Constraint R1 (weight:3): The sum of the lengths of all **elements** in a solution should be minimised. An example for constraint R1 is shown in Figure B.R1.

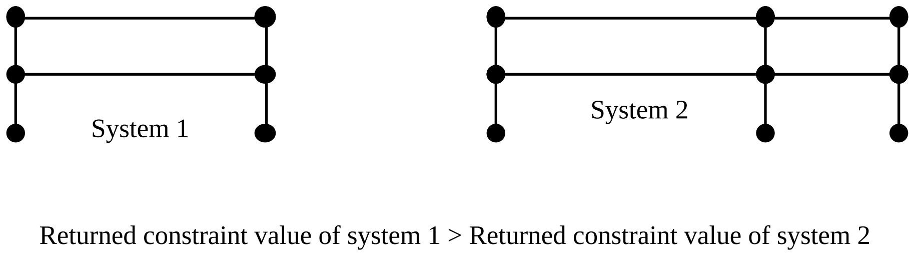

System 1
System 2
Returned constraint value of system 1 > Returned constraint value of system 2

Figure B.R1 Examples for constraint R1

Ranking Constraint R2 (weight:3): The sum of the number of all **elements** in a solutions should be minimised.

B.4 Ranking constraints for Problem

Ranking Constraint R3 (weight:3): The total number of constrained free spaces should be maximised.

Ranking Constraint R4 (weight:3): The total number of loads should be maximised.