Modelling Design Exploration as Co-Evolution

Mary Lou Maher and Josiah Poon

Key Centre of Design Computing
University of Sydney
Australia
(mary, josiah)@archsci.arch.su.edu.au

Abstract

Most computer-based design tools assume designers work with a well defined problem. However, this assumption has been challenged by current research. The explorative aspect of design, especially during conceptual design, is not fully addressed. This paper introduces a model for problem-design exploration, and how this model can be implemented using the genetic algorithm (GA) paradigm. The basic GA, which does not support our exploration model, evaluates individuals from a population of design solutions with an unchanged fitness function. This approach to evaluation implements search with a prefixed goal. Modifications to the basic GA are required to support exploration. Two approaches to implement a co-evolving GA are presented and discussed in this paper: one in which the fitness function is represented within the genotype, and a second in which the fitness function is modelled as a separately evolving population of genotypes.

Keywords: exploration, genetic algorithm, co-evolution, conceptual design

1. Introduction

Most computer-based design tools (like CAD and CASE) assume designers work with a well-defined problem. However, this assumption of a well-defined problem has been challenged by current research[2,3,10]. Recent work has demonstrated that the explorative aspect of design, especially during conceptual design, is not fully addressed. The assumption that designers have a clear picture of the problem and solution is not valid. Designers do not usually have a complete problem description before commencing a design synthesis. During *conceptual design*, designers play around with ideas to get more understanding about the problem rather than focus on just finding a solution. Design is therefore an iterative process of searching the design problem space as well as the solution space. This kind of phenomenon is called *exploration* in design. Since the phenomenon of exploration is not well understood, this is rarely supported by current design tools. A computational model of exploration is required in order to provide better assistance to designers. If we view the change in problem definition as a response to the search for alternative solutions, we can model exploration as a change in goals. Evolutionary systems provide a mechanism for modelling change, however, they assume the goal is fixed. Allowing the goals to change over time, as well as the solution space, can be modelled as a co-evolutionary system.

Genetic Algorithms[9] (GAs) provide an alternative to traditional search techniques by adapting mechanisms found in genetics. Three notions are borrowed from biological systems:

- the *phenotype*, which can be a living organism for biological systems or a design solution for design systems;

- the *genotype*, which is a way of representing or encoding the information which is used to produce the phenotype; and
- the *survival of the fittest*, which determines whether a genotype survives to reproduce.

In GA systems the genotype is usually represented as a binary string whose length varies with each application. For example, a genotype may look like: 001001101. GAs introduce a representation (genotype) that differs from its expression (phenotype) in order to perform changes that couldn't be possible at the phenotype level. The genotype representation allows combination or mutation to occur in order to construct better strings. Some measure of fitness is applied to each string after combination and mutation to determine which strings participate in generating the next generation of the population.

A simple genetic algorithm considers a population of n strings and applies the operators: reproduction (or selection), crossover, and mutation in order to create the next generation. Reproduction is a process in which strings are copied according to their fitness function. Crossover is a process in which the newly reproduced strings are mated at random and each pair of strings partially exchanges information. Mutation is the occasional random alteration of the value of one of the bits in a string. Algorithms used to implement these processes are described in detail Goldberg's book[6].

In this paper a design process based on a genetic algorithm is presented which can model characteristics of explorative design: the search for problem definition as well as the search for solution. The use of an evolutionary system in which the genotypes represent alternative problem definitions and alternative solutions provides the basis for the co-evolution of problem space and solution space.

1.1 Exploration in Design

Since design has been categorised as a problem solving activity [20] , design is treated as a search of the solution space for a result. This idea has dominated the direction of artificial intelligence in design for some time. However, the validity of this hypothesis has been queried by recent work. For example, Corne and his colleagues[2] from Edinburgh University suggest that it is inappropriate to consider design as a search problem because a search problem requires a well defined problem space whereas a design problem is usually ill-structured. They propose design as "exploration" as follows:

> ".. involves the construction and incremental extension of problem statements and associated solutions .."

Logan and Smithers[12] further elaborate this definition that

> ".. the formulation of the problem at any stage is not final ... As the design progresses, the designer learns more about possible problem and solution structures as new aspects of the situation become apparent and the inconsistencies inherent in the formulation of the problem are revealed. As a result, .. the problem and the solution are redefined..."

This proposal on design has stressed the importance of incremental and interactive aspects of problem with solution.

Navinchandra[18] defined exploration in the program CYCLOPS as

> ".. Exploration is the process of generating and evaluating design alternatives that normally would not be considered.."

He focuses on alternatives and this is achieved through criteria relaxation and criteria emergence. The relaxation is not constraints relaxation but a relaxation of the threshold value. This changing of threshold values causes a part of the solution space which is originally inside bound of the pareto curve to be explored. Solutions in this inside bound solution space can be examined as alternatives. Emergence described in CYCLOPS is a recognition activity. Criteria from precedent cases may be recognised to be relevant and interesting enough to apply to current situation. The introduction of new criteria adds a new dimension for the designer to consider. The new criteria will be included to be part of the evaluation of design solution.

Another definition of "exploration" is offered by Gero[3] that:

> ".. Exploration in design can be characterised as a process which creates new design state spaces or modifies existing design state spaces..."

This definition extends the "state space" concept of search[20], so that the state space is changed during exploration. This definition implies that the solutions in the given or predefined state space are insufficient for exploration. Gero continues to suggest that

> ".. exploration precedes search and it, effectively, converts one formulation of the design problem into another .. Part of designing involves determining what to design for (function or teleology), determining how to measure satisfaction (behaviour), and determining what can be used in the final artefact (structure).."

The definition relates exploration to search, indicating that exploration precedes search, and at the same time differentiates exploration from search.

Maher[14] provides a definition which also relates search and exploration:

> ".. search becomes exploration where the focus of the search changes as the process continues .."

This definition identifies search as a part of exploration, but not the same as exploration and also characterises the two as distinct, i.e. search has a definite goal while exploration doesn't. Search as part of exploration cannot guarantee convergence because the design requirements change with the design solutions at the same time. However, convergence criteria could be externally defined and separate to the design requirements, recognising the fact that design usually completes when time has run out or factors external to the concerned problem.

For the remainder of this paper, we present a formal model of exploration.The model is illustrated in Fig.1 as the interaction of problem space and solution space. The problem space (or the functional requirements) is represented by *P*, and the solution space is represented by *S*. *Exploration* is defined as a phenomenon in design where *P* interacts and evolves with *S* over time. This is to emphasised here that the exploration addressed in this paper is the phenomenon of a (design) process. This is in contrast with exploration in *search algorithms*.[13]

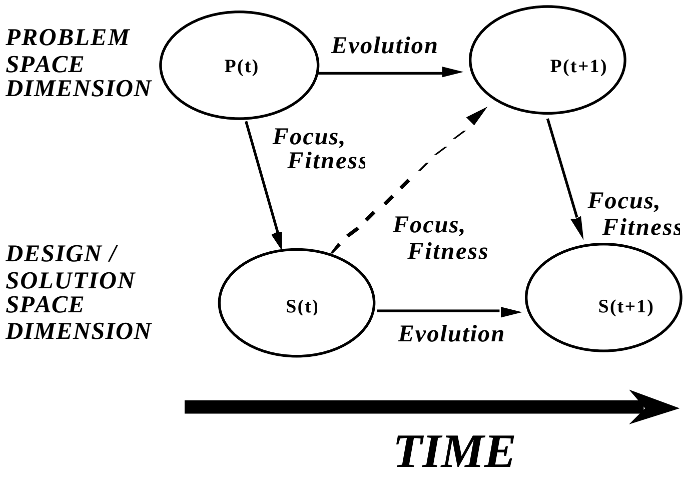


Fig.1 Problem-Design Exploration Model

The phenomenon of exploration as illustrated in Fig.1 has the following characteristics:

1. There are two distinct search spaces: Problem Space and Design Space.
2. These state spaces interact over a time spectrum.
3. Horizontal movement is an evolutionary process such that
   a. Problem space P(t) evolves to P(t+1), P(t+2), and so on;
   b. Solution space S(t) evolves to S(t+1), S(t+2), and so on.
4. Diagonal movement is a search process where goals lead to solution. This can be "*Problem leads to Solution*" (downward arrow) or "*Solution refocusses the Problem*" (upward arrow).

The problem space P(t) is the design goal at time *t* and S(t) is the solution space which defines the current search space for design solutions. The solution space S(t) provides not only a state space where a design solution can be found, but it also prompts new requirements for P(t+1) which were not in the original problem space, P(t). This is represented by the dashed upward arrow from design space S(t) to problem space P(t+1). The upward arrow is an inverse operation where S(t) becomes the goal and a "*search*" is carried out in the problem space, P(t+1), for a "*solution*". This iterative relationship between problem space and design space evolves over time.

This model of exploration depicts an evolutionary system, or in fact, two evolutionary systems. The evolutionary systems are the problem space and the solution space. The evolution of each space is guided by the most recent population in the other space. This model is called co-evolution and provides the basis for a computational model of design exploration. The basis for co-evolution is the simple genetic algorithm where special consideration is given to the representation and application of the fitness function so that the problem definition can change in response to the current solution space.

1.2 Related Research In Genetic Algorithms

Genetic algorithms provide the basis for modelling evolutionary systems. The application of GAs to design include the solution to the truss design problem. The ten-member truss problem[7] aims to find the optimal weight of each member for a given pre-determined configuration, such that the whole structure is stable and has a minimum weight. The configuration and the fitness function remains unchanged throughout the GA process. This represents a basic application of genetic algorithms to a design optimisation problem.

Watabe and Okino[21] further study this problem by searching for structural shape as part of the problem. This is achieved by the introduction of new genetic operator called T-mutation. There are two types of T-mutation. The first one, T1-mutation, adds one new node to a randomly selected bar. The second one, T2-mutation, changes the topological structure without changing the number of nodes. The effect after the application of the T-mutation results in a new species which consists of individuals with different structural configurations. The changed configuration opens up a further new solution space to search. This application of a genetic algorithm shows how the representation of the genotype determines the level at which the search occurs, in this case the search included a search for a configuration. However, the goal as defined by the fitness function remains to be the minimum weight configuration.

SAGA[8], Species Adaptation Genetic Algorithms, allows the genotype to change in length as well as content so that species can emerge. He suggests that the notion of a search space is a metaphor when the question of “Where in this whole search space is the optimum?” is asked. However, this metaphor implies a space of pre-defined extent with a predefined goal. If a structure is to be evolved with potentially unrestricted capabilities, the simple GA, which has fixed length genotypes, is not an appropriate tool. The capability to represent a variable-length genotype is important to evolution. As the length increases, the population evolves as a species rather than global search. However, his model does not show how individuals from the solution space can affect the problem space.

Koza[11] recognises the importance of co-evolution and suggests the term in biology is sometimes used to reflect the fact that all species are simultaneously co-evolving in a given physical environment. He uses Game Playing Strategy to elaborate on co-evolution, where two players in a game are represented as two populations of individuals. The fitness of a strategy of a player is measured by its performance against all strategies deployed by the other player. The fitness is, thus, a relative score. The performance of the two players continue to evolve with respect to the strategies by the opposing player. The mutual interactions and implicit relationships between players in a game is extended to a general conclusion as follows:

> “In co-evolution, there are two (or more) populations of individuals. The environment for the first population consists of the second population. And conversely, the environment for the second population consists of the first population ... Co-evolution is a self-organizing, mutually bootstrapping process that is driven only by relative fitness.”

This provides a model for co-evolution where two solution spaces evolve in competition to each other, yet the goal remains the same. Co-evolution is affected by each search space defining the threshold for survival in the other search space. We present a co-evolutionary system in which the two spaces are not in competition with each other, yet they evolve in response to each other. Three differences between the co-evolution of Game Playing Strategy and the co-evolution of Problem-Design Space are:

1. the two populations in a game are opponents with the aim to beat each other, whereas in our co-evolution model, the aim is to explore the Problem Space and Design Space and to help each other to acquire better fitness values;

2. the purpose of co-evolution in the Game Playing Strategy is to measure how good an individual strategy can stand when played against various strategies by the opponent, while our co-evolution model aims to measure how good an individual from a population can satisfy (adapt) the expectations of individuals from another population; and
3. the same fitness function is used for both spaces in the Game Playing Strategy, only the threshold for reproduction is changed in co-evolution, while our co-evolution model applies a potentially different fitness function to each space.

2. A Co-Evolutionary Process for Explorative Design

Artificial evolutionary systems have been developed by John Holland[9] whose goals have been twofold :

1. to abstract and explain the adaptive processes of natural systems and
2. to design artificial software systems that retain the important mechanisms of natural systems.

The efficiency and flexibility of biological systems is due to rules of self-repair, self-guidance and reproduction that hardly exist in most artificial systems. We use these ideas as a basis for defining a computational model of exploration. We build on the simple GA and extend these ideas to allow for the co-evolution of search spaces.

A simple GA, as shown below, is the basis for developing an evolutionary process model for explorative design.

```
t = 0;
initialize genotypes in Population(t);
evaluate phenotypes in Population(t) for fitness;
while termination condition not satisfied do
       t = t + 1;
       select Population(t) from Population(t-1);
       crossover genotypes in Population(t);
       mutation of genotypes in Population(t);
       evaluate phenotypes in Population(t);
```

We apply the simple GA to the design process, where we assume the process begins with an initial population of design concepts or styles encoded as genotypes. The evaluation determines which genotypes survive. The evaluation is performed by evaluating a fitness function and operates on the phenotype, which in the design process is the instantiation of the design rather than the design concept. The genotypes and phenotypes are the representation paradigms for design concepts and realised design solutions. The processes of selection, crossover, mutation, and evaluation are the basis of the search for a design solution.

Selection is a process in which individuals are copied according to their fitness function. This means that an individual with a higher value has a higher probability of contributing one or more offspring in the next generation. This operator is an artificial version of natural selection, a Darwinian survival of the fittest among individuals. In a natural population, fitness is determined by an individual's ability to survive. In the context of design, a fitness function representing the design requirements determines whether a design is suitable or not. Once an individual has been selected for reproduction, an exact replica of the individual is made. This individual is then entered into a mating place for further genetic operator action.

Crossover is a process in which the newly reproduced individuals are mated at random and each pair of individuals partially exchange information using a cross site chosen at random. For example if we consider the individuals A1 = 0110 | 1 and A2 = 1100 | 0 (where the separator symbol is indicated by |), the resulting crossover yields the following two news individuals A1' = 01100 and A2' = 11001. Crossover in a design process occurs when two design concepts are partially combined to form a new design concept.

Mutation is the occasional random alteration of the value of one of the bits in an individual. When used sparingly with reproduction and crossover, mutation is an insurance policy against loss of notions. In fact mutation plays a secondary role in the operation of GAs because the frequency of mutation to obtain good results in empirical GAs studies is on the order of one mutation per thousand bit transfers[6]. Mutation has the potential to make small changes to a design concept, rather than a crossover process that makes large changes. We do not employ mutation in our co-evolutionary model of design.

Evaluation is a process of determining if a genotype continues in the next round of crossover. The termination condition is usually related to the evaluation, that is, when the evaluation of the population yields a suitable design, the process is terminated. Evaluation in the design process occurs by testing the performance of the design against relevant criteria. In the GA model of design, the fitness function is the basis for evaluation. The fitness function as a representation of design requirements can be predefined for the entire search process or it can be allowed to change as the genotype population changes. By changing the fitness function in response to the current population, the process models the ability of designers to change their focus when an interesting solution is found. This can be modelled as co-evolution of the design space and the performance space, where each space then becomes the population of genotypes for its own evolution and the fitness function for the other space.

In order to model *exploration* using a GA, the fitness function, Fitness(t), becomes variable and evolves over time. This deviates from the traditional view of the fitness function which remains the same for the evaluation of the population of genotypes from different cycles. The population(t) goes through the normal reproduction, crossover and mutation step to create the next generation. The fitness of each genotype is measured using the function, Fitness(t), which differs from Fitness(t-1).

The co-evolution of the design genes (solution space) and the fitness function (problem space) provides a model for design as exploration. Two approaches to representing coevolution are:

- **CoGA1**: A single composite genotype is formed by the combination of a problem requirements and a design solution. The fitness function is defined locally for each design solution.
- **CoGA2**: The two spaces are modelled as two sets of genotypes and phenotypes: one for modelling problem requirements and one for modelling design solutions. The current population of each space provides the fitness measurements for the other.

2.1 CoGA1: Fitness Functions and Design Solution in the Same Chromosome

This first co-evolving algorithm has two modifications to the basic GA:

1. the fitness function (problem part, P) and design solution (solution part, S) are put into one genotype,
2. there are two phases of crossover-evaluation operations in each generation instead of the convention of one phase.

The algorithm, CoGA1, is shown below.

```
CoGA1
t = 0;
initialize genotypes in Population(t);
evaluate phenotypes in Population(t) for fitness;
while termination condition not satisfied do
t = t + 1;
select Population(t) from Population(t-1);
/* Phase 1: from S to P */
       crossover genotypes in Population(t) at Performance_space;
       evaluate phenotypes in Population(t);
/* Phase 2: from P to S */
       crossover genotypes in Population(t) at Design_space;
       evaluate phenotypes in Population(t);
```

Inside the repeating loop, there are two phases of GA operations for each generation. If no satisfactory solution is found in previous operations with the stated problem, the problem is revised to give new dimensions for the solution space. Hence, the first phase corresponds to the shift of attention of fitness function when a solution space is given, i.e. the upward arrow from *S* to *P* in our model of problem-design exploration. In phase 1, crossover occurs in the problem part of the genotype, as illustrated in Fig.2(a). For example, the crossover point to the parent genotypes cut the problem part to P11 and P12, and P21 and P22. For the same solution carried forward from the previous phase, the fitness is evaluated using a different fitness function, i.e. the same S1, which is evaluated by P21 and P12 in parent genotype, is evaluated by P11 and P22 in the new recombined child genotype. The fitness value for each design solution represents a local fitness.

After the problem is revised, the second phase relates to the search for a solution with the reformulated fitness function from Phase 1. This corresponds to the downward arrow from *P* to *S* in the model, as shown in Fig.2. Crossover occurs on the design solution part of genotypes: the S11, S12, S21 and S22 in Fig.2(b). The fitness of a design solution is not evaluated by a common global fitness function, but by the fitness function defined as the problem part in the same genotype. In other words, the fitness score of each genotype is again a local fitness value. In our example, the offspring which has solution part composed of S11 and S22 is evaluated by P1; while the other offspring, which has S21 and S12 in its solution part, is evaluated by P2.

After phase 2 the solutions are used to check the termination condition to determine whether another generation is necessary. Currently, the termination condition is defined as a fixed number of generations, meaning that the exploration stops when a predetermined amount of time has passed. However, the termination condition can be any globally defined condition that does not evolve in response to the alternatives found in the solution space.

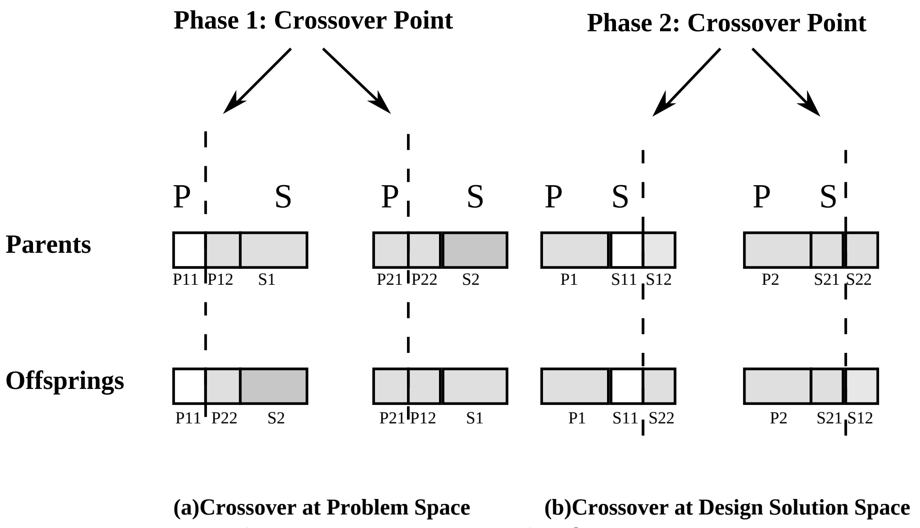
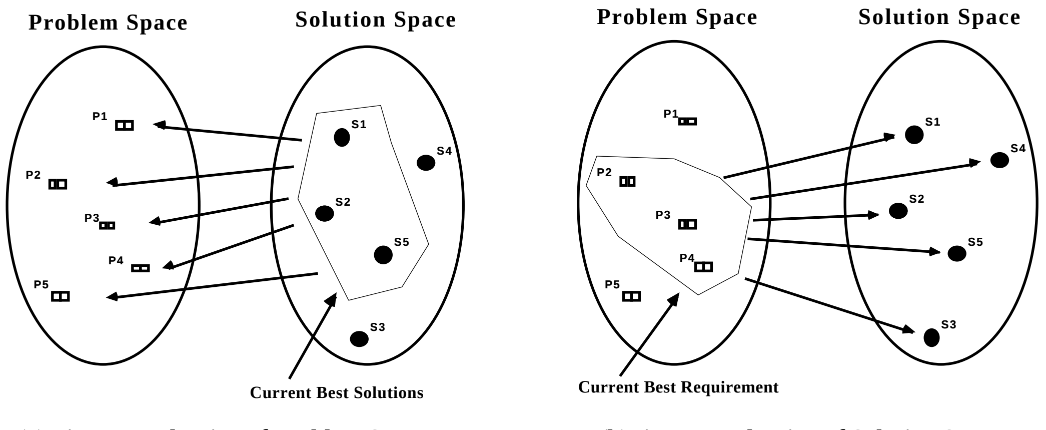


Fig.2 Crossover Operation for CoGA1

2.2 CoGA2: Problem Requirements and Design Solution as Two Interacting Populations

A second approach to co-evolution is to maintain separate spaces of genotypes for problem requirements and design solutions. As illustrated in Fig.3, the CoGA2 uses the current selection from each population to be the fitness function for evaluating the individuals in the other population. The problem requirements is modelled as a collection of criterion, where each criterion is represented as a genotype in the Problem Space. Every problem criterion genotype has a label and a weighting (i.e. the genotype has a length of 2). A problem is, thus, a combination of individual genotypes with their current weights. If we allow the crossover operator to cut and paste a different weight to a criterion, followed by selecting a random number of genotypes. These problem criteria will collectively define a problem which has a different perspective and emphasis to be solved. The fitness of a solution is defined by the current collection of criterion. In the other direction, the fitness of a criterion is defined by the number of times that criterion is satisfied in the current collection of individuals in the Solution Space.



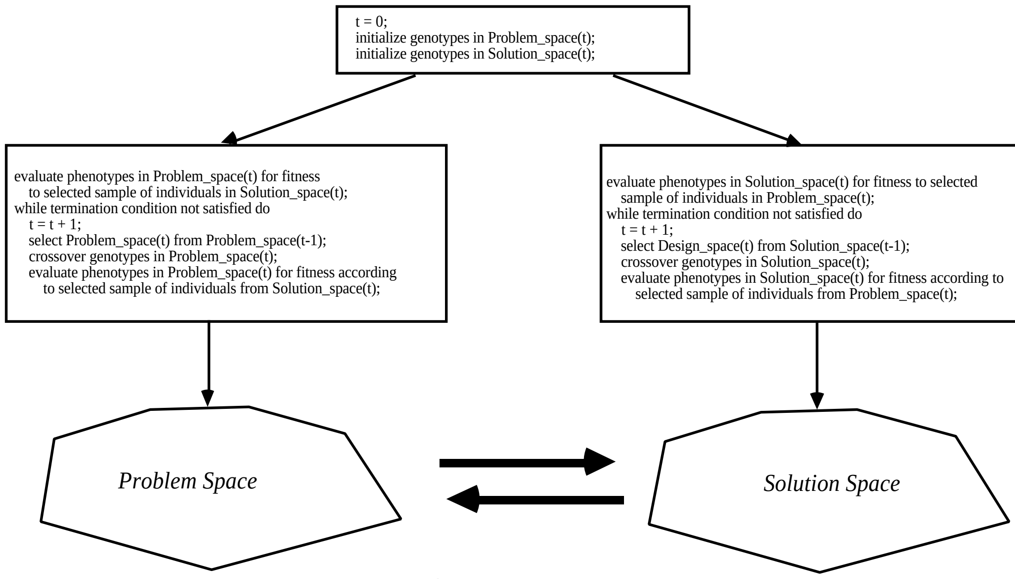
(a) Fitness Evaluation of Problem Space (b)Fitness Evaluation of Solution Space

Fig.3 Fitness Evaluations in CoGA2

The CoGA2 starts with initialising the two populations which represent problem and solution. An initial evaluation of individuals from the Solution Space is performed using the initial design requirements as defined by the user. The initial evaluation of the Problem Space is performed based on the selected individuals from the Solution Space. The termination condition is checked and the pattern of “phases” appear in CoGA2 as well. Each phase in CoGA2 corresponds to a different evaluation function, rather than to a different crossover operation as in CoGA1. The CoGA2 algorithm is shown below.

```
CoGA2:
t = 0;
initialize genotypes in Problem_space(t);
initialize genotypes in Solution_space(t);
initial-evaluate phenotypes in Solution_space(t) for fitness according
       to user’s initial requirements;
initial-evaluate phenotypes in Problem_space(t) for fitness according
       to user’s initial requirements;
prepare sample of individuals in Problem_space(t) for measuring
       fitness of phenotypes in Solution_space(t+1);
prepare sample of individuals in Solution_space(t) for measuring
       fitness of phenotypes in Problem_space(t+1);
while termination condition not satisfied do
       t = t + 1;
       /* Phase 1: from S to P */
              select Problem_space(t) from Problem_space(t-1);
              crossover genotypes in Problem_space(t);
              evaluate phenotypes in Problem_space(t) for fitness
                     according to selected individuals from
                     Solution_space(t-1);
              prepare sample of individuals in Problem_space(t) for
                     measuring fitness of phenotypes in Solution_space(t+1);
       /* Phase 2: from P to S */
              select Solution_space(t) from Solution_space(t-1);
              crossover genotypes in Solution_space(t);
              evaluate phenotypes in Solution_space(t) for fitness
                     according to selected sample of individuals from
                     Problem_space(t-1);
              prepare sample of individuals in Solution_space(t) for
                     measuring fitness of phenotypes in Problem_space(t+1);
```

The algorithm of CoGA2 is, in fact, an interaction of two GA processes: a GA process on the Problem_space and another on the Solution_space, as illustrated in Fig.4. Each space evolves as in a simple GA, where the fitness function of one space is based on the current population of the other space.



```
t = 0;
initialize genotypes in Problem_space(t);
initialize genotypes in Solution_space(t);
```

```
evaluate phenotypes in Problem_space(t) for fitness
    to selected sample of individuals in Solution_space(t);
while termination condition not satisfied do
    t = t + 1;
    select Problem_space(t) from Problem_space(t-1);
    crossover genotypes in Problem_space(t);
    evaluate phenotypes in Problem_space(t) for fitness according
        to selected sample of individuals from Solution_space(t);
```

```
evaluate phenotypes in Solution_space(t) for fitness to selected
    sample of individuals in Problem_space(t);
while termination condition not satisfied do
    t = t + 1;
    select Design_space(t) from Solution_space(t-1);
    crossover genotypes in Solution_space(t);
    evaluate phenotypes in Solution_space(t) for fitness according to
        selected sample of individuals from Problem_space(t);
```

Fig.4 Interactions of Two Genetic Populations in CoGA2

3. Examples of Co-evolutionary Design

Examples are presented in this section to illustrate how various algorithms, introduced in previous sections, can be applied to the design of floor plans. The design problem is kept simple to illustrate the difference between the co-evolutionary approach and the simple GA approach to design. The sample problem is based on the use of GA's by Maher and Kundu[15] to model adaptive design where a floor plan for a building is developed, based on a population of existing floor plans. The purpose of the Maher and Kundu application was to demonstrate the role of crossover in combining design concepts to form new concepts. Here we use the same problem of floor plan design to illustrate both adaptation through crossover of concepts and co-evolutionary exploration through the representation and modification of the fitness function.

In their paper, the topology of the floor plan is represented as a set of vertices, where each vertex is a room in the floor plan, and the edges represent the adjacencies of the rooms. Since the adjacency matrix for a graph-based representation of floor plan topology is a sparse matrix, a list-based representation is used to store only the non-zero elements of the adjaceny matrix. The general form of representing a floor plan is as follows:

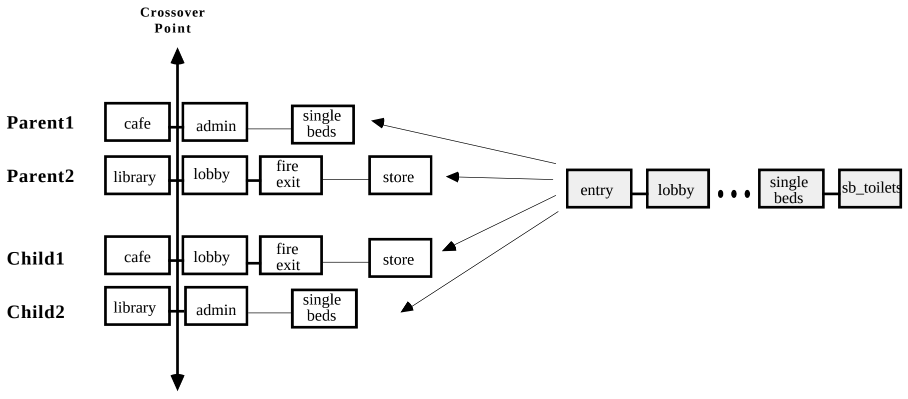
graph(graph_label, building_label, [vertex_list], [edge_list])

The design process is initiated with the definition of a set of rooms to be included in a new building, and the final solution is a topology of a floor plan that has the highest percentage match to the required rooms. In the Maher and Kundu application, the initial population of floor plans was the basis for finding new floor plan topologies. In our example, we will also allow the predefined required list of rooms to change as different floor plan topologies are generated.

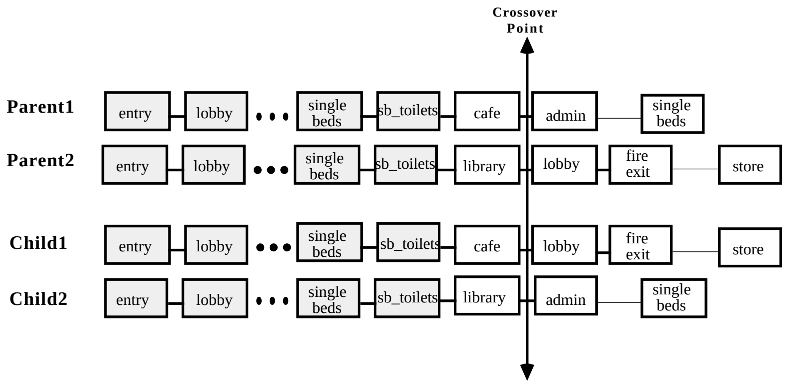
3.1 Example of CoGA1

In CoGA1 each genotype has two parts: the Problem and the Solution. The problem part is a representation of the set of rooms that comprise the required list of rooms for the new building. The solution part is a representation of the set of rooms that the new building will have. In this example we will only include the rooms that are part of the floor plan and do not include the adjacencies that represent the topology of the floor plan.

The problem part and the solution part representation is illustrated in Fig.5, where the problem part is represented by shaded boxes and solution part by white boxes. The original implementation of this example, as illustrated in Fig 5(a), is to evaluate new designs by an external performance function, ie. *{entry, lobby, ... , single_beds, sb_toilets}*. In the implementation of CoGA1 the required list of rooms is attached to the list of rooms in the solution part, forming a composite genotype, as illustrated in Fig.5(b).



(a) Representation of Solutions with Fixed Fitness Function



(b) Representation of Solutions with Variable Fitness Function

Fig.5 Floor plan design with fixed and variable fitness

The initialisation of the genotypes for CoGA1 requires the representation of previous floor plan designs in which the required set of rooms and the complete list of rooms is represented within a single genetic code. The initial evaluation of the genotypes is based on a comparison of the population with the user's new design requirements. A subpopulation is selected, where each member has at least one room in the solution part in common with the user's list of rooms.

In the first phase of CoGA1 (*"a change of focus"*), the genetic operations are performed in the Problem part. If the crossover operator cuts between locus 1 and 2 of the Problem part in Fig.6, these partial problems from parent genotypes (Parent1 and Parent2) recombine to form new offspring (Child1 and Child2). As a result, a different problem is generated to test the fitness of solutions carried forward from their parents, i.e. the solution list *{cafe, admin, ..., single_beds}* of Parent1 is evaluated by a different criteria list *{library, lobby, ..., single_beds, gym}* of Child1. The performance is measured by the number of matching rooms in the solution with the local requirements. Every matched room will have the fitness score increased by 1. Hence, for Child1, it has a fitness measure of 1 because there is only one room in the solution list matching the requirements (i.e. the *single_beds*). Child2 also has a score of 1 when its *store* meets one of the criteria. The local fitness score will be compared with fitness scores from other genotypes so that better performing genotypes can get into the next phase.

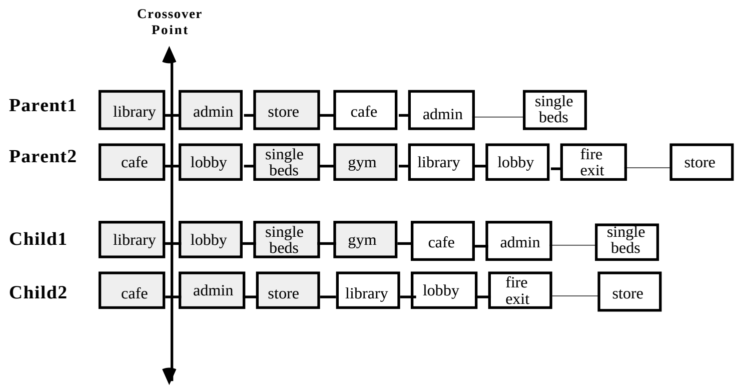


Fig.6 Phase 1 of CoGA1: A Change of Focus

Assume both Child 1 and Child2 are selected to the second phase. In Phase 2, they become Parent1 and Parent2, as illustrated in Fig.7. Genetic operators are applied to the Solution_part of the genotype in Phase 2 to indicate an attempt to find a solution under the revised problem. Assume the crossover point is between locus 1 and 2 in the Solution_part. Part of the solution is moved to another genotype while the problem remains intact in this phase. The same requirement *{library, lobby, ..., single_beds, gym}* retained from Parent1 to Child1 is used to evaluate a different solution. The evaluation to Child1 is 1 (the matching of *lobby*) and so is Child2 (the matching of *admin*). The GA cycles continue until termination conditions are met .

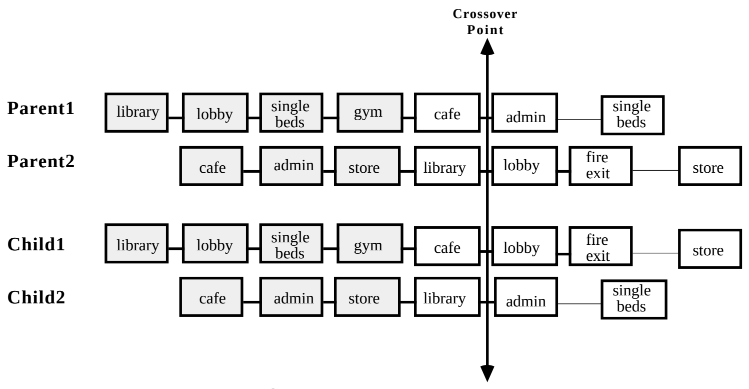


Fig.7 Phase 2 of CoGA1: From Problem to Solution

3.2 Example of CoGA2

This section continues with the same problem from Section 3.1 to illustrate how CoGA2 works. The set of required rooms now forms a separate space of genotypes, in which each genotype represents a room and the importance of having that room in the final design. The solution space is comprised of genotypes that each represent a list of rooms that were used in previous floor plan designs. Each space is initialised, so that the Problem space includes all possible rooms and their relative importances and the Solution space includes a set of floor plans used in previous designs.

The initial evaluation of the phenotypes in the solution space is based on the user's required set of rooms for the new design problem. Each solution is given a fitness value equal to the weighted sum of the rooms in the solution that match the user's required room list.

The initial evaluation of the phenotypes in the problem space is also based on the user's required set of rooms for the new design problem. The fitness value of each problem criterion is incremented by 1 if it matches the user's initial required room list.

In Phase 1 of CoGA2, the problem requirements are revised. Based on the value determined in the previous evaluation, a selection of solutions forms the set of Current Best Solutions (CBS) and a selection of requirements forms the set of Current Best Requirements (CBR). During crossover, the genotypes in the CBR are mated, resulting in a modification of the weights associated with each requirement, or in this example, with each required room. The evaluation in Phase 1 is illustrated in Fig.8. During evaluation, the value of each requirement is determined by the number of solutions in the CBS that satisfy the requirement. For example, since the problem feature *lobby* can be found in both S2 and S3, the fitness of the *lobby* is 2; whereas the problem feature *library* can only be found in S2, its fitness score is 1 in this generation. The result of evaluation is used in the next phase as the basis for the next CBR. The CBR represents a collection of requirements and their importance in the evaluation of genotypes in the solution space.

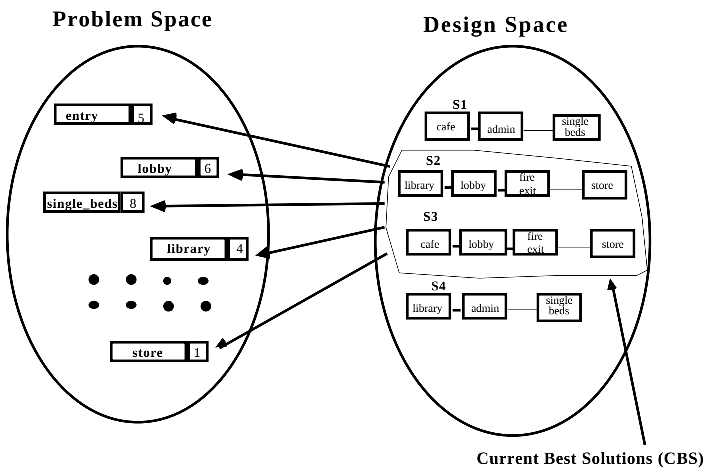


Fig.8 Evaluation of Problem Criteria by Selected Design Solutions

In Phase 2 the genotypes in the solution space are mated. The CBS provides the subpopulation for the crossover operation. The result of crossover is a new set of solutions to be evaluated by the CBR. As illustrated in Fig.9, the CBR is composed of four atomic problem features: *entry, lobby, single_beds and library.* Assume that S1 is a floor plan resulting from the crossover operation in Phase 2, and it comprises *cafe, admin, single_beds*. This design solution only satisfies one feature from the CBR, i.e. *single_beds*. Since the feature *single_beds* has a weight of 8, the fitness score of S1 is 8. The fitness of S2, S3 and S4 are measured in the same way and they are given a value of 10, 6 and 12 respectively. The fitness value provides the basis for the selection of the next CBS in the next generation.

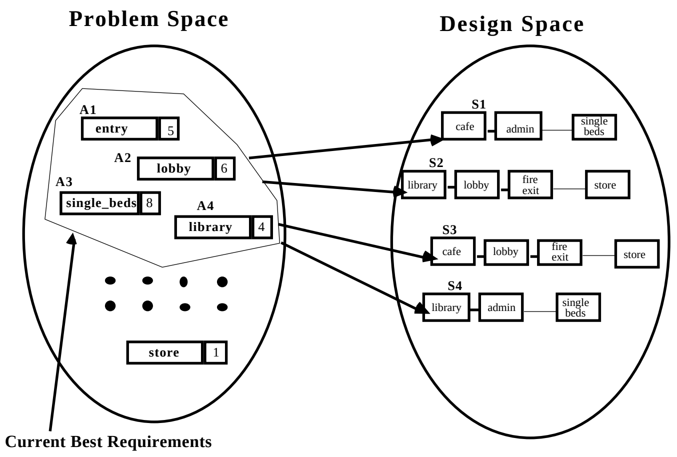


Fig.9 Evaluation of Solutions by Selected Criteria

4. Comparison of Co-GA1 With Other Examples of Exploration

This section illustrates how the representation and process of CoGA1 compares to other implementations of design exploration. Specifically, CoGA1 is considered for the representation of adapting design prototypes[1]; the representation of the "best optimum" beam design[5]; and landscape design in the CYCLOPS program[18]. Each of the following subsections begin with description of the problem domain, the representation of the problem in our co-evolution model, and how the use of CoGA1 can add the exploration dimension to the implementation.

4.1 Co-evolution of Design Requirements and Prototypes

In the work of Alem and Maher[1], the design prototype is used as the basis for creative design using a GA to adapt and combine design prototypes for new design requirements. A design prototype is a generalization of groupings of elements in a design domain which provides the basis for the commencement and continuation of a design[4]. A prototype represents a class of elements from which instances of elements can be derived. It comprises the knowledge needed for reasoning about the prototype's use as well as about how to produce instances in a given design context. Applying the concept of prototypes to CoGA1 in our model of co-evolution, a design prototype is a representation of the solution part of a genotype. For example, the representation of a beam design prototype may be:

```
BEAM PROTOTYPE
        Functions:
                span-distance
                support-gravity-load
        Behaviours:
```

bending-stress
shear-stress
deflection
Structure:
length
cross-section-area
section-modulus
material

The problem part of the genotype comprises the features of the beam that form the requirements of a particular design problem. Each individual in the population, therefore, comprises a set of design requirements and a set of solution features. The prototype formalism adds the extra dimension of categorising the features according to their role in the design process, ie. function, behaviour, or structure.

Design requirements can be requirements on function, requirements on behavior, requirements on the structure, or any combination of the three. Because of the incomplete nature of design requirements, no exact value is required. Each attribute value can be given in terms of a range of values. For example, a set of design requirements may be:

DESIGN REQUIREMENTS
Function:
span-distance: 30 ft
gravity-load: 30 kips/ft
Behavior:
bending stress: < 36 k/sq.in.

The genotype in CoGA1 has the following two components:

DESIGN REQUIREMENTS | DESIGN PROTOTYPE

Design selection operator initially uses the design requirements for the new design problem to evaluate and select an initial population of design prototypes. Selection is based on a partial match between the set of requirements and the attributes of a design prototype. An acceptable performance is one in which any subset of the requirements are matched. Following the initial selection, evaluation and selection is based on the local requirements in the genotype.

The use of CoGA1 differs from the implementation of Alem and Maher because it allows the design requirements to be modified as well as the original design prototypes. It could be argued that creative design can result either from the modification of design solution spaces (ie. the modification of design prototypes) and/or the modification of the design requirements.

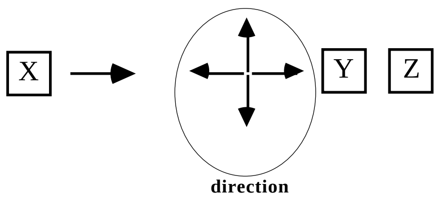
4.2 “Better Optimum” Beam

In the work carried out by Gero and his students[5], a genetic algorithm formalism is introduced that can learn a new shape grammar to produce better beam section. Their problem is to generate different beam sections from a set of 4 square building blocks, called cells, which build up a section to be juxtaposed with one another. There is also a set of rules that governs this juxtaposition. Each cell has a label that is referred to by the composition rules of the grammar that has a weight associated with it. The aim of the exercise is to find the optimum performance from a population of different beam topologies after an application of these grammar rules nine times, and the two properties chosen to be optimised are:

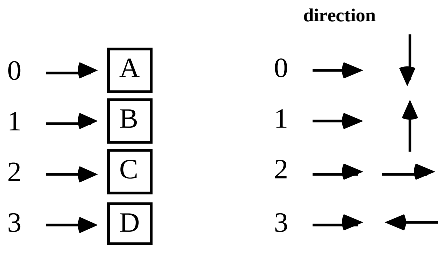
- The moment of inertia of the beam section, which has to be maximised.
- The perimeter of the beam section, which has to be minimised.

There are two approaches to this problem. The first one is to encode the order of execution of the rules of the shape grammar at the genotype level. The second one is to encode the rules themselves. The former one is a search to a fixed state space for a given set of rules in the grammar. The second one, argued by these authors, is to create and explore new state spaces through the evolution of new rules. This paper will concentrate on this latter approach.

To make the grammar implicit, they encode the nine repeating groups (rules) in the genotype. Each rule is composed of four labels (A, B, C, D) and a direction. The rule is of the following format:



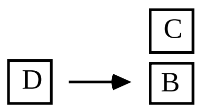
where X is specified by the last applied rule, then followed by a direction symbol (which can be up, down, left and right) and new labels Y and Z to represent current move. Their encoding is as follows:



For example, the string

0,2,3,1,1,2,...

specifies the following moves: For initialisation, the first two numbers are ignored and, therefore, we start off at the third digit. The digit ‘3’ which specifies the fourth label, which represents D. The next three numbers after ‘3’ is ‘1,1,2’. The first ‘1’ is a direction symbol which indicates to move up. And the second ‘1’ is a label which corresponds to B. The ‘2’ is a label symbol which represents C. The nine moves are continued in this way. The result of this first move is shown below:



It is claimed that the feasible solution space for this learned grammar has a marked improvement in performance over the fixed grammar. The new state space occupies a larger feasible area and they argue a substitution of new solution space, S_n, to the original space, S_o, constitute to this result, where S_o, S_n.

To model their evolutionary learning of shape grammars with CoGA1 is similar to the graph-based GA with a fixed problem as in Fig.5(a). The problem is represented by the two properties: *moment of inertia (I)* and *length of perimeter (L).* The genotype is a composite of the problem and the solution (the rules) as follows:

$$\boxed{\boldsymbol{I}} - \boxed{\boldsymbol{L}} - \boxed{0} - \boxed{2} - \boxed{3} - \boxed{1} - \boxed{1} - \boxed{2} - \cdots$$

Although the original implementation is an exploration where alternative state spaces are created; exploration is limited to changes in the solution space. The problem space is left untouched throughout the genetic operations. In applying CoGA1, we allow different fitness measurements to be attached to the genotype, such as *cost of production (C), durability of the beam (D)* and so on. Thus, typical genotypes that facilitate exploration are as follows:

$$\boxed{\boldsymbol{I}} - \boxed{\boldsymbol{L}} - \boxed{0} - \boxed{2} - \boxed{3} - \boxed{1} - \boxed{1} - \boxed{2} - \cdots$$
$$\boxed{\mathbf{C}} - \boxed{\boldsymbol{I}} - \boxed{\boldsymbol{D}} - \boxed{2} - \boxed{1} - \boxed{3} - \boxed{0} - \boxed{2} - \boxed{3} - \cdots$$

The inclusion of the Problem part in the genotype has extended the original exploration of solution spaces alone to exploration of both state spaces.

4.3 CYCLOPS

CYCLOPS[18] is a design problem solver. The problem domain is landscape design. A hypothetical scenario is used for the design process. The scenario allows new designs to be explored by relaxing constraints and by making tradeoffs among objectives. New criteria can sometimes emerge. This leads to the discovery of opportunities or to the recognition of unexpected problems. Hence, exploration in CYCLOPS is a result of two issues: criteria relaxation and criteria emergence.

It is suggested that constraint relaxation and objective maximisation/ minimisation are just two sides of the same coin. Hence, they can be treated in a uniform manner. Initial objectives and constraints are grouped to become what is called *criteria*. Instead of being simply satisfied or not-satisfied, a criterion has several levels of satisfaction. The relaxation was not performed directly on the constraint, but relaxing the threshold value so that alternatives can be considered. In its current implementation, the relaxed form of a criterion is represented as a matrix of ranks. The best value is assigned a rank of 1, the next best region with a rank of 2, and so on. When the criteria are relaxed (i.e. moving to higher ranks), solution space which is originally inside the pareto curve can be explored. Solutions in this inside-bound solution space can be examined as alternatives.

Criteria can emerge during the design process. The emergence of new criteria will create a new dimension for the designer to consider. New criteria are recognised by CYCLOPS program from a database of past experiences. In CYCLOPS terminology, a past experience is called a precedent which is a record of the conditions and the effects experienced. For example, "good view" emerges as a evaluation criterion because of a home situates in a higher altitude and has a valley in the front where this valley has lakes and woods. This criterion emerges with the precedent shown below:

```
conditions:   (home altitude high) AND (home view valley)
              AND (valley has lakes) AND (valley has woods)

effect:(favourable: home good-view) AND (good-view rank 1)
```

Hence, emergence in CYCLOPS is achieved through recognition, which is a knowledge-intensive activity. If a design has certain conditions that match a precedent, then the precedent is retrieved and the *n* effects are applied to current design. The newly *n* applied effects serve as criteria for evaluating new designs. In other words, the problem space is expanded by *n* -dimensions.

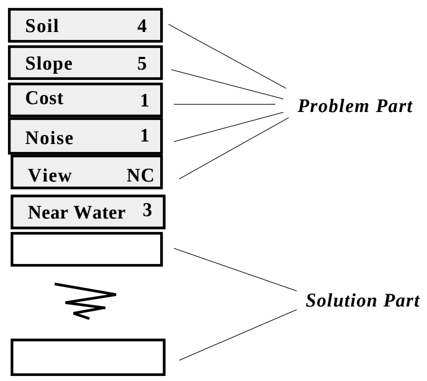


Fig.10 Criteria with Ranks in Problem Part of Genotype

The ideas in CYCLOPS can be modelled with our CoGA1 algorithm, as illustrated in Fig.10. Each design case is represented by a genotype. The conditions (or criteria) of a design case are included in the problem part of the genotype. Each criterion in the problem part carries a rank which is initially randomly generated. Hence, the rank of "1" indicates the best value is expected, and a higher rank implies the relaxation of the appropriate criteria. When the rank of each criterion is assigned with value other than "1", this indicates the relaxation of the appropriate criterion. In other words, the area below the pareto surface is explored. The design mutation operator can further change the ranks of some of the "problem part" genotypes such that evaluation of alternatives in solution space would become possible.

The mechanism to produce emergence in CoGA1 is unlike CYCLOPS' recognition approach. Conditions of a previous design case are represented in the "problem part" of the genotype. As the design process continues in CoGA1, the crossover operation adds and takes away the criteria used to evaluate each design solution. This provides a mechanism for the emergence of new criteria for a specific solution, since the fitness for each solution is locally defined in each genotype. However, all potential criteria are represented somewhere in the space of genotypes.

5. Conclusions

In this paper, we propose a model for Problem-Design Exploration that can be implemented using modified genetic algorithms. The modifications to the GA results in co-evolution. One modification of the simple GA is called CoGA1. This approach is novel in its representation of the fitness function and design solution in the same genotype and the alternate GA operations on different parts of the genotype allows the co-evolution of design requirements and design solutions. A second approach, called CoGA2, represents the design problem as genotypes in one space and the design solutions as genotypes in a second space. These space evolve in response to each other by providing the fitness function for the other space. These algorithms change the notion of search from having a fixed goal to an exploration of potential goals.

A number of issues need to be resolved for the implementation of these algorithms to a specific problem domain. The issues raised here are: the notion of a local fitness function, the selection of individuals based on local fitness, and the need for a fixed termination condition.

The fitness function in CoGA1 varies from one genotype to another; in CoGA2 the fitness varies from one generation to another. In using the fitness value as a basis for selection, the issues of heterogeneous performance criteria needs to be addressed. Because of the difficulty in comparing heterogenous performance criteria, we advocate this can be resolved by the normalisation of raw values. However, should the normalisation be standard for all features? Or can different feature defines its own normalisation function? Will this divergent approach to normalisation causes an inconsistency in the problem part? Also, a normalisation process can simply map a raw value to a scale of [0..1], the normalised value does not indicate how well a solution is matching up with an expected performance. Hence, another approach is to use the membership function from the fuzzy logic paradigm. The outcome of mapping a raw value to a membership function is also in a scale of [0..1], however, the '1' indicates the raw value achieves an expected requirement while '0' represents a failure. Any other number between the [0..1] scale indicates the closeness of the initial raw value to the desired behaviour.

The selection process in CoGA1 and CoGA2 is based on the evaluation of variable length performance criteria. Given that the value is typically a summation of satisfied criteria, the issue of the length of the criteria list becomes important. Is a solution that satisfies more criteria necessarily better than a solution that satisfies fewer but all relevant criteria? One approach to allow solutions that satisfy fewer criteria to be selected is to use a modified roulette[17] to determine the selected individuals. In a modified roulette, the number of criteria used to evaluate the solution could be used in the first round of selections, followed by a selection based on the fitness value of the solution. Also, if a threshold value is used to determine further consideration, should the threshold be a constant? Should this threshold evolve? Do we need to guarantee the fitness of individuals in the next generation by imposing an absolute threshold value? Will the performance be improved if individuals from all previous generations are used instead of the last generation?

A major difficulty with both CoGA1 and CoGA2 is that the criteria for fitness changes over time, precluding the possibility of convergence. A termination condition is needed to stop the evolutionary process. If convergence is not a condition to terminate the process, is *time* the only consideration? or should there be a meta fitness function? are there any guidelines or global criteria to indicate an exploration process should terminate?

Since a simple GA only has one value to be optimised, the common approach relies on representation such that the result can be directly derived from a genotype. However, in the co-evolving GA, there are heterogeneous performance criteria with changing goals. How should we interpret the evaluation? Is the best performing result in the last generation to be *the* result for the problem? Or should we consider the generation trace to identify the best ever design solution? i.e. the current best or the global best? Suppose we find the *best* solution, but what does it mean to have a solution satisfying a problem which is different from the initial problem specification? A metric has to be devised to determine the “goodness” or “usefulness” of a solution.

A remark is inserted here that crossover is the only recombination operator in our present algorithms, a usual secondary operator, mutation, is omitted. Mutation may serve as an useful operator to recover information which is lost together with the low performing genotypes. This can become a critical operator if we want to recover lost information from the problem space.

The work to date is the implementation of CoGA1 and this algorithm was trialed on the design of a braced frame[16]. Two implementations of CoGA1 are written up. They are the *Phases CoGA1* and the *Flip Flop CoGA1*. Performance and efficiency of these two implementations are measured and compared. It has also been reported that Case-Based Initialisation (CBI), which is the seeding of an initial population with relevant cases, can help to improve performance and efficiency[19]. Tests have been carried out to assess these claims, however we cannot find a conclusive result using our co-evolving paradigm.

Although the algorithms in this paper introduce many unresolved issues, the algorithms provide a basis for the co-evolution of a design problem with the potential design solutions. The implementation of these algorithms will force a resolution of the issues raised. The purpose of introducing the algorithms and the issues here is to demonstrate the variations possible in the interpretation of the algorithms CoGA1 and CoGA2.

Acknowledgments. This work is supported by the Australian Research Council and an Australian Postgraduate Research Award.

References

1. Alem, L. and Maher, M.L., A model of creative design using a genetic metaphor. In T. Dartnall (Ed.) *Artificial Intelligence and Creativity: An Interdisciplinary Approach*, Kluwer, 1994, pp 281-291.
2. Corne, D; Smithers, T. and Ross, P., Solving Design Problems by Computational Exploration. In John S. Gero and Fay Sudweeks (Eds.), *Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design*, Talinn, Estonia, June 16-19, 1993, 1993, pp.249-270.
3. Gero, J.S., Towards a Model of Exploration in Computer-Aided Design. In John S. Gero and Fay Sudweeks (Eds.), *Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design*, Tallinn, Estonia, June 16-19, 1993, pp.271-291.
4. Gero, J.S., Prototypes : A Knowledge Representation Schema for Design, *AI Magazine*, Winter, 1990.
5. Gero, J.S.; Louis, S.J. and Kundu, S., Evolutionary Learning of Novel Grammars for Design Improvement. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 8,* 1994, pp.83-94.
6. Goldberg, D.E., *Genetic Algorithms: In Search of Optimization and Machine Learning*. Addison-Wesley, 1989.
7. Goldberg, D.E. and Samtani, M.P., Engineering Optimization Via Genetic Algorithm. In *Proceedings of the Ninth Conference on Electronic Computation,* 1986, pp.471-482.
8. Harvey, I., Species Adaptation Genetic Algorithms: A Basis for a Continuing SAGA. In F.J.Varela & P.Bourgine (Eds.), *Toward a Practice of Autonomous Systems: Proceedings of First European Conference on Artificial Life*, MIT Press, 1992.
9. Holland, J.H., Concerning Efficient Adaptative Systems, In M.C Yovits, G.T. Jacobi, & G.D Goldstein (Eds.), *Self-organizing Systems*, Spartan Books, 1962, pp.215-230.
10. Jonas, W., Design as Problem-Solving? or: Here is the Solution - What was the Problem? *Design Studies*, 14(2), 1993, pp.157-170.
11. Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
12. Logan, B. and Smithers, T., Creativity and Design as Exploration. In J.S. Gero and M.L. Maher (Eds.), *Modelling Creativity and Knowledge-Based Creative Design*, Lawrence Erlbaum Associates, 1993, pp.139-175.
13. Louis, S.J., Genetic Algorithms as a Computational Tool for Design. PhD Dissertation for the Department of Computer Science, Indiana University, USA, 1993.
14. Maher, M.L., Creative Design Using a Genetic Algorithm. *Computing in Civil Engineering*, ASCE, 1994.
15. Maher, M.L. and Kundu, S., Adaptive Design using a Genetic Algorithm. In the *Preprints of the IFIP WG5.2 Working Conference on Formal Design Methods,* 1993.
16. Maher, M.L., Poon, J. and Boulanger, S., Formalising Design Exploration as Co-Evolution: A Combined Gene Approach. In John S. Gero and Fay Sudweeks

(Eds.), *Preprints of the IFIP WG5.2 Workshop on Formal Theory of Design on CAD*, Mexico, 1995.

17. Michalewics, Z., *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.
18. Navinchandra, D., *Exploration and Innovation in Design*. Springer-Verlag, New York Inc, 1991.
19. Ramsey, C.L. and Grefenstette, J.J. (1993). Case-Based Initialization of Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, (ed.) Stephanie Forrest, Morgan Kaufmann Publishers, pp.84-91.
20. Simon, H.A., *The Sciences of the Artificial*. MIT Press, 1969.
21. Watabe, H. and Okino, N., Structural Shape Optimization by Multi-Species Genetic Algorithm. In Chris Rowles, Huan Liu and Norman Foo (Eds.), *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence (AI'93)*, Melbourne, Australia, November 16-19, 1993, pp.109-116.