Adapting Problem Specifications and Design Solutions Using Co-evolution

M.L. Maher[1], A. Gómez de Silva Garza[2]

[1]Key Centre of Design Computing and Cognition
Faculty of Architecture
The University of Sydney, NSW 2006
Australia
mary@arch.usyd.edu.au

[2]Instituto Tecnológico Autónomo de México (ITAM)
Río Hondo #1, Colonia Tizapán San Ángel
01000—México, D.F.
México
agomez@itam.mx

Abstract

In this paper we present a co-evolutionary model of design in which potential solutions to a design problem evolve in parallel with the problem description. This computational model is based on the observation that creative designers often refine and revise the design requirements of a particular problem at the same time as they generate and propose an evolving series of potential solutions to the problem. Genetic algorithms guide the search for a solution using a fixed fitness function, and revisions to the crtieria for the best solution involve manually modifying the fitness function. In our model of co-evolutionary design, the fitness function is automatically changed as the problem space and solution space co-evolve. In the paper we describe the model in general, show how we have applied it to the design domain of structural engineering, and present some preliminary experimental results.

1 Introduction

Generally, a design problem originates from a need that is specified as a set of requirements and results in a design solution. A characteristic of designing is the reconsideration and revision of the requirements while developing a solution. This reconsideration of the requirements is integrally related to ideas proposed and developed in the design solution. We present a computational model of design that can reason about and change populations of requirements and solutions through the application of a co-evolutionary algorithm. There is no user involvement in our framework: the co-evolutionary model is autonomous.

This paper reports on recent research that characterises the properties of co-evolutionary design, as introduced in [1]. Specifically, the development of a model of co-evolutionary design raises questions regarding fitness, convergence, and the interaction between problem specification and design solution. The issue of "fitness" in co-evolutionary design is raised because we assume a model in which the requirements, as goals of the design process, change, and therefore the fitness measure of proposed solutions may change during the course of problem solving and convergence may not mean achieving the "best" fitness. In our formulation of co-evolutionary design, we separate the idea of fitness from convergence, present the idea of convergence as achieving a more homogenous search space, and propose that termination is based on an analysis of the alternative fitness functions.

2 Design as Co-evolution

Co-evolutionary systems are those in which there are two or more interacting search spaces. Paredis [2] provides an overview of co-evolutionary algorithms. The co-evolutionary approach is suitable for solving complex problems that can be decomposed into a set of sub-problems when these sub-problems can be solved by using conventional GA or EC methods. Solving the sub-problems will lead to solving the original problem. CCGA-1, described in [3], allows n species to evolve independently.

We have developed co-evolutionary design systems that model interacting search spaces related to design problem solving. In our model of co-evolutionary design we include two search spaces: the requirement space and the solution space. Both the requirements of a design problem and the solutions proposed for those requirements evolve in time, in response to changes in each search space. We use the word focus to mean the current fitness function being used to

evaluate the individuals in a search space. The focus of the search is based on the requirements when searching the solution space, and based on the solutions when searching the requirement space, as illustrated in Figure 1. There is only one focus at any one time for a given space. However, the focus of the search in both spaces evolves as a side effect of the evolution of the individuals in the two spaces.

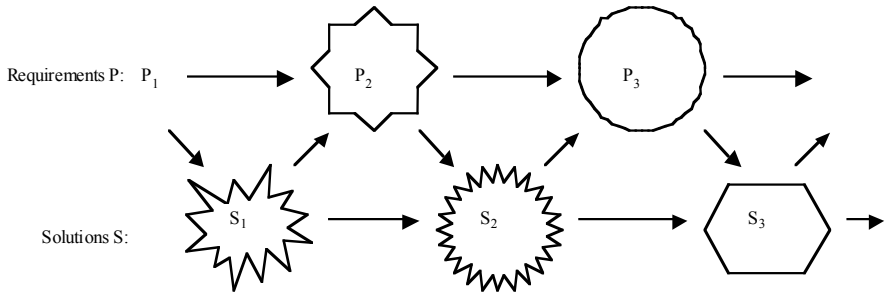

Figure 1: Illustration of co-evolutionary design

We have developed a general algorithm for co-evolutionary design, shown in Figure 2 (see also [4]). The algorithm is adapted and extended from a simple GA (for example, see [5]). Each of the two phases of co-evolutionary design is a search process using a simple GA and unchanging fitness function, denoted Φ. Therefore, each phase corresponds to one design focus and a change in phase indicates a change in focus. The concept of a "generation" or "cycle" in co-evolution has different meanings from a simple GA, since there is more than one population that is evolving.

```
/* CoDESIGN algorithm: */
T=0;            /* counter for co-evolutionary cycles */
t=0;            /* counter for evolutionary cycles */
Φ={};           /* set of fitness functions */
Initialise genotypes of requirements P_t and solutions S_t;
While termination conditions, f(T,Φ), are not satisfied:
        T=T+1;
        t=t+1;
/* Phase 1: determine focus for new problem requirements, redefine problem requirements space,
search for best problem requirements */
        If T≠1 then
                v=1;                    /* counter for local evolutionary cycles */
                φ'_t,T = f(S_t);        /*fitness function */
                Φ=Φ∪{φ'_t,T}            /* history of fitness functions */
                P_t = g(P_t, φ'_t,T)    /* revised problems requirements space */
                Repeat
                        t=t+1;
                        v=v+1;
                        P_t ::= select genotypes in P_t-1;
                        Reproduce, crossover and mutate genotypes in P_t;
                        Calculate fitness of phenotypes in P_t using φ_T';
                Until convergence by similarity of members of population: h(v, P_t, P_t-1);;
        End-if
/* Phase 2: determine focus for new design solution, redefine design solution space, search for best
design solution */
        v=1;                    /* counter for local evolutionary cycles */
        φ_t,T = f(P_t);         /* fitness function */
        Φ=Φ∪{φ_t,T}             /* history of fitness functions */
        S_t = g(S_t, φ_t,T)     /* revised design solution space */
        Repeat
                t=t+1;
                v=v+1;
                S_t ::= select genotypes in S_t-1;
                Reproduce, crossover and mutate genotypes in S_t;
                Calculate fitness of phenotypes in S_t using φ_T ;
        Until convergence by similarity of members of population: h(v, S_t, S_t-1);
End-while
```

Figure 2: CoDESIGN: an algorithm for co-evolutionary design

Each space is alternately searched using an evolutionary algorithm that cycles through several generations, and the alternation of evolutionary searches of the two spaces continues through several co-evolutionary generations. Therefore, the algorithm has three counters for time: T, t, and v. T indicates the change in generation from one full cycle of co-evolution to another; T is updated after the solution space and the problem space have each been searched and

converged once. t indicates a new generation in the simple GA sense; t is updated after each generation of search in one of the spaces, *i.e.*, after a cycle of selection, crossover, and mutation in either P or S. v is similar to t, except that the value is reset for each phase so that v is a local counter for evolutionary cycles. The value of v is reset after each evolutionary search in each space (*i.e.*, it is reset twice for each value of T, once for the search in S and once for the search in P), whereas the value of t continues to be incremented with each evolutionary cycle throughout the execution of the entire co-evolutionary algorithm.

The concepts of fitness, convergence and termination for our model of co-evolutionary design are discussed below.

Fitness: Survival of the fittest in evolution has been translated as a fitness function in simple GA's. This fitness function is the basis for the comparison of alternative solutions. In co-evolutionary design, when we let the definition of the fitness function change, the value of the fitness function can no longer serve as the basis for comparison for all alternative designs. The performance of individuals in the solution space can only be compared when they are evaluated using the same fitness function. This makes it difficult to compare the performance of solutions across different phases of the co-evolutionary design process. The performance of individuals is used to determine which members of the population "survive," or are selected to participate in the next generation of search. When searching a space (either the problem space P or the solution space S), the performance is measured by how well the alternatives satisfy the focus. The focus consists of a dynamic component and a static component. When evaluating design solutions in S, the dynamic component of the focus is a function of the set of possible design requirements currently present in P; when evaluating design requirements in P, the dynamic component of the focus is based on the current set of design solutions in S. The dynamic component of the fitness function is the part of the focus that evolves as a side-effect of changes in the makeup of the two spaces being searched. The static component of the focus depends on knowledge about a particular design domain, which defines some overall constraints on what is considered a satisfactory solution or a satisfactory requirement.

In the CoDESIGN algorithm, we represent the set of possible design requirements in the space P and the corresponding focus, or fitness function, as ϕ. We represent the current set of design solution alternatives in the space S and the corresponding focus as ϕ'. The fitness function, ϕ, is defined as a function of P, *i.e.*, ϕ=f(P), and the fitness function ϕ' is defined as a function of S, *i.e.*, ϕ'=f(S). The subscripts of ϕ and ϕ' are t and T, where t indicates which generation of evolutionary search in P or S was used to derive ϕ or ϕ' and T indicates in which co-evolutionary generation the focus ϕ or ϕ' is used. There will be a new ϕ and a new ϕ' for each T, but not for each t, since each phase may result in multiple generations of P or S in which the focus stays the same.

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

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

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

one space ends (converges) and a new fitness function (focus) is thereby generated with which to evaluate the results of the search which is about to commence in the other space.

3 Example: Structural System Layout

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

Structural system layouts are specified by combining elements. For example, a frame can be represented as a combination of a horizontal and two vertical elements. Elements are connected to each other to form a structural system. Each element has two end-points, as shown in Figure 3.

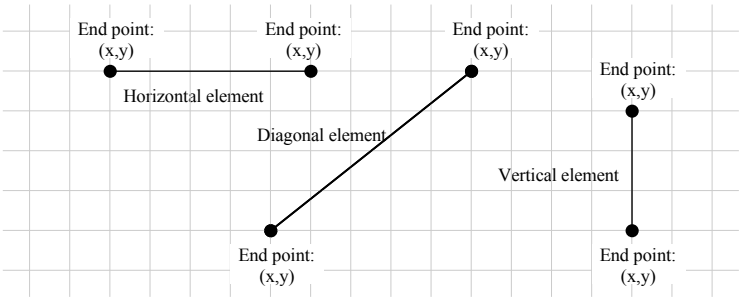

Figure 3: Horizontal, diagonal, and vertical elements with end points

It is only possible to have connections to other elements at the first and/or at the second end-point. The connection types are not considered in this implementation. An example of a possible structural system as a solution for a design problem is shown in Figure 4.

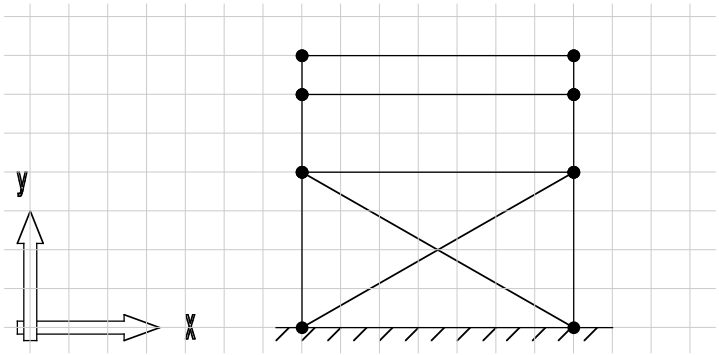

Figure 4: A possible structural system as a solution

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

The constrained free spaces of a problem represent openings, such as doors, windows, tunnels, and required areas for technical equipment like heating or lights. Neither loads nor structural members can be placed within a constrained free space. A constrained free space is described by its geometry (length and height) and its location ((x,y)-coordinates of the lower left point). An example of a problem specification consisting of loads and constrained free spaces is shown in Figure 5.

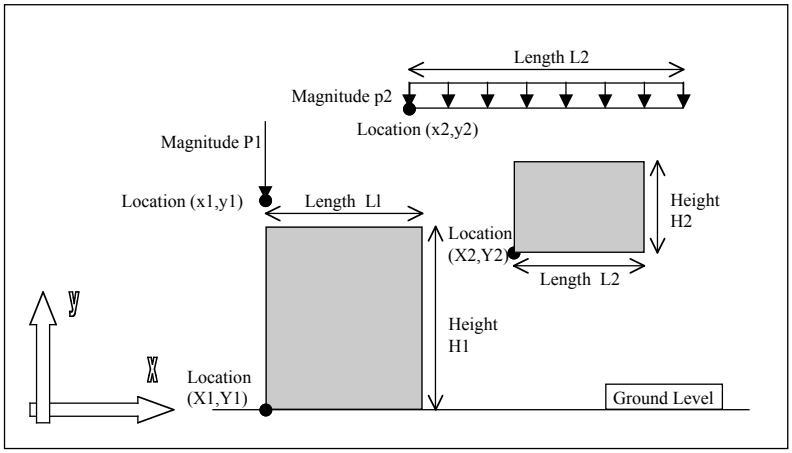

Figure 5: A specification of a structural system layout problem

3.1 Problem Space Representation

The problem space contains a population of individuals, where each individual represents a problem. As shown in Figure 6, each individual has a name, a phenotype, a genotype, and (after it has been evaluated) a fitness value. Additional information about its parents and the way it was created is also included. The method of creation can be crossover, mutation, or problem space initialisation.

Individual					
name	created-by	parents	fitness	phenotype	genotype

Figure 6: Individual problem object

The problem phenotype of an individual problem is a list of problem phenes (we use the word phene to refer to an element of a phenotype, analogous to the use of the word gene to signify an element of a genotype). A problem phene can represent a load or a constrained free space. It is possible for a phenotype to have either only loads or only constrained free spaces, but typically loads and constrained free spaces are combined. The phenes within a phenotype do not have to be ordered in any particular way, and there can be any number of phenes within a phenotype. The representation of an example of a phenotype is shown in Figure 7. For this demonstration, the genotype of a problem is exactly the same as the phenotype of the same individual problem.

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

Figure 7: Representation of a possible problem phenotype

A load is represented by its location and its length of distribution. If the length of distribution is zero the load is a point load; otherwise the load is a distributed load. The content of a load phene structure object is shown in Figure 8. The location describes the grid point where a point load acts or the leftmost grid point of application of a distributed load.

Load phene	
Location: grid point	Length of Distribution

Figure 8: Load phene structure object

The representation of a constrained free space phene includes its location, its length, and its height. The content of a free space phene object is shown in Figure 9.

Constrained Free space phene		
Location : grid point	Length	Height

Figure 9: Constrained free space phene object

3.2 Solution Space Representation

The solution space also contains a population of individuals. The solution phenotype of an individual solution is a list of elements containing any number of solution phenes as shown in Figure 10.

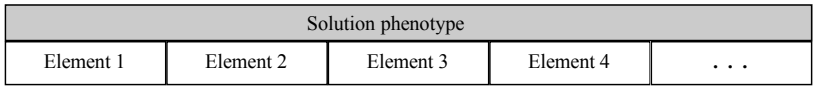

Figure 10: Solution phenotype

A solution phene represents a horizontal, a vertical, or a diagonal design line element. Each element is characterised by the locations of its end-points, using their absolute grid coordinates as shown in Figure 11.

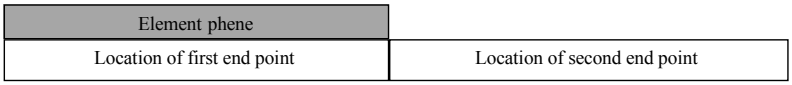

Figure 11: Element phene structure object

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

The genotype of a solution individual is not equivalent to the phenotype of the individual. The solution genotype is a list of points, each of which is a gene. The first point is an absolute location on the ground (y=0), and each of the following points is described relative to the previous point. Each pair of adjacent points represents an element which is placed between them. It can happen that an element appears two or more times in a genotype because of the use of relative coordinates, but this repetition can be eliminated during genotype-to-phenotype mapping. Relative grid points are needed to avoid producing a lot of useless or meaningless solutions when the crossover mechanism is applied. The representation of a solution genotype is shown in Figure 12.

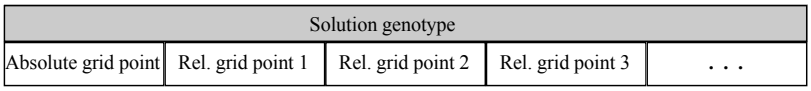

Figure 12: Representation of a solution genotype

3.3 Fitness Evaluation

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

In this application, we define the fitness function as a set of constraints on the combined problem/solution pair. When evaluating the fitness of alternatives in the problem space, we combine each alternative problem individual with the current best solution. When evaluating the fitness of alternatives in the solution space, we combine each alternative solution with the current best problem. For the structural system layout domain we have implemented 15 constraints. Some of these constraints are only valid for evaluating problems, some are only used for evaluating solutions, and some are equally applicable to both problems and solutions.

An example of a constraint on the design solution is: “each vertical element must have a support beneath it, which can be either another vertical element or the ground”. An example of a constraint that checks how well the current design problem is solved by the best design solution is: “there must not be a design element in a constrained free space”. Each constraint returns a value between 0 and 1, reflecting the quality of the individual according to the constraint. If the constraint is fully satisfied the returned value is 1, if the constraint is not satisfied at all the returned value is 0, and if the constraint is satisfied by only some of the components of the individual the returned value is between 1 and 0, according to the degree of satisfaction. The final constraint value for the phenotype is the average of the constraint values for each phene in the phenotype being evaluated. This approach to a fitness function is similar to multi-criterion optimisation using weighted constraints.

3.4 Implementing the Co-evolutionary process

Initialisation: A single problem individual and a population of solution individuals initialise the problem space and solution space, respectively. Search begins in the solution space, using the initial problem as the set of requirements used to evaluate the potential solutions generated during the search. In the demonstration results presented below, the initial solution space has 20 different solutions. The phenotype of each individual solution represents a possible structural system. The initial problem represents the load and space requirements for the structural design of a typical tunnel or overpass.

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

Convergence: Convergence can be determined genetically, optimally, or by number of generations. Convergence is determined genetically by keeping track of which genotypes have been encountered before and during each GA cycle. The search converges or ends when most genotypes generated in the current cycle have already been encountered. The threshold for ending the search is the repetition percentage parameter. Convergence is determined optimally by ending the search when an individual has been found that has a fitness value of 1. Convergence is determined by number of generations when the v parameter reaches a specified value. This last method of determining convergence is the one we used in the demonstration results given below.

Interaction: Interaction in this demonstration occurs after convergence in the solution space. The 20 individuals in the solution space are the basis for contributing alternative problems to the problem space. Each of the best solutions produces one alternative problem using the following method: a new problem is defined by a point load for each vertical element in the solution, a distributed load for each horizontal element, and a constrained free space for each open space (*i.e.*, a space without diagonal members) below each horizontal element (if the horizontal element is not supported on the ground). This interaction provides a mechanism for the best solutions to influence the evolving problem specification.

Termination: Termination occurs when the co-evolution process ends. Where convergence determines when the search in one space ends so that search in the other space can begin, termination determines when search in both spaces ends. Termination can be determined using optimality criteria (when a given problem or solution achieves a fitness value of 1 during a given search), by no change from one phase to another (when problems or solutions being generated have all been encountered in previous co-evolutionary cycles), or by the number of co-evolutionary cycles. In this demonstration, we terminate the co-evolutionary process after a fixed number of co-evolutionary cycles, indicated by the parameter T.

3.5 Results

We performed two experimental runs on our implementation of the CoDESIGN algorithm. The two runs, including an indication of the parameter settings that we varied in each one, are given below. The parameter settings that we didn't change from one run to the other are the following:

Initial solutions: 20
Initial problems: 1
Reproduction percentage: 100%
Selection percentage: 50%
Individuals and genes to cross and mutate: chosen at random

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

Convergence at v=1
Termination at T=10

This run forces convergence to occur after only one generation in each evolutionary algorithm within the co-evolutionary algorithm. We call this tightly-coupled co-evolution, which is similar to the way in which Biological populations continuously influence each other as they evolve over time. The initial problem and the best solution and problem at the end of each evolutionary search in our run is shown in Figure 13. The first solution found and the initial set of requirements are shown at the bottom of the figure, with the resulting solutions and problems of subsequent co-evolutionary cycles shown further and further up the figure. The co-evolutionary process in this run produces a variety

of solutions and alternative requirements. The final solution and requirements do not have much in common with the initial ones. This is an appropriate exploratory process, where each phase converges after one generation, when the goal of the design process is to identify a broad range of requirements and solutions.

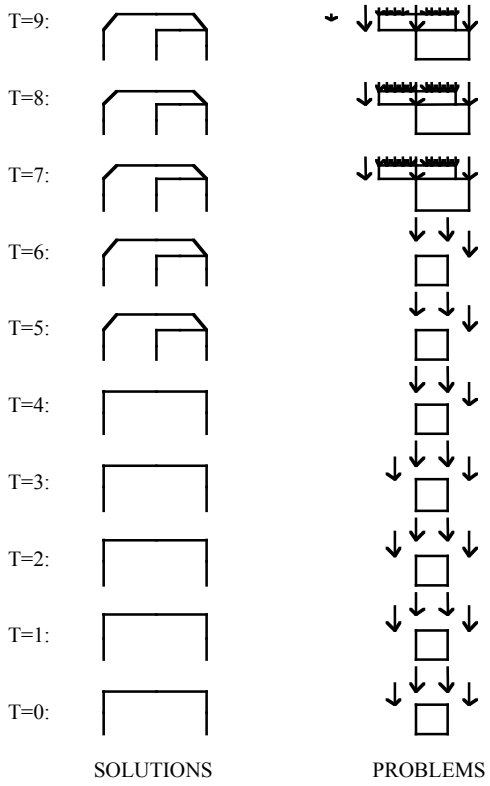

Figure 13: Results of 10 co-evolutionary cycles for run 1

Run 2: In the second run, we used the same parameters except for the convergence and termination criteria: this time we let the search in each space continue for 20 generations and stopped the co-evolutionary process after only 6 cycles when it became evident that nothing new was happening. The parameters for this run are:

Convergence at v=20
Termination at T=6

This run allows each population to evolve on its own for several generations before stopping the search and introducing the influence of the other population, in what we can call loosely-coupled co-evolution. The initial problem and the best solution and problem at the end of each co-evolutionary cycle of the run are shown in Figure 14. Again, the solution found after the first search in the solution space and the initial set of requirements are shown at the bottom of the figure, with the resulting solutions and problems of subsequent co-evolutionary cycles shown further and further up the figure. The major difference between this run and the previous is the number of generations in each phase, where this run allows the search to continue for 20 generations. The process shows early convergence, although not to the initial requirements. The requirements and solution change between the first and second co-evolutionary cycles and then there is little change over the next five cycles.

3.6 Discussion

In this demonstration we observed the behaviour of the algorithm to determine whether alternative problem specifications are formulated and if the behaviour can be loosely described as exploratory and creative. The results from these two runs show that a co-evolutionary model of design does allow the problem specification to change in response to the solutions (with the subsequent proposal of different solutions that correspond to the new problem specifications). In both runs, the problem continues to change to better fit the current best solution. In the first run there is more exploration of alternative solutions because the GA did not optimise the solution space. In the second run, with 20 GA cycles in each space before searching the other space, the solution and problem specification do not change after the first cycle. An open issue is the question of comparing loosely- vs. tightly-coupled co-evolution, and finding the “right” balance between the two modes.

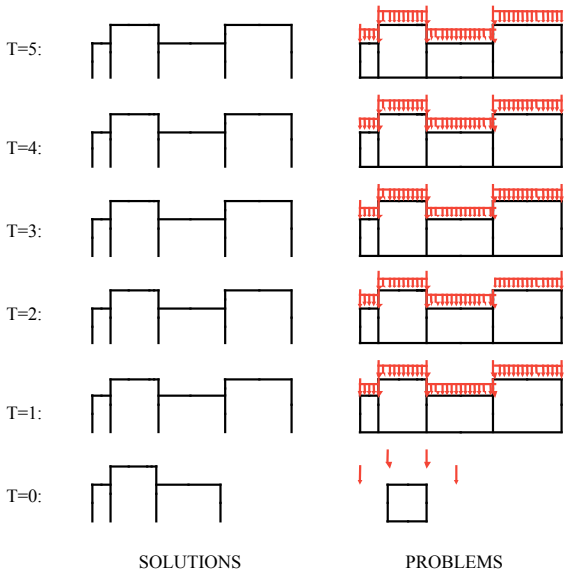

Figure 14: Results of 6 co-evolutionary cycles for run 2

4 Summary

This paper presents co-evolutionary design as a computational model that can potentially produce creative design solutions through an unexpected change in the design specifications. The model is elaborated as a general algorithm for co-evolution and applied to a structural layout problem. By considering co-evolution as a two phase evolutionary process, traditional evolutionary computing can be considered as a case of co-evolution.

Our model of co-evolutionary design has the following features:

- Co-evolutionary design consists of two spaces/populations: design specifications and design solutions.
- Co-evolutionary design consists of two iterative phases: search for design specifications and search for design solutions.
- The focus of search in one space is partially determined by the current makeup of the population in the other space.
- Interactions may add new variables to either space, which may lead to initially unexpected design requirements or solutions.
- Genetic changes in co-evolutionary design consist of two aspects: changes in design focus and changes in genotype.
- Fitness is local and changes in each phase.
- The fitness value is not comparable across different phases.
- There is no relationship between convergence and fitness: fitness is used to determine which individuals survive and convergence occurs when new ideas can not be found within each phase of co-evolution.
- The termination conditions of co-evolution do not rely on the fitness of individuals.

The model of evolutionary design has been used to explain real design projects such as the Sydney Opera House [6] and as the basis for developing a computational model of the design of braced frames [7] and floor plans [8].

Acknowledgements

This work is supported by a grant from the Australian Research Council. Josiah Poon and Peter Wu contributed to the development of the general CoDESIGN algorithm. The development of the structural system layout problem was done with the collaboration of Anja Wetzel and Michael Tang.

References

1 Maher, M.L. (1994) Creative Design Using A Genetic Algorithm. Computing in Civil Engineering, American Society of Civil Engineers, 2014-2021.

2 Paredis, J. (1995) Coevolutionary Computation, Artificial Life, 2:355-375.

3 Potter, M.A., De Jong, K.A. (1994) A Cooperative Co-evolutionary Approach to Function Optimization. In Y. Davidor, H-P. Schwefel, and R. Manner (eds.), Proceedings of the Third Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science Vol. 866, Springer-Verlag, Berlin, 249-257.
4 Maher, M. L. (2001) A Model of Co-evolutionary Design, Engineering With Computers.
5 Mitchell, M. (1996) An Introduction to Genetic Algorithms, MIT Press, Cambridge, Mass.
6 Poon, J., Maher, M.L. (1997) Co-evolution in Design: A Case Study of the Sydney Opera House. In Y.T. Liu, J.Y. Tsou, and J.H. Hou (eds.) Computer-Aided Architectural Design Research in Asia (CAADRIA '97), Hu's Publisher, Taipei, Taiwan, 439-448.
7 Maher, M.L., Poon, J., Boulanger, S. (1995) Formalising Design Exploration as Co-evolution: A Combined Gene Approach. In J.S.Gero and F. Sudweeks (eds.), Advances in Formal Design Methods for CAD, Chapman & Hall, 1-28.
8 Maher, M.L., Poon, J. (1996) Modeling Design Exploration as Co-evolution, Microcomputers in Civil Engineering, 11:195-210.