Annals of the CIRP, Vol.54/1, pp.155-158, 2005

A Hierarchical Co-Evolutionary Approach to Conceptual Design

Y. Jin, W. Li, S.C-Y. Lu (1)
IMPACT Laboratory, Department of Aerospace and Mechanical Engineering
University of Southern California, Los Angeles, USA

Abstract
Conceptual design is a key early activity of product development. Limited understanding of conceptual design process and lack of quantitative information at this stage of design pose difficulties on effective support for concept generation and evaluation. In this paper, a hierarchical co-evolutionary approach is proposed to support conceptual design. In this approach, higher-level functions are decomposed based on a set of grammar rules, and mappings between functions and their solutions, or means, are realized through a co-evolutionary computing process. The paper describes the details of the approach. An example of designing a mechanical personal transporter is presented to demonstrate the effectiveness of the proposed approach.

Keywords:
Conceptual design, design automation, genetic algorithm

1 INTRODUCTION

Conceptual design is a major early activity of engineering design that involves generation and evaluation of design concepts [1]. Due to its creative nature, providing effective tool support for designers at this stage of design has been a highly difficult task. Effort was made to provide information management support for designers to managing their design information. But this approach cannot effectively help designers to extend their reach in the design space for more ideas. Other researchers are more aggressive and attempt to generate design concepts automatically for designers [2].

There are two major issues that must be addressed to achieve effective support for conceptual design. First, the concept generation process of conceptual design is very much unknown. Although researchers have developed various design methods [1] [3], they only specify the steps that designers should follow. Designers must rely on their own experience and expertise to generate concepts and select the best ones. Cognitive models of conceptual design process have been explored recently [4], but it will still take time for these models to be useful for building computational design support tools.

Another important issue is the lack of quantitative information at conceptual design stage [1]. The concepts and information involved in conceptual design are related to function requirements and solution principles, which are usually highly qualitative. This situation makes it difficult to establish evaluation criteria to support conceptual design.

Recent progress in evolutionary computing [5] [6] has attracted researchers' attentions because of its great potential for aiding idea generation. The existing evolutionary design research, however, is limited to dealing with parametric design problems [7] or relatively simple architectural design problems [8]. Grammar based approach [2] has been applied to provide computational support for conceptual design, but they do not explicitly provide criteria for design concept evaluation. Due to the two issues mentioned above, thus far there has been no effective ways for conceptual design support.

The goal of our research is to develop a hierarchical co-evolutionary approach that supports conceptual design by providing automated exploration of design space and generation of design concepts. In this approach, function grammar guides function decomposition, and co-evolution of functions and means leads to best solutions at each layer of decomposition. A co-evolutionary algorithm based on this model is developed for concept generation. In the following, we first provide an overview of the proposed approach. After that, details of the approach are presented in Sections 3 and 4. A case example is presented in Section 5, and conclusions drawn in Section 6.

2 A HIERARCHICAL CO-EVOLUTION APPROACH

Mechanical design problems are usually complex and involve various functional requirements and a large number of potential means as solutions. Decomposing high level complex functions into lower level ones has been a common practice to create design concepts [1] [3]. In his axiomatic design framework, Suh [3] proposed a zigzag design process in which before a *FR1* (functional requirement) is decomposed, a designer is advised to first find a suitable *DP1* (design parameter) so that the information of *DP1* can be used to guide the decomposition of *FR1*. After *FR1* is decomposed into *FR1-1, ..., FR1-n*, then these sub *FRs* can be used to find suitable sub *DPs* for *DP1.* While the axiomatic design suggests the zigzag process, it is designers' job to perform the decomposition of *FRs* and identification of suitable *DPs*. Our research question is *how can we devise a computational support that can help designers 1) decompose functions, 2) identify most suitable means to carry the functions, and 3) generate most desirable design concepts*?

To address this question, we turn to evolutionary computing techniques [5] [6] for solutions. The evolutionary approach is adopted because it can address the two major issues mentioned above. First, *evolution* can be applied as an underlying process for idea generation, and second, the concept of *fitness* can be used to maximize the utility of available information for concept evaluation.

Figure 1 illustrates our proposed evolutionary framework for conceptual design. At the center is the evolutionary computing based on a co-evolutionary design *process* and evolutionary *algorithms.* Function and means *library* serves as the design space in which specific functions and means evolve to form final design concepts. The *fitness functions* are composed of general principles and the information acquired during design. They serve as evaluation criteria to guide evolution. After a list of requirements and a top-level function are given, the function will be decomposed, and more concrete functions and means will evolve to eventually form complete design concepts.

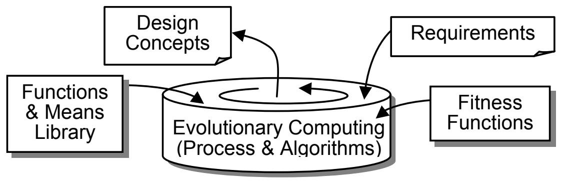

Design Concepts
Requirements
Functions & Means Library
Evolutionary Computing (Process & Algorithms)
Fitness Functions

Figure 1: Evolutionary approach to conceptual design

It is conceivable that generating meaningful concepts requires a rich *library* of functions and means. How to create such a rich library is a future research topic. In this research, a limited library is predefined for testing purpose, and we focus on developing the evolutionary *process*, *algorithms*, and *fitness functions*.

Modelling conceptual design as an evolution process of functions and means is a major challenge. We follow the zigzag process of axiomatic design. A grammar-based approach is adopted for function decomposition, and a co-evolution process of functions and means is introduced for automated function structuring and function-means mapping at each layer of decomposition. The process of our model can be described as follows (see Figure 2):

(0) A designer provides top-level function f_0 based on given design requirements. From f_0, top-level means m_0 is found through search based means selection.
(1) Decompose f_0 into N decomposition sets based on grammar rules and information of m_0.
For each decomposition set, do the following:
(2) Find all feasible means for each of the sub-functions based on a search algorithm.
(3) Compose function structures with genetic programming (GP) and feasible means set information.
(4) Select most suitable means for each sub-function based on genetic algorithm (GA) and function structure information.
(5) Repeat (3) and (4) until a high fitness function value can be reached.
(6) Repeat (1) through (5) if the means identified in (5) are still not implementable.

This evolutionary design process has two important features. First, it is *hierarchical* in the sense that higher-level functions are decomposed into lower level ones for finding most suitable means. Second, the functions and means are *co-evolving* in that the selected means sets are used to evaluate the fitness of function structures, and the functions structures in turn are also used to further select

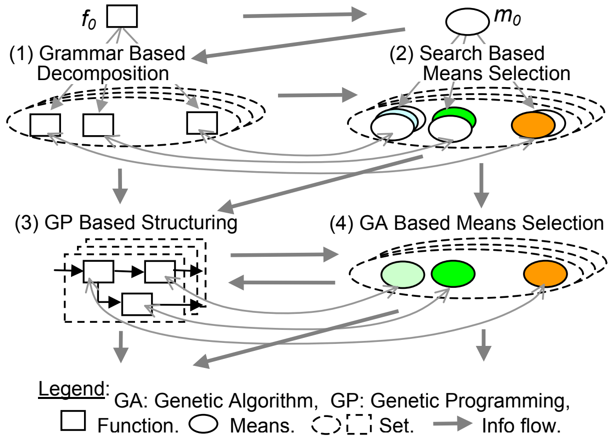

f_0
m_0
(1) Grammar Based Decomposition
(2) Search Based Means Selection
(3) GP Based Structuring
(4) GA Based Means Selection
Legend:
GA: Genetic Algorithm, GP: Genetic Programming,
Function.
Means.
Set.
Info flow.

Figure 2: Hierarchical co-evolution of functions & means

suitable means. We call this process a Hierarchical Co-Evolutionary approach to conceptual Design, or HiCED.

From a computational point of view, it is conceivable that at each layer of decomposition hierarchy, there can be a huge number of possible decompositions. While human designers can do the decomposition and function-means mapping based on their experience, automating this process requires a powerful search mechanism. In our proposed process model, a co-evolutionary approach is devised to let functions and means find each other in an evolutionary way. In the following two sections, we describe some of the details of grammar-based function decomposition and GA & GP based co-evolution of functions and means.

3 GRAMMAR BASED FUNCTION DECOMPOSITION

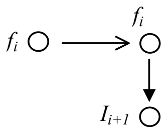
Grammar based function decomposition is carried out by applying a set of grammar rules [2]. As in any rule-based systems, effective grammar based decomposition depends on the balance between generality and specificity of the grammar rules since too specific will limit applicable domains and too general may lead to large amount of incompatible decompositions. In HiCED, we follow a number of general grammar rules [2], and a set of specific rules created based on our following function definition.

$$f = \{<action><object>, \{input\_flow\}, \{output\_flow\}\} \quad (1)$$

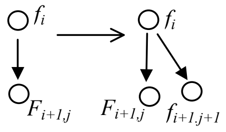
In definition (1), $action$ is the operation to be performed by this function, $input\_flow$ and $output\_flow$ are flows of energy, material and signal [1]. $Object$ is part of $input\_flow$ and represents the object to be processed by this function. In the following, we present both general rules and action-based rules. The application of these rules will be discussed in Section 5.

3.1 General grammar rules

The general function decomposition rules are applied to function decompositions where no action based rules can be matched. Thus general rules have lower priority.

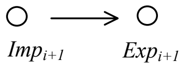


Initial Rule (IR): f_i is a function at the i^{th} decomposition layer, and I_{i+1} is an initial decomposition function set for function f_i. The result of this rule is that it adds a function *Import* for each $input\_flow$ of f_i and a function *Export* for each $output\_flow$ of f_i.



General Expansion Rule (GER): $F_{i+1,j}$ is an incomplete function decomposition set for function f_i at layer $i+1$. j is the number of sub functions in function set $F_{i+1,j}$. The result of this rule is the addition of $f_{i+1,j+1}$ to initial incomplete set $F_{i+1,j}$.

A major issue in applying this rule is how we can find a function $f_{i+1,j+1}$ to make a new function decomposition set that can better fulfill the function f_i. A greedy search algorithm is employed that selects best matching functions based on compatibility of $input$ and $output\ flows$.



Termination Rule (TR): For any function f_i at i^{th} layer, Imp_{i+1} and Exp_{i+1} are the *Import* function set and *Export* function set of f_i, respectively, at $(i+1)^{th}$ layer. If for all *output_flows* of any function in Imp_{i+1} there is a *flow* path that can reach one of the functions in Exp_{i+1}, and vice versa, then the decomposition from i^{th} layer to $(i+1)^{th}$ layer terminates.

3.2 Action based grammar rules

From definition (1) we can see that decomposition of a function can be along the $action$ dimension. In HiCED we introduced following two action-based grammar rules.

Action Decomposition Rule (ADR): In case ACT_h can be divided into sub-actions $[ACT_{l1},...,ACT_{ln}]$, then we have:

$$\begin{aligned} &if \quad f = \langle ACT_h \rangle \langle object \rangle \text{ and } ACT_h = \{ACT_{l1},...,ACT_{ln}\} \\ &\Rightarrow f \xrightarrow{decomp} \{f_1,...f_n\} \; where \;\; f_i = \langle ACT_{li} \rangle \langle object \rangle, \; (1 \le i \le n) \end{aligned} \tag{2}$$

Action Expansion Rule (AER): When execution of ACT_1 either depends on or leads to another action ACT_2, then,

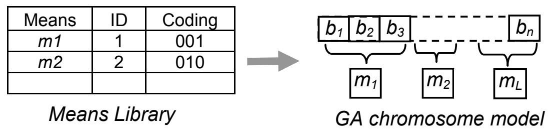
$$\begin{aligned} &if \quad f_1 = \langle ACT_1 \rangle \langle object \rangle \text{ and } ACT_1 \Rightarrow \{ACT_2, ACT_1\} \\ &\Rightarrow f_1 \xrightarrow{\exp and} \{f_1, f_2\} where \;\; f_2 = \langle ACT_2 \rangle \langle object \rangle \end{aligned} \tag{3}$$

4 CO-EVOLUTION OF FUNCTIONS AND MEANS

In HiCED, at each layer of decomposition hierarchy, function and means co-evolve until some satisfactory solutions are found. We developed a genetic algorithm (*GA*) for means selection and genetic programming (*GP*) for function structuring. Fitness functions are composed to evaluate function structures and means selections.

4.1 Co-evolution algorithm

In our co-evolutionary algorithm, means are pre-encoded into strings of binary bits, as shown in Figure 2. Each chromosome [5] represents a combination of means for corresponding functions. The length of bits for m_i is determined by the number of matchable means of the corresponding function f_i.



Means	ID	Coding
m1	1	001
m2	2	010

Figure 2: A chromosome model of means

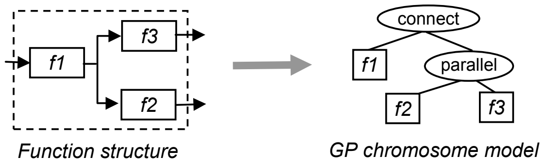


Figure 3: A chromosome model of function structure

In function structuring by GP, each function structure is represented as a genetic programming tree, where internal nodes are genetic programming functions, and terminals are functions that are derived from the function decomposition, as shown in Figure 3. At present, two basic genetic programming functions are used, namely, *connect* and *parallel*, as shown in Figure 3. More functions will be added as our research advances.

Following is pseudo-code of our co-evolutionary algorithm:

```
Given requirements Q, toplevel functin f0;
S = InitializeDesignSolution (Q, f0);
if S is not implementable, do iteration
{
  FDS = GetFunctionDecompositionSets(S, Q);         //Grammar
  CMS = GetCorrespondingMeansSets(FDS) ;            //Search
  PFS = GetPreliminaryFunctionStructureSets (FDS); //GP
  PMS = GetPreliminaryMeansSets(PFS);               //GA
  FsMS =GetFuncStructure-MeansPairSet(PFS, PMS);
          //The above line involves GP&GA based co-evolution
  S = GetTheOneWithHighestFitnessValue(FsMS );
} End;
```

4.2 Fitness functions

Fitness functions in evolutionary computing are used to evaluate candidate solutions. The fitness value attached to each candidate determines whether the candidate, or part of it, will go to the next generation. In HiCED, fitness functions must make maximum use of design principles and all qualitative information since no much quantitative or performance related information is available.

Explicitly modelling function structures, flows, and their mappings to means allows us to compose fitness functions based on *input* and *output flow connections*—i.e., how well two functions or two means are connected to each other—and structural *dimensions*—i.e., how many functions or means are involved. In addition to these structural evaluations, performance based fitness is also included in case quantitative information is available. In the following, we briefly present the fitness functions of HiCED.

Fitness functions for GP-based function structuring

Function connection rules: These rules are applied to check the compatibility of function connections. Three sub rules are defined: *function connection rule (FCR)* for checking flow connections of two functions; *means connection rule (MCR)* for checking corresponding means connection compatibilities; and *input/output rules (IOR)* to check if the candidate structure has the same I/O flows as the top-level function. For functions f_i, f_j with matching means m_i, m_j in structure F_k, we have

$$ff_{FCR} = -\sum unmatch\_flow(f_i, f_j) \tag{4}$$

$$ff_{MCR} = -\sum unmatch\_means(m_i, m_j) \tag{5}$$

$$ff_{IOR} = -unmatch\_flow(f_0, F_k) where \; f_0 \, is \, top \, level \, function \tag{6}$$

ff_{MCR} reflects the fact that means information is used to evaluate function structures in co-evolution.

Structure dimension rule (SDR):This function controls size of function structures. Generally, smaller size is preferred.

$$ff_{SDR} = -w \cdot number\_of\_function(\langle F_k \rangle) \tag{7}$$

Fitness function for GA based means selection

Means connection rule (MCR): This rule checks the compatibility of means connections. Given m_i, m_j we have

$$ff_{MCR} = -\sum unmatch\_means(m_i, m_j) \tag{8}$$

The means connection information comes from function structures. This is how function information is used to evaluate means in co-evolution.

Means performance rule (MPR):If performance information of certain means is available, e.g., maximum power, the information is used as part of fitness functions. Let design requirements be Q={q_i, ..} and performance P={p_i, ..}, then,

$$ff_{MPR} = -\|Q - P\| = -\sqrt{\sum_j (q_i - p_i)^2} \tag{9}$$

Means preference rule (MPRR): Designers can assign specific preference pr_i to certain means m_i. This rule indicates the likelihood a means is selected in a design.

$$ff_{MPRR} = \sum_i pr_i \cdot b_i \;\; where \;\; b_i = \begin{cases} 1 & if \;\; m_i \; is \; selected \\ 0 & if \;\; m_i \; is \; not \; selected \end{cases} \tag{10}$$

In the following section, an example of designing a simple mechanical transporter is presented to show how the grammar rules, algorithms, and fitness functions work together for design concept generation and evaluation.

5 A CASE EXAMPLE

The task in this example is to design a function structure with matched means for a simple *mechanical transporter*. The top-level function for design is <*transport*><*X*>, where *X* stands for any object.

Examples from the function and means library used for mechanical transporter design are shown in Table 1 and Table 2. The functions are chosen from the common function basis defined in [9]. In Table 1, <E> refers to energy and <ME> to mechanical energy.

In Table 2, *application functions* contain functions that the means can fulfil; *high-level means* are the means of which this means is a subtype; and *attributes* such as *weight* and *cost* can are included. The *applicable functions* can help mapping from function to means, and the *high level means* set can be used to determine whether the function that this means is supposed to satisfy is still decomposable.

ID	Function	ID	Function
f_1	<transport><X>	f_8	<input><E>
f_2	<generate><ME>	f_9	<secure><X>
f_3	<stop><ME>	f_{10}	<change><E>
f_4	<guide><ME>	f_{11}	<transmit><E>
f_5	<support><X>	f_{12}	<transmit><ME>
f_6	<supply><E>	f_{13}	<convert><E to ME>
f_7	<control><E>	f_{14}	<move><X>

Table 1: Part of function library

ID	Means	Applicable function	High level means	Weight value	Cost value
w_2	Pedal drive	f_2	Φ	n/a	n/a
w_{14}	Pedal gear	f_{10}	w_2	3	3
w_{20}	Pedal	f_{13}	w_2	1	1
w_{26}	Chain	f_{12}	w_2	2	2
…	…	…	…	…	…

Table 2: Part of means library

General and action-based grammar rules are used in the example. The later are listed below. The abbreviations *ADR* and *AER* are the types of rules discussed in Section 3. Part of the decomposition hierarchy with applied rules is shown in Figure 4.

R1: $\langle transport\rangle\langle X\rangle \xrightarrow{ADR} \{\langle move\rangle\langle X\rangle, \langle \sup port\rangle\langle X\rangle\}$

R2: $\langle move\rangle\langle X\rangle \xrightarrow{ADR} \{\langle generate\rangle\langle ME\rangle, \langle guide\rangle\langle ME\rangle, \langle stop\rangle\langle ME\rangle\}$

R3: $\langle generate\rangle\langle ME\rangle \xrightarrow{AER} \{\langle \sup ply\rangle\langle E\rangle, \langle generate\rangle\langle ME\rangle\}$

R4: $\langle guide\rangle\langle ME\rangle \xrightarrow{AER} \{\langle generate\rangle\langle ME\rangle, \langle guide\rangle\langle ME\rangle\}$

R5: $\langle stop\rangle\langle ME\rangle \xrightarrow{AER} \{\langle generate\rangle\langle ME\rangle, \langle stop\rangle\langle ME\rangle\}$

R6: $\langle \sup ply\rangle\langle E\rangle \xrightarrow{ADR} \{\langle control\rangle\langle X\rangle, \langle input\rangle\langle X\rangle\}$

R7: $\langle import\rangle\langle E\rangle \xrightarrow{AER} \{\langle import\rangle\langle E\rangle, \langle change\rangle\langle ME\rangle\}$

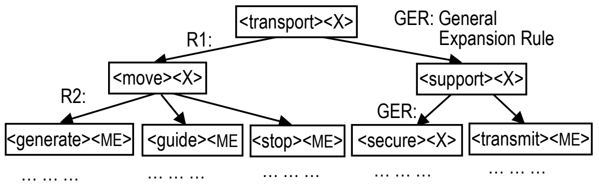


Figure 4: Part of the decomposition hierarchy

With the functions, means, and grammar rules defined in this example case, there can be approximately 430 possible function decomposition sets.

The co-evolution algorithm described in Section 4.1 searches through this huge space. In our test, we set population size to 100, and applied a fitness function that is the weighted summation of fitness functions (4) through (10). One satisfactory function structure together with the function-means mapping is shown in Figure 5 and Table 3.

Function	Means
<control><E>	human
<input><E>	pedal, handlebar, cramp brake
<change><E>	pedal gear, handlebar, lever
<convert><E to ME>	pedal gear, handlebar, cramp brake
<transmit><ME>	frame and wheel, chain wheel, chain, shaft
<stop><ME>	rubber brake
<guide><ME>	wheel
<secure><X>	saddle

Table 3: A means selection result by GA

The convergence curves are in Figure 6. It shows that the best solutions emerge from the 10th generation when selecting suitable means and best function structures are developed after the 45th generation. The total computer elapse time is less than 5 minutes on a Pentium 4 & 2.0GHz PC. Total number of possible solutions examined through this process is well over 1 million.

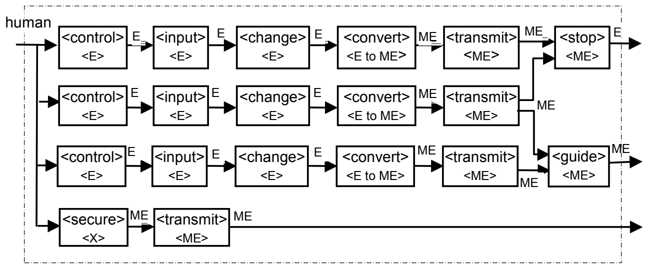


Figure 5: A satisfactory function structure result

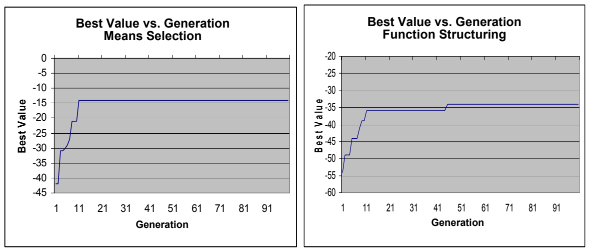


Figure 6: Convergence curves

6 CONCLUDING REMARKS

As design problems become more complex and design lead time more pressing, designers need supporting tools to expand their reach in the design space and increase number and quality of their design concepts. Our research takes a hierarchical co-evolution approach to help designers explore design space and develop design concepts by automatically generating desirable function structures and their mappings to embodiment means. The approach adopts a zigzag design process in which grammar rules are applied to decompose higher level functions and GA and GP based algorithms are employed to let function structures and means co-evolve into design concepts. The concepts of *evolution* and *fitness* of this approach worked well to deal with the two major issues, i.e., unknown idea generation process and lack of quantitative information at the stage of conceptual design.

Modelling conceptual design as a co-evolutionary process and composing a set of effective fitness functions were two major challenges in our research. The co-evolution results are sensitive to genetic functions (we used only *connect* and *parallel* in the case example), grammar rules, and fitness evaluation. Our current research explores more genetic functions and fitness measures.

7 REFERENCES

[1] Pahl, G. and Beitz, W., 1996, Engineering Design – A Systematic Approach, Springer.

[2] Schmidt, L., Cagan, J., 1997, "A Graph Grammar-Based Machine Design Alg." Res. in Eng. Design, 7.

[3] Suh, N.P., 2001, Axiomatic Design – Advances and Applications, Oxford University Press, New York.

[4] Cross, N., Christianns, H.& Dorst, K., 1997, Analyzing design activity, John Wiley & Sons, New York, NY.

[5] Goldberg, D.E., 1989, Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley.

[6] Koza, J.R., 1992, Genetic Programming. MIT Press.

[7] Roy,R. Tiwari,A. Corbett,J.2003, Designing a Turbine Blade Cooling System Using a Generalised Regression Genetic Algorithm. *CIRP Annals* 52/1,p.415-418

[8] Maher, M.L., 2001, "A Model of Co-Evolutionary Design," Eng. with Computers, 16:195-208.

[9] Stone, R. Wood, K., 2000 "Developmt of Functional Basis for Design," J. Mech. Design, 122(4):359-370.