J. S. Gero (ed.), *Artificial Intelligence in Design'00,* 393-412.

A PROCESS MODEL FOR EVOLUTIONARY DESIGN CASE ADAPTATION

ANDRES GÓMEZ DE SILVA GARZA AND MARY LOU MAHER
Key Centre of Design Computing and Cognition
University of Sydney

Abstract. One of the unresolved issues in case-based design is how to perform design case adaptation when the constraints are not numerical. In this paper we present a method of case adaptation that employs an evolutionary algorithm. We present a process model for evolutionary design case adaptation, as well as the knowledge it requires and its implications. This model has been implemented for floor plan layout using a case-base of Frank Lloyd Wright prairie houses and an evaluation using feng shui constraints. The implementation demonstrates the representation issues and shows how the model can address two distinct knowledge sources.

1. Introduction

The design process model presented in this paper addresses design synthesis and evaluation by combining the paradigms of case-based reasoning (CBR, see for example (Kolodner, 1993)) and evolutionary or genetic algorithms (GA's, see for example (Mitchell, 1998)). The process model, illustrated in Figure 1, emphasises solving problems using inspiration from precedents; this is the main idea behind CBR. The solutions to past design problems contained in the precedents that are retrieved serve as starting points for a search for design solutions. Multiple random combinations and modifications (together referred to as adaptations) of the retrieved cases are then generated and evolved incrementally, until a satisfactory solution to the new design problem is found; this is the main idea behind GA's.

In this paper we present the process model in general terms and as applied to floor plan layout. The model can be used to perform a variety of design tasks on a broad set of application domains, so far it has been implemented for the floor plan layout and structural design domains

(Gómez de Silva Garza and Maher, 1999a), (Gómez de Silva Garza and Maher, 1999b), and (Gómez de Silva Garza, 2000).

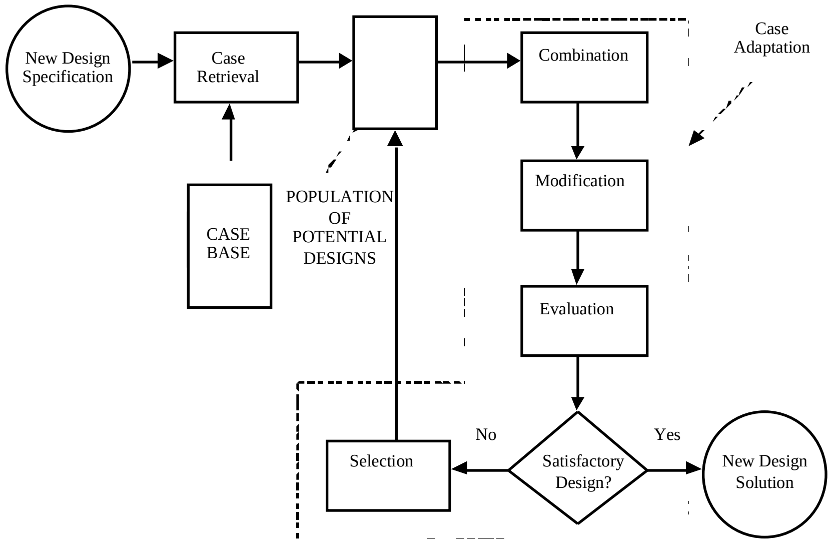


Figure 1. Design process model using CBR and evolution.

2. Overview of the Process Model

Given the specification of the requirements of a new design problem, the first task in the process model is to determine which design cases contain information that might be useful in solving the new problem. This is done by comparing the description of the new problem with the descriptions of the designs stored in the case base. Those cases for which some similarity is found with the new problem are taken to be first approximations towards a solution to the new problem, and are an initial population of potential designs. The cases in this population are then adapted to become satisfactory solutions to the new problem.

Case adaptation is performed by an evolutionary method. Combination and modification are the two types of adaptation that are performed by the evolutionary method through the genetic operators of crossover and mutation, respectively. Crossover of two designs produces two offspring designs, each of which combines features from each of two parent designs. Mutation produces one offspring design which is an altered version of one parent design. Both crossover and mutation insert new designs into the population.

The two types of adaptation generate new potential designs that are evaluated and ranked according to a fitness function. In the process of evaluating them, a potential design may be found that satisfies the requirements of the new design problem and doesn't violate any design constraints. If a solution is not found, the evolutionary adaptation process continues, using the best potential designs as the population to the next cycle of adaptations. The best potential designs are selected from the augmented population containing both previously and newly generated potential designs.

3. Design Case Representation

Case representation issues for this design process model include the representation of a design case in terms of content, as well as the consideration of the genotype vs phenotype representation used in a genetic algorithm.

3.1 CONTENT OF A CASE

The content of a design case can take many forms and can include different categories of design information taken from the precedent it represents, for example, a design case can include:

- the problem specification of the design episode,
- the problem-solving steps that were taken to reach the final design,
- annotations, explanations or justifications of aspects of the final solution, and/or
- annotations, explanations or justifications of the problem-solving steps that were taken.

In addition, if CBR is being used for design, the representation of the final design can include more than the design specifications, such as contextual or support information. For instance, the representation of the solution might also include a description of the environment in which the designed artefact is meant to operate in or other related information that might be of use or might influence design decisions.

The content of a design case determines what knowledge can be transferred from the previous designs to the new design situation. In our example of floor plan layout, we have developed a case base of residential designs that capture a particular style, specifically, Frank Lloyd Wright prairie houses. The content of the case is also influenced by the constraints that are used to evaluate the alternative designs. We have developed a fitness function based on feng shui constraints to demonstrate the use of the model for evaluating designs with non-numerical constraints. The selection of two distinct sources for the case base and the evaluation function also demonstrates that the model can combine different knowledge sources in one design model.

A case is represented at three levels, as shown in Figure 2. In order to represent the layout with enough information to determine whether the feng shui constraints are satisfied, each case includes a description of the landscape surrounding the residence, the layout of the rooms in a floor plan, and the layout of the furniture in the rooms. We identify the components in each level of the case representation and locate them in a simple 3x3 grid. A more detailed analysis of the domain and implementation are included in (Gómez de Silva Garza, 2000).

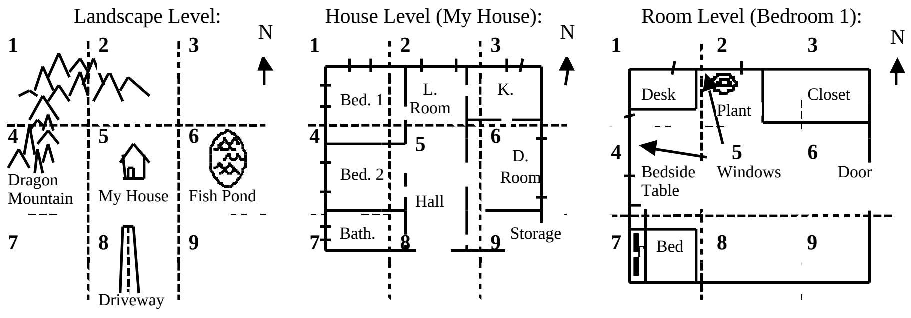


Figure 2. Example of three-level representation of a case

Not all the knowledge that is stored in a case is relevant to the case adaptation process. Only the description of the solution itself, without any contextual information, is used during case adaptation, while the supporting information is used during case retrieval. For example, when adapting floor plan layouts, we may not need the information about the landscape or the arrangement of the furniture within its rooms. The landscape- and room-level information in the case can play a role in matching (during retrieval), but it becomes contextual information when it comes to adapting the floor plan layout of the house.

Implicit in the process model, therefore, is that some preparation of the cases retrieved from memory is done before they can become the initial population of the evolutionary case adaptation method. Figure 3 shows this expanded view of the case retrieval task from the process model. The preparation phase involves stripping away irrelevant information, such as information about the context that is fixed in the new design description and therefore should not be adapted, or re-representing some of the information stored in the case for more efficient use in adaptation. Since our process model is general and makes no commitments about the content or representation of cases, this preparation phase is considered separately for each application of the model.

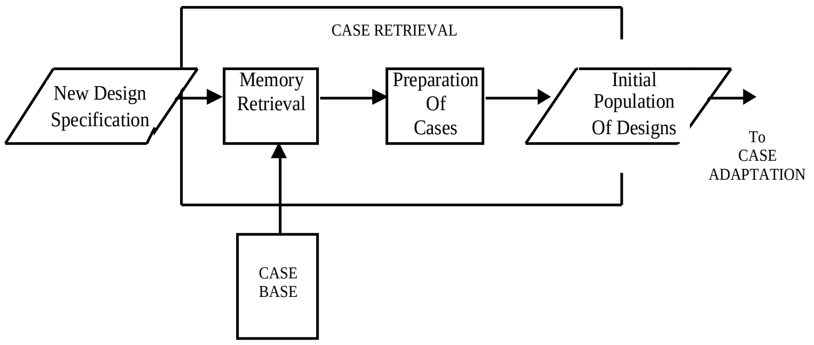


Figure 3. An expanded view of case retrieval

In the floor plan layout example, the preparation phase takes the case as shown in Figure 2 and returns a representation of the information in the house-level. This is the part of the case that becomes the input to case adaptation. The aspects of the design that are adapted are the components of the layout and their connections. A summary of the information in a case that is adapted is shown below.

```
House level
       Components:
              Bed-1 Type: bedroom Location: 1 Shape: square
              Bed-2 Type: bedroom Location: 4 Shape rectangle
              Hall-1 Type: hallway Location: (5 8)
              …
       Connectors:
              Bed-2-Hall-1 Type: door Location: 5 Side-a: Bed-2
                     Side-b Hall-1 Direction EW
              Bed-1-Window-1 Type: window Location: 1 Side-a:
                     Bed-2 Side-b: Hall-1 Direction EW
```

3.2 PHENOTYPE-GENOTYPE DICHOTOMY IN GA'S AND ITS RELATION TO CASES

In GA's a distinction is made between the phenotype and the genotype of an individual in an evolving population. The genotype is the genetic material that describes the individual using a set of primitive genetic instructions/units. The phenotype is the manifestation of the physical, behavioural, and functional characteristics displayed by the individual after the genetic instructions in the genotype are interpreted. The synthesis subtasks of a GA, crossover and mutation, operate on genotypes: it is genetic material that gets transformed by these operations. The analytical subtasks of a GA, evaluation and selection, operate on phenotypes. In some applications of a GA, the distinction between genotype and phenotype is not maintained, however we have kept the distinction in our applications of our design process model. The benefit to the duality is realised when the operations on the genotype representation manipulate information that is independent of the knowledge needed for evaluation, allowing the generation of new designs to be independent of the knowledge of a particular design's performance.

In the floor plan layout application, the case representation contains information about both elements and connectors. The two types of information are interrelated, because the representation of each connector includes information about the elements it connects. Because of the multiple mutual dependencies between the two types of information, crossover (which operates on independent genes) cannot be performed easily on the symbolic case (or phenotype) representation. We have developed a genotype representation of the layout design based on the adjacency matrix.

If different representations are chosen for phenotypes and genotypes in a given implementation, the initial population of designs retrieved from case memory are transformed into the genotypes used during crossover and mutation. The new population of genotypes produced after crossover and mutation are then transformed into a population of phenotypes before evaluation and selection can be performed. These transformations are shown in Figure 4.

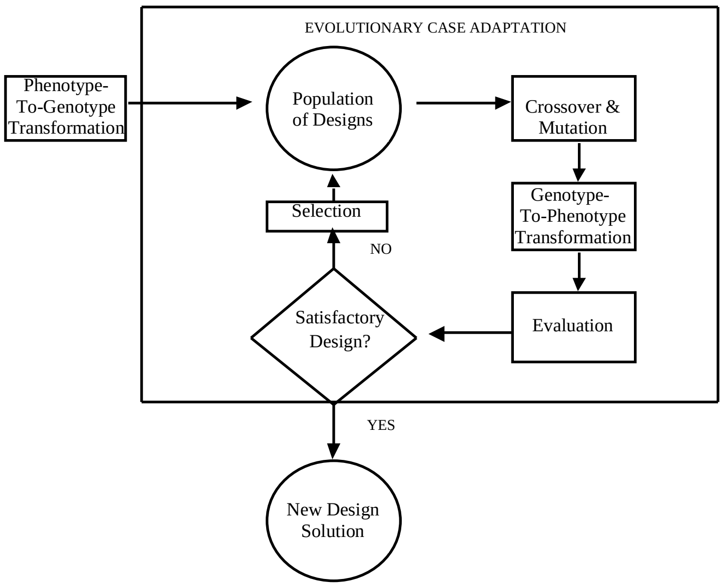


Figure 4. An expanded view of evolutionary case adaptation

4. Evolutionary Design Case Adaptation

Generating a new design d^{new} using evolutionary design case adaptation can be characterised by:

$d^{new} = {}_{design}(R,M,K)$, where

- · design is a transformation process
- · R is a set of problem requirements {r1, r2, …, rρ} that together define a new design problem
- · M is a case memory consisting of a set of cases {c1, c2, …, cμ}
- · • K is a set of domain constraints {k1, k2, …, kκ }.

The three major components of the process model are: a new design problem, a case memory of precedent designs, and the domain constraints that guide the evolution of new designs. These three components allow the separation of the precedent designs (using knowledge of the past) from the current domain constraints (using current and possibly changing design knowledge). The transformation ${}_{design}$ consists of two parts, the case retrieval part ${}_{retrieval}$ and the case adaptation part ${}_{adaptation}$.

4.1 CASE RETRIEVAL

The role of case retrieval is to select a subset of cases from memory that is relevant to a new design problem. The selected cases become an initial population $P^{initial}$ of individual designs that can then be adapted using the evolutionary method. That is:

$P^{initial} = {}_{retrieval}(R,M)$.

As we have seen previously, ${}_{retrieval}$ consists of two steps, a memory access and retrieval step ${}_{mem\text{-}retrieval}$ followed by a case preparation step ${}_{preparation}$. The memory retrieval step can be characterised as follows:

$C = \{c_1, c_2, \ldots, c_\gamma\} = {}_{mem\text{-}retrieval}(R,M)$, where

- · $C \;\; M$ (therefore $\gamma \leq \mu$ and $c_i \;\; C, c_i \;\; M$), and
- · $c_i \;\; M, c_i \;\; C \leftrightarrow \;\; r_j \;\; R½f_{match}(c_i,r_j)$=true.

That is, C is a subset of M containing γ of the cases in M, and a case c_i is retrieved from the case memory to become an element of C if and only if there is some problem requirement r_j in R that matches the case c_i. The specifics of the matching function f_{match} will vary depending on the characteristics of the domain and of the representation chosen for a given domain. Different application domains and different representations for cases and problem requirements may impose the need for different types of operations to be performed in order to compare the features of a given case with a given problem requirement to determine whether they match or not.

The set of retrieved cases C now has to be transformed into an initial population of individual designs that can be manipulated by an evolutionary case adaptation algorithm. This transformation, the case preparation step, can be characterised as:

$P^{initial} = \{d_1, d_2, \ldots, d_\gamma\} = \quad_{preparation}(C)$, where

· i½$i=1, 2, \ldots, \gamma$, we have that $c_i \quad C$ and $d_i = f_{prepare}(c_i)$.

That is, each individual design d_i in the initial population is obtained by applying the preparation function $f_{prepare}$ to the corresponding case c_i from C. The specifics of $f_{prepare}$ will vary depending on the characteristics of the domain and of the representations chosen, but in general it may consist of stripping away irrelevant information (not required for adaptation) from the case description and/or re-representing the information in the case in some way.

In our layout design example, we have represented 12 cases that are Prairie House designs created by Frank Lloyd Wright, obtained from (Hildebrand, 1991). These 12 cases comprise M in the above characterisation. A design problem is given by specifying how many bedrooms, bathrooms, etc. are required in a solution. These requirements are R in the above characterisation. Matching during retrieval (i.e., the function f_{match}) involves determining which cases have (as a minimum) the required number of bedrooms and/or the required number of bathrooms, etc.

We made some assumptions when implementing the process model. First, we assume that there are many cases in the case base and that they provide a reasonable distribution of design features. Second, we assume that, given the specified requirements of a new design problem, more than one case will be retrieved from memory. Third, a case may provide useful information (and therefore will be retrieved) if it matches any part of the new problem requirements. Thus, partial (and in the case of numeric values, approximate) matching is performed, and even if only one of the requirements of a design problem matches one of the features of a given case, the case will be retrieved.

The final assumption is that the cases retrieved from memory will all have the same potential for contributing information towards the final solution. Thus, no ranking or weighting procedure is performed on the retrieved cases, and all of them (rather than only one or a subset of them) are used to create the initial population of the evolutionary case adaptation method. The constraints in the fitness function may bias the usefulness of the cases, but the cases themselves are not modified to respond to this bias.

The assumptions discussed above are valid for the following reasons. In design, it is unlikely that any previously designed solution will be directly applicable to a new problem—otherwise the task being performed would not be design, it would be copying. Thus, no case in memory is likely to match all of the problem requirements, unless these are extremely vague. All the retrieved cases will probably only partially match the requirements. This is why the cases that are retrieved have to be adapted. The following section describes the evolutionary approach to case adaptation.

4.2 CASE ADAPTATION

The role of case adaptation in our framework is to adapt an initial population of cases $P^{initial}$ to find a new design d^{new} that is a solution to the new design problem whose requirements are R. This can be characterised as:

$$d^{new} = {}_{adaptation}(P^{initial},R,K).$$

The adaptation transformation ${}_{adaptation}$ is performed using an evolutionary method which evolves new (populations of) designs over time out of previously known ones, evaluating them until one of them satisfies the requirements of the new problem R and any design constraints imposed by the domain of application K. The criteria used to evaluate potential designs can vary from problem to problem (R), or can be fixed across problems and vary only from application domain to application domain (K).

The evolutionary method for case adaptation provides a mechanism for case combination and case modification. Each of the retrieved cases partially matches some of the requirements of the new design problem. Because of this, there may be some combination of the features of several of the retrieved cases that is a satisfactory solution to the new problem. Thus, case combination is a useful way of performing case adaptation.

If there is even one problem requirement that does not match any of the cases in memory, however, even an exhaustive search of all the possible combinations of the features of the retrieved cases will not find one that is acceptable. Even if *all* known cases are used to generate new potential solutions by combining their features (instead of using only the partially matching cases that were retrieved), they will in general not cover the entire set of possible combinations of features. Because of this, more than just case combination has to be done during case adaptation. Case modification is a way to overcome these limitations and achieve more flexible case adaptation. These two types of case adaptation, case combination and case modification, map onto the concepts of crossover and mutation from evolutionary algorithms, respectively.

The transformation that takes place at each evolutionary cycle, ${}_{GA}$, consists of six stages, ${}_{p\text{-}to\text{-}g}$, ${}_{combination}$, ${}_{modification}$, ${}_{g\text{-}to\text{-}p}$, ${}_{evaluation}$, and ${}_{selection}$. We discuss these steps in more detail below.

4.2.1 Phenotype-to-Genotype Transformation

The transformation ${}_{p\text{-}to\text{-}g}$ converts an initial population of design cases (or phenotypes) $P^{initial}$ into a population of genotypes $G^{initial}$. This can be characterised as:

$$G^{initial} = \{g_1, g_2, \ldots, g_\gamma\} = {}_{p\text{-}to\text{-}g}(P^{initial}), \text{ where}$$

· i½i=1, 2, …, γ, we have that $p_i \; P^{initial}$ and $g_i = f_{map\text{-}p\text{-}to\text{-}g}(p_i)$.

The mapping function $f_{\text{map-p-to-g}}$ depends on the representations of phenotypes and genotypes chosen for a given application domain and the relation between them. In our layout design example, $f_{\text{map-p-to-g}}$ involves finding the adjacency matrix that is equivalent to a house-level design case.

4.2.2 Combination of Genotypes

At evolutionary cycle t+1, the crossover transformation generates a new population of genotypes of new potential designs G_x^{t+1} from that cycle's initial population of genotypes of designs G^t as follows:

$G_x^{t+1} = \{g_{x1}, g_{x2}, \ldots, g_{x\chi}\} = {}_{\text{combination}}(G^t)$, where

- i½i=1, 2, …, χ, we have that g_i G^t and $g_{xi} = f_x(g_i)$.

Genotype combination is achieved through genetic crossover operators. If we assume that crossover combines information from only two genotypes g_i^t and g_j^t at a time, then the crossover function f_x also produces two of the new genotypes g_{xi}^{t+1} and g_{xj}^{t+1} each time it is applied. At evolutionary cycle t+1, this function is defined by:

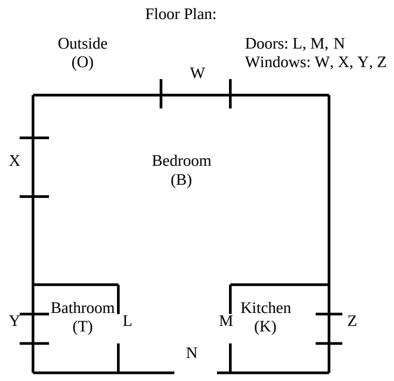
$g_{xi}^{t+1}, g_{xj}^{t+1} = f_x(g_i^t, g_j^t)$, where

- g_i^t G^t and g_j^t G^t, with $g_i^t \neq g_j^t$ i,j,
- g_{xi}^{t+1} G_x^{t+1}, and g_{xj}^{t+1} G_x^{t+1}, and therefore
- χ is even (unless g_{xi}^{t+1} or g_{xj}^{t+1} can be degenerate genotypes which are discarded before ${}_{\text{combination}}$ terminates).

That is, each application of f_x takes two distinct parent genotypes and produces two offspring genotypes to put into G_x. The choice of which two parent genotypes to cross, and where to cross them, is done randomly as part of ${}_{\text{combination}}$ before each use of f_x. The number of new genotypes produced through crossover χ can be a constant across the entire process model, or it can be chosen dynamically for each evolutionary cycle.

Case combination of floor plan layouts is illustrated in Figures 5, 6, and 7. Each selected case is illustrated by the graphical representation of the phenotype and the adjacency matrix that is the genotype. Crossover is implemented as an exchange of submatrices. In the example, submatrices of size 3x3 are exchanged, each originally located in matrix positions (0,0) and (2,2) of the parent genotypes. The resulting offspring genotypes are shown in Figure 7.

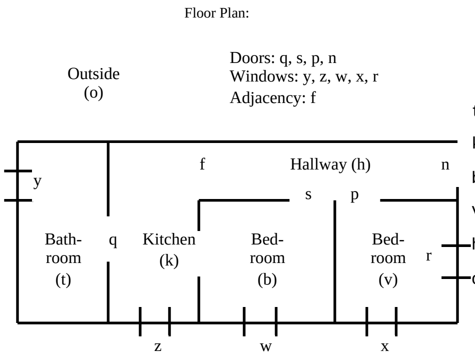
The type of adaptation performed by genotype combination is a structural adaptation (see Kolodner, 1993). From the example shown above it can be seen that the content of the information held in the two offspring is the same as that in the parents, though in different combinations. But the structure of the offspring is different from the structure of the parents: the types of room in each of the offpsring are different from the types of rooms in their parents, and they are connected in different ways



Genotype:

$$\begin{array}{c|cccc}
 & B & T & K & O \\
\hline
B & () & (L) & (M) & (N,W,X) \\
T & (L) & () & () & (Y) \\
K & (M) & () & () & (Z) \\
O & (N,W,X) & (Y) & (Z) & ()
\end{array}$$

Figure 5. Phenotype and genotype of first parent



Genotype:

$$\begin{array}{c|cccccc}
 & t & k & b & v & h & o \\
\hline
t & () & (q) & () & () & () & (y) \\
k & (q) & () & () & () & (f) & (z) \\
b & () & () & () & () & (s) & (w) \\
v & () & () & () & () & (p) & (x,r) \\
h & () & (f) & (s) & (p) & () & (n) \\
o & (y) & (z) & (w) & (x,r) & (w) & ()
\end{array}$$

Figure 6. Phenotype and genotype of second parent

OffspringC0:

$$\begin{array}{c|cccc}
 & b & v & h & O \\
\hline
b & () & () & (s) & (N,W,X) \\
v & () & () & (p) & (Y) \\
h & (s) & (p) & () & (Z) \\
O & (N,W,X) & (Y) & (Z) & ()
\end{array}$$

OffspringC1:

$$\begin{array}{c|cccccc}
 & t & k & B & T & K & o \\
\hline
t & () & (q) & () & () & () & (y) \\
k & (q) & () & () & () & (f) & (z) \\
B & () & () & () & (L) & (M) & (w) \\
T & () & () & (L) & () & () & (x,r) \\
K & () & (f) & (M) & () & () & (n) \\
o & (y) & (z) & (w) & (x,r) & (n) & ()
\end{array}$$

Figure 7. Offspring genotypes.

4.2.3 Modification of Genotypes

At evolutionary cycle t+1, the modification transformation generates genotypes of new potential designs G_m^{t+1} from that cycle's initial population of genotypes of designs G^t as follows. It could also be from G^t G_x^{t+1}, if we commit to performing combination before mutation, though we will assume that this is not necessary here:

$$G_m^{t+1} = \{g_{m1}, g_{m2}, \ldots, g_{m\eta}\} = {}_{modification}(G^t), \text{ where}$$

· i½i=1, 2, …, η, we have that g_i G^t and $g_{mi} = f_m(g_i)$.

Genotype modification complements genotype combination by making changes to the content of a single genotype. Genotype modification is achieved through genetic mutation operators. At evolutionary cycle t+1, the mutation function f_m applied to genotype g_i^t to produce a new genotype g_{mi}^{t+1} is defined by:

$$g_{mi}^{t+1} = f_m(g_i^t), \text{ where}$$

· g_i^t G^t, and g_{mi}^{t+1} G_m^{t+1}.

That is, each application of f_m takes one parent genotype and produces one new offspring genotype to put into G_m. The choice of which parent genotype to mutate, and how to mutate it, is done randomly as part of ${}_{modification}$ before each use of f_m. The number of new genotypes produced through mutation η can be a constant across the entire process model or can be chosen dynamically for each evolutionary cycle.

If the new value for the mutated design parameter is generated completely at random, spurious phenotypes that have no semantics (i.e., their meaning cannot be interpreted or makes no sense in the real world) may be produced. For a particular domain, knowledge of which values are meaningful for which attributes can be used to ensure that mutation produces only valid values, in order to reduce search time. In our application this knowledge is stored in an associative memory.

The type of adaptation performed by case modification is also known as parametric adaptation (Kolodner, 1993). The appearance (i.e., the structure) of the offspring is the same as that of the parent. But the content of the offspring is different from the content of the parent: the two phenotypes describe different objects.

4.2.4 Genotype-to-Phenotype Transformation

At a given evolutionary cycle t+1, the transformation ${}_{g\text{-}to\text{-}p}$ converts the populations of new genotypes that have been generated through crossover and mutation (G_x^{t+1} and G_m^{t+1}, respectively) into a population of new phenotypes P_n^{t+1}. This can be characterised as:

$$P_n^{t+1} = \{p_{n1}, p_{n2}, \ldots, p_{n\pi}\} = {}_{g\text{-}to\text{-}p}(G_n^{t+1}), \text{ where}$$

· i½i=1, 2, …, π, we have that g_{ni} G_n^{t+1} and $p_{ni} = f_{map\text{-}g\text{-}to\text{-}p}(g_{ni})$,

· $G_n^{t+1} = G_x^{t+1}$ G_m^{t+1}, and

· $\pi = \chi + \eta$.

The mapping function $f_{\text{map-g-to-p}}$ depends on the representations of genotypes and phenotypes chosen for a given application domain and the relation between them. In our layout design example, $f_{\text{map-g-to-p}}$ involves interpreting the information about the elements and connectors in an adjacency matrix to generate the object-based representation of the floor plan.

4.2.5 Evaluation of Phenotypes

At a given evolutionary cycle t+1, the transformation $_{\text{evaluation}}$ evaluates the populations of new phenotypes P_n^{t+1} to create an evaluated population of new phenotypes P_e^{t+1}. This can be characterised as:

$P_e^{t+1} = \{p_{e1}, p_{e2}, \ldots, p_{e\pi}\} = {}_{\text{evaluation}}(P_n^{t+1},R,K)$, where

· i½=1, 2, …, π, we have that p_{ni} P_n^{t+1} and $p_{ei} = f_{\text{evaluate}}(p_{ni},R,K)$.

The evaluation function f_{evaluate} implements what is known in GA terminology as the fitness function, which assigns a fitness value to each proposed solution. The fitness value f_i for an individual phenotype p_i in a population based on the fitness criteria imposed by both the domain K and the current problem R as follows:

$f_i = f_{\text{evaluate}}(p_i,R,K)$

The operations that the evaluation function f_{evaluate} has to perform will be discussed in more detail below. First we introduce the concept of constraints and how they can be used to represent fitness criteria for evaluating designs.

Evaluation has to determine whether a proposed solution looks and/or behaves adequately. This means that it has to analyse the phenotype of a proposed solution. This evaluative analysis can be seen as a constraint-satisfaction problem, where the space of satisfactory solutions is defined and limited by a series of constraints describing the conditions under which a proposed design is acceptable. Some of these conditions are dynamic: each problem will impose different constraints on the subset of possible solutions that can be considered satisfactory. Some of the conditions are static: the domain and the representation chosen will impose design principles and other absolute conditions on the acceptability of potential designs, irrespective of the requirements of a given problem.

The task of evaluation is then to analyse each proposed design according to each condition that defines the space of satisfactory solutions for the current problem, whether it is imposed by the problem requirements or by the domain. In our layout design example, each domain-imposed constraint to be verified is represented procedurally, as a function that takes a proposed design as input, analyses it according to the conditions corresponding to the constraint, and returns a yes/no answer indicating whether the proposed design violates the constraint or not. That is, for each domain-imposed evaluation criterion e_i, a function

f_{ei} is required that accepts a phenotype p_j as an argument and returns an indication of whether the corresponding criterion is violated by p_j or not.

Problem-imposed constraints, equivalent to problem requirements, may be represented in different ways, depending on the type of design problem one is dealing with. A function $f_{p\text{-match}}$ is required which returns a yes/no value depending on whether there is a match between the features of a given phenotype and a given problem requirement. It may be possible to use the function f_{match}, which is used in case retrieval and determines whether there is any match between the features of a given case and a given problem requirement, for this purpose, if phenotypes and cases are represented in the same way.

The fitness value is a quantitative measure of the quality of a potential solution. The total fitness of a proposed design will be the sum of its domain fitness and its problem fitness. In our layout design example, the domain expertise for the evaluation of potential designs is easier expressed in the negative rather than the positive way. The design evaluation knowledge is criteria for recognising and rejecting bad designs, rather than for recognising and accepting good designs. In order to calculate a fitness value, and assuming each constraint has the same weight or importance, each yes answer will be assigned a value of 1 and each no answer a value of 0. After evaluating each proposed solution according to all domain constraints, the total domain fitness value obtained by adding the resulting partial values will indicate the total number of domain constraints that were violated by the proposed design.

In our example, domain constraints embody feng shui principles. These principles are constant across floor plan layout design problems. The feng shui knowledge was obtained from (Rossbach, 1987). An example of a constraint at the house level is given by the following quote from (Rossbach, 1987):

> Traditionally, the Chinese avoid three or more doors or windows in a row...this...funnels ch'i [positive energy] too quickly...[CURE:]...to stop ch'i from flowing too quickly, hang a wind chime or crystal ball... (page 89)

The pseudocode that represents this constraint procedurally is the following (given a phenotype P). Other feng shui domain constraints have been implemented in a similar fashion:

```
Get the list C of all connectors in P;
Get the list Q of all wind chimes/crystal balls (potential
cures) in P;
For each connector c in C or until a bad omen has been found:
 Get the location l of c;
 Get the direction d of c;
 Set the list of connectors LU lined up with c to the empty
 list;
 Get the list Reduced of all elements in C except c;
 For each connector r in Reduced:
 If the direction of r is d And
 the location of r lines up with l along direction d,
 Then add r to LU;
 End-if;
 End-for;
 If there are two or more connectors in LU And
 no potential cure in Q lines up with r,
 Then constraint is violated;
 End-if;
End-for;
```

In contrast to the static domain constraints, the dynamic constraints embodied in the specified requirements of a given problem are expressed as positive features, i.e., they will describe conditions that an acceptable answer must satisfy or characteristics that it must have. As mentioned in Section 4.1, in our example these problem requirements specify how many of each type of room are needed in the solution.

In order to use the requirements to calculate a fitness value, and assuming each requirement has the same weight/importance, each requirement that is satisfied by a proposed solution will be assigned a value of 0, and each requirement that is not satisfied will be given a value of 1. After evaluating each proposed design according to all the problem requirements, the total problem fitness value obtained by adding the resulting partial values will indicate the number of problem requirements that were not satisfied by the proposed design.

The total fitness of a proposed solution will be the sum of its domain fitness and its problem fitness, assuming the constraints imposed by the domain and the problem are given equal importance. The total fitness f_i of a given potential solution i_i, given κ constraints (k_1 through k_κ) which are used to analyse its domain fitness and ρ problem requirements (r_1 through r_ρ) which are used to analyse its problem fitness, is calculated with the following equation in our framework:

$$f_i = \sum_{i=1}^{\kappa} k_i + \sum_{j=1}^{\rho} r_j$$

where $k_i = 0$ if constraint k_i is not violated by the solution or
$k_i = 1$ if constraint k_i is violated by the solution, and
$r_j = 0$ if requirement r_j is met by the solution or
$r_j = 1$ if requirement r_j is not met by the solution.

Determining if a constraint k_i is violated or not by a given phenotype is done using the function f_{ki}, as described above, where constraint k_i corresponds to evaluation criterion e_i. Determining if a problem requirement r_i is met by a given phenotype or not is done using the function $f_{\text{p-match}}$, as described above.

Convergence of the GA to an acceptable solution (i.e., an acceptable adaptation of the originally retrieved cases) occurs if a proposed solution has a total fitness of 0, meaning that none of the domain constraints has been violated and all of the problem requirements have been satisfied. The goal of the GA is thus to minimise the fitness value of the designs in its population. In order to achieve this goal, the average fitness of the designs in the GA’s population should monotonically decrease at each GA cycle. The subtask for determining which members of the population participate in the next round of alternatives is the selection subtask.

4.2.6 Selection of Phenotypes

The transformation $_{selection}$ consists of two parts, f_{sort} and f_{select}. At a given evolutionary cycle t+1, the sorting function f_{sort} takes the phenotypes in the initial population of the cycle P^t (it is presumed they have previously been evaluated and their fitness value is known), plus the phenotypes newly generated and evaluated in the cycle P_e^{t+1}, and sorts them according to their fitness value, producing a sorted population of phenotypes P_s^{t+1}. That is:

$$P_s^{t+1} = f_{sort}(P^t \quad P_e^{t+1}).$$

The selection function f_{select} then chooses the best of the sorted phenotypes, to produce an initial population for the next evolutionary cycle P^{t+1}. That is:

$$P^{t+1} = f_{select}(P_s^{t+1}).$$

Selection produces an initial population for the next evolutionary cycle. During the current GA cycle, an initial population of γ designs was used as a basis for creating π (where $\pi=\chi+\eta$) new designs, χ of them through crossover and η of them through mutation. Let us assume that we want to keep the size of the GA population constant across evolutionary cycles. Because of this, of the sorted population P_s^{t+1} consisting of $\gamma+\pi$ designs (γ of them coming from P^t, and π of them newly generated in the current evolutionary cycle), selection has to choose the best γ. These γ designs become the initial population for the next GA cycle. f_{sort} is implemented as a quicksort function, and f_{select} simply takes the first γ individuals of the population sorted by fitness resulting from f_{sort}.

The selection module has the role of deciding which combinations and modifications of previously known designs and/or which previously known designs to keep for further adaptation and which ones to discard. This choice is made according to the relative quality of the proposed designs as determined by the evaluation module. This quality in turn depends on the fortuitous results of the random decisions made while performing crossover and mutation to produce new adaptations of past designs. Once selection has been performed, a new evolutionary cycle can start to generate new potential designs again, starting with an improved initial population resulting from the selection process. After several evolutionary cycles, this process ensures that the average quality of the designs in the population monotonically improves.

5. Experimental Results

We implemented the process model using 12 Frank Lloyd Wright prairie house designs taken from (Hildebrand, 1991) as the case memory. The constraints used for evaluation were taken from text descriptions of feng shui principles presented in (Rossbach, 1987). The fitness function includes 11 constraints at the landscape level and 16 constraints at the house level. The population size was kept constant during the evolutionary cycles. The selection of which members of the population to include in the next generation was made from the combination of parents and offspring, so a parent design could participate in more than one generation. Of the new individuals generated in each cycle, 80% were generated by crossover and 20% by mutation.

The design process for generating floor plans can be considered in terms of the efficiency of using design cases as the starting point for generating new designs and in terms of the quality of the design solutions as examples of prairie house style. To analyse the efficiency of the approach, the program was run 20 times using the case memory of 12 prairie house layouts and 20 times using 12 randomly generated floor plans. We consider convergence and number of cycles to convergence as the basis for comparing efficiency. In the 20 runs, 5 of the runs converged using both the randomly generated case memory and the prairie house memory. The frequency of convergence was not affected and therefore our results show that using an initial population of actual designs does not improve the probablity of convergence. The number of cycles to convergence was affected by the use of a case memory of actual designs, the convergence rate was on average, 50% faster than when using the randomly generated floor plans. The details are shown in Table 1.

To analyse the quality of the design solutions, we look at one example in detail. The new design specification is stated as:

((bedroom 3) (bathroom 2) (fireplace 1) (music-room 1))

That is, we want a floor plan with 3 bedrooms, 2 bathrooms, 1 fireplace and a music room.

TABLE 1. GA cycles required before convergence:

Trial #:	Random:	Trial #:	FLW cases:
1	114	25	54
9	333	31	34
11	357	36	32
14	274	37	406
17	160	39	90
Avg.:	241.6	**Avg.:**	123.2

The solution generated for this specification is shown in Figure 8. In this run, all 12 cases were retrieved, the process converged after 61 cycles, and 28 crossover operations and 3 mutations occurred. The solution combines features from 3 of the 12 designs that were originally retrieved. Features from the other cases were present in the final generation, but were not part of the final solution.

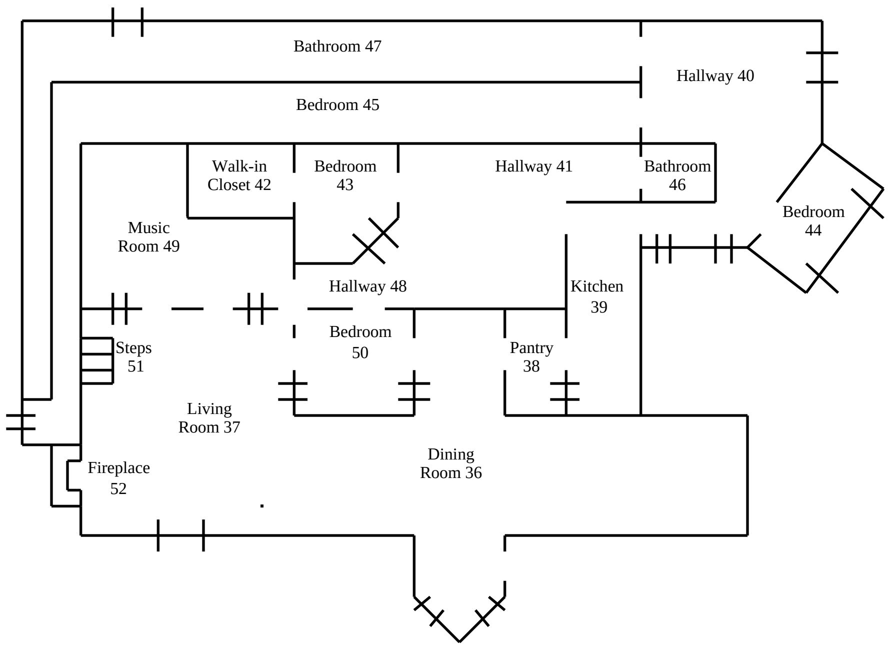


Figure 8. Floor plan design generated by evolutionary case adaptation.

We can observe some of the features of the parent designs in the new design, for example:

- The floor plan has many hallways, a feature inherited from the Avery Coonley House.
- The protuberance from the dining room is a feature of the Robie House.
- The skewed room is a feature inherited from Willits House.
- In general, the prairie house style can be recognised in the large dining and living rooms that merge into each other with the fireplace.

6. Summary

We have presented a process model for the evolutionary adaptation of design cases. This process model combines aspects of CBR and GA's into one framework for adaptive design. The four subtasks of the evolutionary algorithm, combination, modification, evaluation, and selection, together guide the search for a solution to a new problem, while at the same time the search has a degree of flexibility due to the random nature of some of the decisions made in the subtasks. The search is given additional initial bias by the use of cases. Cases are retrieved from a case memory based on their relevance to the requirements of the new problem and are used to initialise the population of potential solutions of the evolutionary algorithm. In the process model, knowledge is only required in the form of design cases which provide a framework for the construction of new potential solutions, and in the form of domain constraints which help evaluate potential solutions generated by the evolutionary algorithm.

This process model has the following characteristics:

· The design precedents as cases can define a type or style of design. In our example the cases described the prairie house style of Frank Lloyd Wright.
· The GA operators provide a knowledge-lean mechanism for adaptation. In our implementation, the crossover and mutation operators were independent of floor plan layout.
· The evaluation of adapted designs need not be numerical constraints. In our example, the evaluation constraints representing the practice of feng shui considered direction and number of components and connectors in a layout.
· The knowledge needed during adaptation can be independent of the knowledge in the cases. In our example, the knowledge in the cases of prairie houses was expressed independently of the knowledge in the feng shui constraints.

References

Gómez de Silva Garza, A. and Maher, M. L.: 1996, Design by interactive exploration using memory-based techniques, *Knowledge-Based Systems*, **9**(1), 151-161.

Gómez de Silva Garza, A. and Maher, M. L.: 1999a, Evolving design layout cases to satisfy feng shui constraints, *Proceedings of the Fourth Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA99)*, Shanghai.

Gómez de Silva Garza, A. and Maher, M. L.: 1999b, An evolutionary approach to case adaptation, *Case-Based Reasoning Research and Applications: Proceedings of the Third International Conference on Case-Based Reasoning (ICCBR99)*, Monastery Seeon, Munich, Springer-Verlag, pp. 162-172.

Gómez de Silva Garza, A.: 2000, *An Evolutionary Approach to Design Case Adaptation*, Ph.D. Dissertation, Department of Architectural and Design Science, University of Sydney, Australia.

Hildebrand, G.: 1991, *The Wright Space: Pattern and Meaning in Frank Lloyd Wright's Houses*, University of Washington Press, Seattle.

Kolodner, J.L.: 1993, *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, California.

Maher, M.L. and Balachandran, B.: 1994, Multimedia approach to case-based structural design, *Journal of Computing in Civil Engineering*, **8**(3), 359-376.

Mitchell, M.: 1998, *An Introduction to Genetic Algorithms (Complex Adaptive Systems Series)*, MIT Press, Cambridge, Massachusetts.

Rossbach, S.: 1987, *Interior Design with Feng Shui*, Rider Books, London.

Wang, S.H.: 1995, *WIN: A Case-Based Design Approach to the Structural Design of the Wind Systems of Buildings*, Master's Thesis, Department of Architectural and Design Science, University of Sydney, Australia.

Zhang, D.M.: 1994, *A Hybrid Design Process Model Using Case-Based Reasoning*, Ph.D. Dissertation, Department of Architectural and Design Science, University of Sydney, Australia.