Design Concept Generation: A Hierarchical Coevolutionary Approach

Yan Jin
e-mail: yjin@usc.edu

Wei Li

Department of Aerospace and Mechanical Engineering,
University of Southern California,
Los Angeles, CA 90089-1453

As design problems become more complex and design lead time more pressing, designers need effective support tools to expand their design space exploration. In this paper, a hierarchical coevolutionary approach is proposed to support designers by automatically generating design concepts based on the designers' inputs. The approach adopts a zigzag design process in which function structures and their corresponding solution principles coevolve in parallel across different levels of an abstraction hierarchy. A grammar-based approach is applied to decompose higher-level functions and generate an initial population of function structures. From this initial population, a coevolutionary algorithm is devised to coevolve more function structures and their corresponding solution principles. A case study of designing a mechanical personal transporter is presented to demonstrate the effectiveness and features of the proposed approach. [DOI: 10.1115/1.2757190]

1 Introduction

Conceptual design is a complex process and involves generating and selecting design concepts to satisfy given design requirements. Designers must find effective and efficient ways to fill the gap between the design requirements and available solution technologies. A general approach to design complex mechanical systems is to break a complex design problem into more manageable subproblems, solve the less complex subproblems, and then develop the overall design by synthesizing the subsolutions [1,2]. This divide-and-conquer process usually involves multilayered problem and solution spaces in which various functions and solution principles are explored and design concepts generated.

To develop computer tools for supporting concept generation, two important issues must be addressed. First, the current understanding of the design concept generation process is limited, and this lack of understanding makes it difficult to apply any existing design process model directly to computer based concept generation. Extant design methods [1–4] prescribe how design should be done, but provide little guidance for how to generate design concepts. The interest in cognitive models of conceptual design has grown recently [5–7], but it is still too early for these models to be useful for building computational design tools. Another important issue related to providing conceptual design support is the lack of *quantitative* information for evaluating design concepts. The design space is open, but the useful information for design evaluation is very limited. This situation makes it difficult to form effective evaluation criteria for design concept generation.

Our research attempts to develop a computational tool to help designers effectively explore their design space by automatically generating design concepts based on the inputs from the designers. In our proposed hierarchical coevolutionary design (HiCED) approach, a *design concept* is a conceptual realization of a product and is defined as a pair of a *function structure*, composed of multiple functions and the energy, material and signal flows between them, and a set of means, called a *means combination*, that performs the functions and transforms the flows. A *function* is an identified purposeful transformation of given input flow(s) into the output flow(s) and a *means* is a working principle [1] or a technology solution [8] that achieves a given function. A means is not a physical *component* that usually does not have dimensional and other physical specifications. A means is *implementable* if a specific physical component can be found to embody the means. Since we do not assume the availability of the physical component information, how to evaluate the generated design concepts is a challenge for this research.

Built on the previous research [1,2,9–14], our proposed HiCED approach focuses on functions and means and views conceptual design as a process of coevolution of both function structures and means combinations across different levels of an abstraction hierarchy. By *coevolution*, we mean that the function structures and their corresponding means combinations evolve in parallel by using each other as their evaluation focus, i.e., function structures are used to evaluate and select means, and the selected means in turn are used in the evaluation criteria for further evolving function structures. This coevolutionary computational process model for design concept generation was inspired by the *zigzag* design process [2], in which functions and corresponding means are decomposed and structured in parallel. To make this hierarchical coevolutionary process computable, we combine the zigzag design process with a grammar and evolutionary computing based algorithm that continuously generates function structures and searches for best matching means along the abstraction hierarchy, from more abstract to more concrete levels, until "best" implementable design concepts are found. The evaluation of design concepts is based on the *fitness functions* associated with the evolutionary algorithm and is part of the design concept generation process.

The basic components of HiCED are the following: (1) a function and means library, (2) a coevolutionary mechanism, and (3) a number of fitness functions, as shown in Fig. 1. The function and means library serves as a knowledge base of what can be specified as functions, and what are the possible *technologies* or *solutions* to achieve these functions. It provides a basis for decomposing functions and coevolving function structures and their corresponding means combinations. The richness of the library determines the size of the design space. In our research, we expect that eventually an agent-based system can be developed to fuel the library by collecting function and means information from designers' design logs, and relevant databases and websites. Since the focus of our current research is on the other two components, we created a library for the purpose of testing.

The *coevolutionary mechanism* provides procedures for effective design concept generation and evaluation. As mentioned

Contributed by the Design Theory and Methodology Committee of ASME for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received September 6, 2006; final manuscript received December 16, 2006. Review conducted by Clive L. Dym. Paper presented at the ASME 2005 Design Engineering Technical Conferences and Computers and Information in Engineering Conference (DETC2005), Long Beach, CA, September 24–28, 2005.

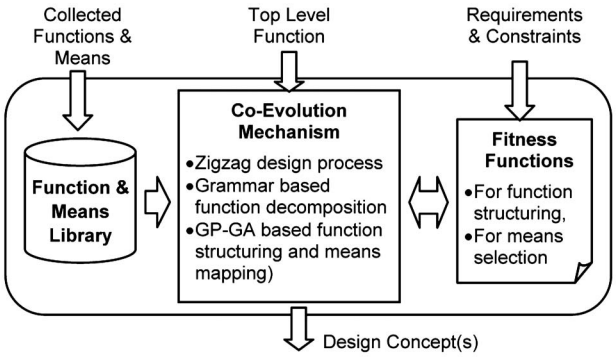


Fig. 1 Components of HiCED

above, a zigzag process is adopted as the overall coevolutionary design process. In this process, a grammar-based approach is applied to decompose higher-level functions. At a given level of the function decomposition hierarchy, a genetic programming (GP) and genetic algorithm (GA) based coevolutionary design algorithm is devised to evolve function structures and their corresponding solution means. If the resulting corresponding means contain unimplementable ones, further decomposition and coevolution are performed.

The fitness functions in HiCED can be derived from general design principles, design requirements and constraints, and designers' preferences. They serve as the evaluation criteria for evolving design concepts from one generation to another. There are two categories of fitness functions: those for evaluating function structures and those for selecting means combinations. To address the issue of the lack of quantitative information at the conceptual design stage, the fitness functions are framed to utilize maximum available information including the relations between different levels in the decomposition hierarchy, the function relations for means evaluation, and the means relations for evaluating function structures. This hierarchical and function-means coevolving mechanism for evaluation makes it possible for HiCED to apply various types of available information.

In the rest of the paper, we first review the related work in Sec. 2 and then present the details of the HiCED framework in Sec. 3. A case example is presented in Sec. 4 to demonstrate the effectiveness of the proposed approach, and finally the concluding remarks are discussed in Sec. 5.

2 Related Work

Our research on HiCED draws up on two major research fields, namely, engineering design and evolutionary computing. In the following, we review the related work in these two fields.

Design process study. The study on *design processes* has led to various practical process models and methods. *Axiomatic design* [2] is a general design approach that helps designers identify functional requirements (FRs) and corresponding design parameters (DPs) through a zigzag design process and two axioms, i.e., independent axiom and information axiom. Our HiCED adopts the zigzag design process and treats design as hierarchical coevolution of both function structures and their corresponding means. While in axiomatic design, designers have to generate FRs and DPs based on their experience; the grammar and coevolutionary mechanism of HiCED allows it to generate design concepts automatically based on designers' inputs. The *systematic design* [1] approach clearly identifies function structures and working principles for concept generation and applies morphology charts for design concept generation. Our HiCED uses the same definitions of functions, function flows, and function structures introduced in Ref. [1] and applies a hierarchical coevolutionary computing mechanism to generate and manipulate these concepts.

Grammar-based design. The *grammar-based approach* attracted the attention of design researchers for its ability of design exploration through a flexible expression of grammar rules [9,10]. From the early shape grammars [15] to the more recent graph grammars [10,11], synthesis grammar has been adopted as a formal definition of syntactic structure of mechanical systems. Some researchers have introduced the grammars of function and explored their application together with form grammars in specific engineering domains [16,17]. Our HiCED approach adopts the grammar-based approach for decomposing functions. However, unlike most existing grammar-based approaches, we use relatively "lean" grammar rules to decompose higher-level functions and generate the initial population of function structures. The further creation of function structures from this initial population is carried out through a GP and GA based coevolution process. This approach does not require creating strict grammar rules and allows generating more creative function structures. It, however, adds burdens to design evaluation.

Function modeling, The research on *functional representation* [18,19] provides a method to describe the contents of design and helps designers achieve the repeated and meaningful results from functional modeling. A standardized set function-related terminology, called a functional basis, has been developed for design knowledge capture and reuse. Based on the functional representation, Sridharan and Campbell [11] introduced an approach to function structuring by combining the functional representation and graph grammars. Our research also focuses on functions and treats function structuring as an important process in conceptual design. However, instead of being limited in functions, HiCED's coevolution approach also takes into consideration the means that perform the functions. This dual-focus approach allows effective evaluation of the function structures and the final design concepts.

Evolutionary and coevolutionary computing. Evolutionary computing is inspired by the natural evolution process [20,21]. The evolutionary computing approach solves a given problem by evolving the best solution(s) from a population through reproduction, mutation, recombination, natural selection, and survival of fitness. Recently, this approach has been taken to solve various engineering problems including design optimization, configuration design, and design automation [12,22–29]. While in evolutionary computing, individual solutions are selected based on a common fitness measure that does not depend on the state of the other individuals in the population, in coevolutionary computing individuals evolve within the context of other individuals either in the same population or in another population [30,31]. In *cooperative coevolution*, individuals of different species evolve separately and they together form a solution for the problem. For evaluation, each species is considered in turn in the context of other species. Examples of cooperative coevolution include evolving robotic soccer teams and evolving a group of predators. On the other hand, in *competitive coevolution*, individual fitness is evaluated relative to the others in the population, rather than through an absolute fitness measure. In this case, a fitness increase in one individual usually leads to a decreased fitness of another. In our research, we adopt a cooperative coevolution process. In HiCED, at each level of the abstraction hierarchy, two species, i.e., function structures and means combinations, coevolve in a shared design context until satisfied design concepts are found.

Coevolutionary design. The idea of design as coevolution between problems and solutions is not new [32]. Potter and De Jong [33] proposed a coevolutionary process to evolved design solutions by splitting a complex problem solving process into simpler subevolutionary search processes that cooperate to produce an overall solution. Information is implicitly communicated between the subevolutionary processes through the use of a shared fitness function. The problems that have been dealt with by this approach are "homogeneous" in the sense that each subproblem is simply a subset of a set of similar elements that have the same requirements of the composite structure. Maher and Poon [34] proposed a *coevolutionary design* approach and provided a computational

model to evolve designs by exploring both the problem space and the solution space. In this approach, the solutions are evaluated based on the problems and the problems are evaluated in the context of the evolved solutions. The coevolution in this approach is single level and can be applied only to the design problems that require incremental improvement. Campbell et al. [13,14] introduced an agent-based approach, called *A*-Design, to design synthesis that employs a coevolutionary search process. In *A*-Design, not only the designs but also the design creators, i.e., the agents, coevolve through an iterative search process. During each cycle of iteration, the agents modify the population of designs and are themselves modified based on the quality of designs they produce. The coevolution in *A*-Design is based on a goal-directed search process rather than through genetic operations such as mutation and crossover, meaning that rich knowledge is needed for the coevolution process but design moves are more purposeful. Rosenman and Saunders [35] proposed a hierarchical coevolution process in which the solutions of the components at a lower level of a decomposition hierarchy coevolve with the solutions of the assemblies at the immediately higher level. At a given level, the solutions of a component are evaluated by both the *intrinsic fitness*, which measures how well the solutions satisfy the criteria unique to this component, and the *extrinsic fitness*, which measures how the solutions satisfy the higher-level, i.e., the assembly level, requirements. Our HiCED approach is different from theirs in three ways. First, the abstraction hierarchy in HiCED is defined by functional decomposition rather than component decomposition where the lower-level solutions are the components of the higher-level assemblies. Furthermore, the functional decomposition in HiCED is controlled by a set of grammar rules not the evolution process. Second, while the search process in Ref. [35] is depth first and mostly bottom up, HiCED takes a breadth first and strict top-down process, mimicking the human zigzag design process. Lastly, the coevolution in HiCED is between function structures and their corresponding means combinations rather than between two adjacent levels in the hierarchy, although the information of the function structures and the means obtained at a higher level is used for fitness measures at the immediately lower level. This function-means coevolution mechanism increases the generality of HiCED.

3 Hierarchical Coevolutionary Design Process

An effective and efficient computational process for design concept generation must be able to deal with complex and nonroutine design problems, promote creativity, and adapt to the available design knowledge and design context information. Figure 2 illustrates the overall design process of HiCED. This process employs a hierarchical strategy to divide-and-conquer complex design problems and a general function-means mapping scheme to deal with nonroutine problems. By not requiring strict predefined design knowledge or rules and by applying context specific information in solution evaluation through a cooperative coevolution algorithm, our proposed design process encourages "creativity" in the sense that the latest available information of functions and means is utilized.

As shown in Fig. 2, there are three inputs to the HiCED design process: a function and means library, design requirements, and a top-level function. We assume that the library is continuously updated so that the most up to date knowledge is acquired and ready to be used by the HiCED. Both the design requirements and the top-level function are design problem specific. HiCED requires that a designer develops a top-level function that can be understood by the system. The design requirements influence the design process in two ways, i.e., they affect the preferences of means selections, and they are reflected in the fitness functions. After the top-level function and the requirements are given, HiCED starts its design exploration process. The following is a brief description of the major steps of this process.

Step 1: Grammar-based function decomposition. Based on the grammar rules, the overall function is decomposed into lower-level subfunctions that are more specific and form an *initial function structure set*. Unlike many other grammar-based design systems (e.g., Refs. [9–11]) that use grammar rules to generate a space of function structures to choose from, the function decomposition in HiCED creates an initial population of function structures at a given level. The creation of more, and possibly better, function structures is carried out through a function-means coevolution process. Because we do not expect the grammar rules to cover all possible search spaces, the rules can be relatively lean and, therefore, more general and easier to maintain. On the other hand, the lean rules leave much work for the "function structuring" step.

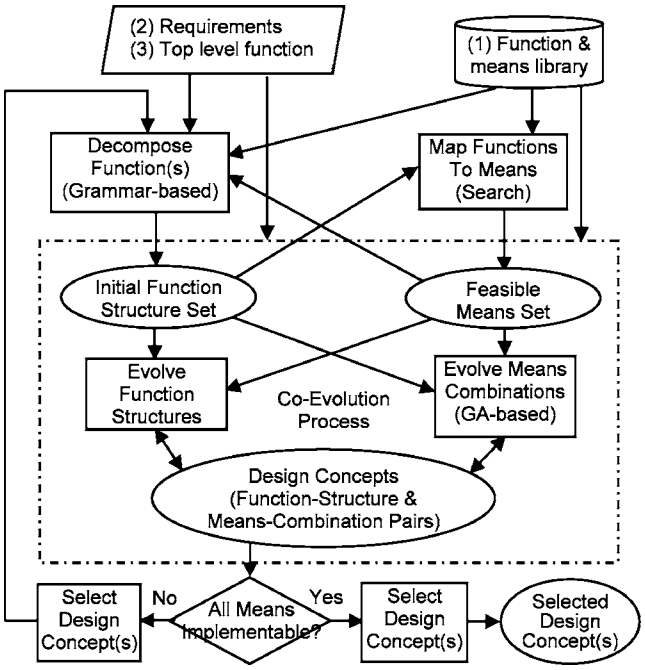


Fig. 2 The design process of HiCED

Step 2: Means selection. After the initial function structure set is generated, for each unique function in the set, a group of *feasible means* can be identified through a relatively simple search algorithm. Adding these feasible means groups together, we have a corresponding *feasible means set*. In HiCED, we assume that the knowledge of matching between a function and its feasible means is encoded in the function and means library. By organizing feasible means into groups and restricting the means space to the feasible means set, instead of the whole means library, we can increase the efficiency of coevolving function structures and means combinations in the next step.

Step 3: Function structuring and means combination selection. Once the initial function structure set and their corresponding feasible means set are generated, in this step, a cooperative coevolution algorithm is devised to evolve the best function structures and their corresponding means combinations. The following is the coevolution process.

- First, a GP based algorithm is employed to evolve *partially feasible function structures*. To ensure that the generated function structures are feasible, a number of constraints are associated with the genetic operations. To promote creativity in the evolution process, on the other hand, the constraints do not guarantee that all generated functional structures are feasible, that is why the function structures generated here are partially feasible. The final feasibility is maintained through the fitness functions, as described in Sec. 3.4. It is worth mentioning that in HiCED, any unique function can

be used in a function structure multiple times to form different function structures. As a result, the number of functions contained in a given function structure can be huge. This "dimension growth" is managed by imposing "dimension control" in the fitness function, as described in Sec. 3.4.
- After new function structures are generated, their information is used to evolve means combinations through a GA based algorithm. During this process, for each function structure, a GA based search is carried out to find best means combinations from the corresponding feasible means set. A bit string is used to model the space of feasible means. Since the size of the feasible means space is changing for different function structures, a dynamic modeling scheme is devised for the GA algorithm.
- In the light of the identified means combinations, the system goes back to further evolve better function structures by taking into consideration the information of the corresponding means combinations. The newly evolved function structures will then lead to generation of new means combinations and the coevolution of function structures and their corresponding means combinations continue until the satisfactory design concepts are found.

Step 4: Design concept selection. Once the coevolution process of the function structures and means combinations terminates with a set of function structure and means combination pairs, the system selects the best pairs whose fitness values are higher than an allowable threshold. If the selected pairs contain means that are not implementable, then go to step 1. Human designers can participate in this selection process and select the final design concept(s).

The HiCED computational design process described above was inspired by the human zigzag design process [2], also known as the requirement-solution tree method and the function-means tree method [8], and coevolutionary computing. In the zigzag design process, when designers attempt to decompose higher-level functions, they first try to identify solution means at the same level and then use the information of the means to guide the decomposition. It is conceivable that during this mapping (functions to means) and decomposing (higher-level functions into lower-level ones) process, a designer must mentally develop function structures and their corresponding means so that the mapping and decomposing are guided by these implicit function structures. Our process resembles this human design process. Our overall strategy is to use a relatively general and simple top-down grammar-based method to generate the initial population of function structures and then search for better ones through the function-means coevolution. At the core of this computational design process are the grammar rules, the genetic representations of function structures and means combinations, and the fitness functions.

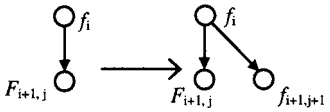
3.1 Grammar Rules. To decompose a higher-level function into lower-level ones, we employ a graph grammar-based approach. A number of grammar rules are introduced to generate subfunctions and their flow relations from a higher-level function. Grammar rules in a design synthesis system are the encoding of design knowledge. The advantage of using grammar rules is that partial design knowledge can be relatively easily represented as rules and the design moves are restricted to only the feasible ones. On the other hand, grammar rules are mostly design domain specific, and it is usually difficult to manage the rules when the new possibilities arise and old ones obsolete. In order to take advantage of the power of grammar rules without losing generality, in HiCED, we use grammar rules to generate the initial population of function structures and leave the further development of function structures to the coevolution process. HiCED has two sets of grammar rules: *general decomposition rules* that are generally applicable and similar to those in Refs. [10,17], and *action-based decomposition rules* that can be applied to more specific design situations and more in line with the rules found in expert systems [36] and those in Ref. [11].

3.1.1 General Function Decomposition Rule Set. The general function decomposition rule set is applied to function decomposition based on function flows. In HiCED, a function is represented as

$$f = \{\langle action\rangle\langle object\rangle, \langle input\_flows\rangle, \langle output\_flows\rangle\}$$

where $\langle action\rangle$ denotes the operation to be performed, $\langle object\rangle$ the object to be acted upon, and the attributes $\langle input\_flows\rangle$ and $\langle output\_flows\rangle$ are flows of energy, material, and signal. The following are the four rules in this rule set.

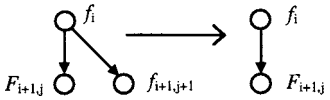
RULE 1. Function expansion rule.



$F_{i+1,j}$ is an unsatisfactory function decomposition set for function f_i at level $i+1$. j is the number of subfunctions in function set $F_{i+1,j}$.

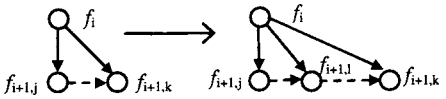
A major issue in applying this rule to function decomposition is how to choose the next function $f_{i+1,j+1}$ from the function library and make the new function decomposition set $F_{i+1,j+1}$ better satisfy the higher-level function f_i. In HiCED, a *greedy search algorithm* [37] is introduced to select the most *compatible* function $f_{i+1,j+1}$. First, we examine how many unmapped input flows (i.e., those that cannot find any feasible providing function) and unmapped output flows (i.e., those that cannot find any feasible receiving function) are there in the function set $F_{i+1,j}$, and then choose a function $f_{i+1,j+1}$ that can most effectively reduce the number of unmapped input and output flows. The role of the greedy search in HiCED is similar to the characteristic functions in Ref. 10. Focusing on function flows makes the rule more general.

RULE 2. Function reduction rule.



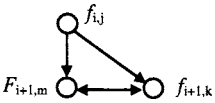
This rule removes function $f_{i+1,j+1}$ from the original decomposition set. One issue with the greedy search algorithm is that after a function $f_{i+1,j+1}$ is added into the decomposition set, the function may introduce new unmapped input and/or output flows that cannot be satisfied by any function in the function library. In this case, we need to remove the function from the function decomposition set and look for other functions for expansion.

RULE 3. Refinement-based expansion rule.



Following rule 1, only those functions that can reduce the number of unmapped input and/or output flows of the existing function structure have the chance to be selected. Any other functions, such as *channel* function [19] that has the same input and output flows, are excluded. Rule 3 is employed to enhance function decomposition by inserting a "refinement" function into the decomposition set generated by rule 1.

RULE 4. Decomposition termination rule.



For any function f_j at the ith level, if each input flow of a function $f_{i+1,k}$ in the function decomposition set $F_{i+1,m}$ comes

from either an output of another function in $F_{i+1,m}$ or f_j, and each output flow of $f_{i+1,k}$ goes to either an input of another function in $F_{i+1,m}$ or f_j, then the further decomposition for f_j terminates.

3.1.2 Action-Based Function Decomposition Rules. The action-based decomposition rules decompose a higher-level function into lower-level ones based on the ⟨*action*⟩ and ⟨*object*⟩ part of functions [36], instead of ⟨*input-flows*⟩ and ⟨*output-flows*⟩. The following three rules are defined in this rule set.

RULE 5. Action-based semantic expansion rule. In the engineering domain, it is often the case that a higher-level action act_h can be divided into a set of lower-level actions $\{\text{act}_{l1}, \text{act}_{l2}, \ldots, \text{act}_{ln}\}$ in the sense that the completion of the lower-level action set implies the completion of the higher-level action. We call this type of decomposition *semantic expansion*. For example, in the function ⟨transport⟩⟨object⟩, the action ⟨transport⟩ can be expanded as ⟨transport⟩→{⟨support⟩,⟨move⟩}. Therefore, the original function ⟨transport⟩⟨object⟩ can be expanded (i.e., decomposed) into two subfunctions, ⟨support⟩⟨object⟩ and ⟨move⟩⟨object⟩.

RULE 6. Action-based requiring rule. In some cases, the execution of a certain action requires the execution of some other actions. The original function can be decomposed by "including" new actions while keeping the original one. For example, the function ⟨test⟩⟨specimen⟩ requires ⟨load⟩ ⟨specimen⟩ and ⟨unload⟩⟨specimen⟩.

RULE 7. Action and object based function decomposition rule set. Sometimes, it is heuristically known that a certain function can be decomposed into a number of other functions. The result of this type of rule is that the original function is replaced by a new function set. In this case, not only the action but also the object in the function is expanded or replaced. For example, the function ⟨move⟩⟨object⟩ can be decomposed into ⟨generate⟩⟨ME⟩, ⟨guide⟩⟨ME⟩, and ⟨stop⟩⟨ME⟩.

Of these grammar rules, the general rules may lead to variety and novelty of the generated function structures by combining greedy search and refinement-based expansion, but it requires much computational resource. On the other hand, the action-based rules make the decomposition more efficient with the help of designers' experience. The drawback is that the search is done within a limited space and innovative opportunities may be lost.

3.2 Genetic Modeling of Function Structures. As shown in Fig. 2, after an initial function set is generated through applying grammar rules and their corresponding feasible means are found through a search algorithm, the next step is to coevolve function structures and means combinations. In HiCED, we adopt a GP framework for modeling function structures and a GA for means combinations.

There has been research that uses GP to automate the generation of configurations of electrical circuits [27] or microelectronic mechanical system [29]. These configurations are similar to our function structures. However, the circuit configuration problems are relatively simple in the sense that there are only a few building blocks, such as resistors, capacitors, and inductors, and one type of wire connections to deal with. Function structures of mechanical systems involve a lot more function blocks and they can be connected by different functional flows, namely, energy, material, and signal [1]. In addition, after an electrical circuit is generated by GP, all its parameters are determined; so designers can evaluate its performance. But for function structures in mechanical conceptual design, the available information is highly qualitative. Direct evaluation is almost impossible.

One of the main challenges in constructing a valid function structure by GP is how to create a transformation mechanism that can map a graphlike function structure into a GP tree. As mentioned above, function structures can be complex. A unique function can be used multiple times in a function structure, and each function can connect to other functions by any number of three types of flows. In order to convert a graphlike function structure to a treelike GP structure, we introduced a concept called *proxy node*

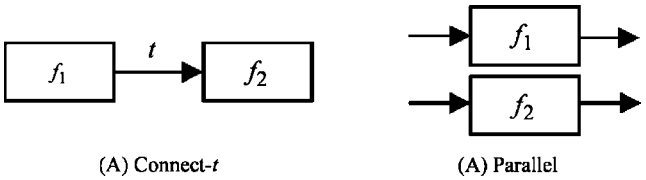


Fig. 3 Topological relationships in function

in our GP model. For example, function f_A can have one or more proxies, denoted as f_A', in a GP tree. The proxy nodes are considered as the same of their original nodes when they are converted back to functional structures from a GP tree.

To model the topology of function structures, we introduce two relationships, namely, *Connect* and *Parallel*. *Connect* indicates that a function is linked to another by a function flow t, while *Parallel* stands for a situation that two functions have no direct connections. Figure 3 illustrates the two relationships.

In the HiCED GP algorithm, four GP functions are defined and they are three *Connect* functions and one *Parallel* function.

Connect-E(f_A,f_B,e): connect f_A to f_B by energy flows e.

Connect-M(f_A,f_B,m): connect f_A to f_B by material flows m.

Connect-I(f_A,f_B,s): connect f_A to f_B by information flows s.

Parallel (f_A,f_B): f_A and f_B are not connected with each other.

In a GP tree, internal nodes are GP functions, and terminals are the functions of a given function decomposition set or their proxies.

Figure 4 illustrates an example of a function structure (Fig. 4(*a*)) and its representations in terms of a GP tree (Fig. 4(*b*)) and a GP function (Fig. 4(*c*)).

3.3 Genetic Modeling of Means. While the evolution of function structures is GP based, a GA approach is taken for evolving means combinations since the topological information is already captured by the function structures. In this algorithm, the means are encoded into strings of binary bits. To ensure that the genetic operations yield feasible means for each function, the length of bits for each means of a given function's feasible means set is dynamically determined by the total number of the feasible means in that set. For a given function decomposition set $\{f_1,\ldots,f_n\}$ with corresponding feasible means sets $\{M_1,\ldots,M_n\}$, the means combinations can be modeled as the chromosome, as shown in Fig. 5.

The evolution of means in GA is determined by three genetic operations, namely, reproduction, crossover, and mutation [20]. *Reproduction* simply copies a parent into a child. For *crossover*, we use a single-point crossover operator to recombine the selected individuals of current population. *Mutation* is single bit based.

3.4 Fitness Functions for Function Structuring. How can we evaluate the fitness of a given function structure is a major

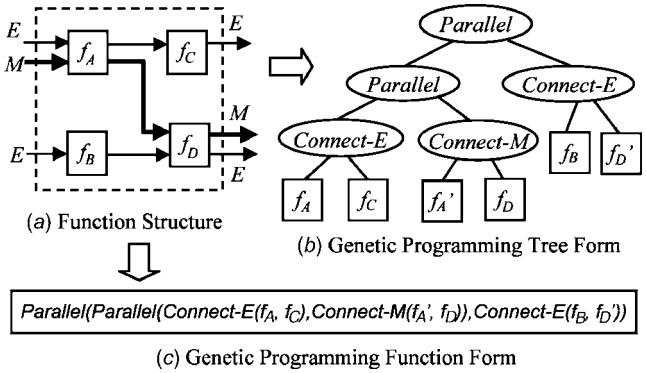


Fig. 4 A chromosome model of function structure

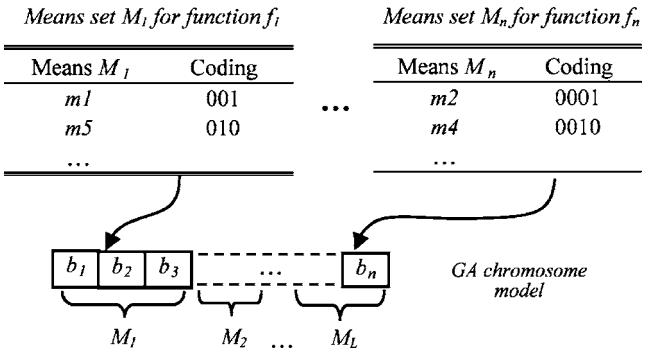


Fig. 5 A chromosome model of means for means combination selection

challenge in our research because there is little quantitative information available. In this research, we introduce two groups of fitness measures for evaluating function structures: one evaluates the *feasibility* and other the *desirability*.

As mentioned above, while we impose constraints on the genetic operations so that most flow connections of the child function structures generated are feasible ones, the constraints do not guarantee the feasibility of all generated function structures. Untested function structures may be generated whose feasibility is to be evaluated. This shift of feasibility treatment from "up front" (i.e., generate only the feasible ones) to "back end" (i.e., select the feasible ones) has important implications. First, it resembles human trial-and-error design process. Human designers, when developing designs for original design problems, often try different new functions and their connections, lay them out, and then evaluate their feasibilities. At the conceptual design stage, finding *feasible* designs through trial-and-error is a major part of the design process. Second, by not being overly restricted in the generation process, the system can explore a wider space and has the potential of generating "creative" solutions. However, the process may be less efficient since certain amount of computation is devoted to infeasible solutions.

Following Ref. [1], a feasible function structure must maintain three basic properties: (1) each input (or output) flow of a function in a function structure should come from (or goes to) one or more other functions, or outside of the system; (2) a lower-level function structure should be consistent with its higher-level ones; and (3) the function structure should have the same input and output flows as its top function. In HiCED, the following three consistency fitness functions are introduced.

Function flow consistency (FC). For a function f_i in a function structure F_s, all its input flows should come either from other functions or from the outside of the system. If a flow does not meet this condition, it is called an unsatisfied flow. We have

$$ff_{\mathrm{FC}} = -\sum_{i}^{\mathrm{Fs}} unsatisfied(f_i);\ unsatisfied(f_i)$$

returns the number of unsatisfied flows of f_i.

Hierarchical flow consistency (HC). When two functions f_i and f_j are decomposed into lower-level functions F_i and F_j, respectively, it is expected that the flow relations between f_i and f_j be maintained. Therefore, *relations* (F_i,F_j) should match *relations* (f_i,f_j). We have

$$ff_{\mathrm{HC}} = -\sum_{i}^{\mathrm{Fs}} unmatch\begin{pmatrix} relations\ (f_i,f_j), \\ relations\ (F_i,F_j) \end{pmatrix};\ unmatch(R_i,R_j)$$

returns the number of unmatched flows between R_i,R_j.

Global flow consistency (GC). When a given function structure Fs is viewed as a single blackbox, then its input and output flows must be consistent with those of the top-level blackbox function f_{top}. The number of unmatched flows is used as global flow feasibility fitness measure.

$ff_{\mathrm{GC}} = -unmatched(\mathrm{Fs},f_{\mathrm{top}})$; *unmatched*(Fs,$f$) returns the number of unmatched flows of Fs compared with f.

Evaluating the desirability of a function structure is not easy because there is not much quantitative information available. In HiCED, we focus on dimension, variety, and desirable connections, and introduced *function structure dimension* (SD), *structure function variety* (SV), and *grammar rule usage* (RU) as desirability fitness functions. The SD fitness measure is used to keep the dimension (i.e., total number of functions) in a desirable range. For a given function structure, a higher level of variety may lead to better functional performance, but a lower level of variety may increase the robustness of the system. The fitness *function structure function variety (FV)* can be used to reflect this desirability. In our current case study, higher function variety is considered more desirable. When the initial population of function structures is generated from the grammar-based decomposition, certain connections are imposed between certain functions. It is desirable that these functions are maintained through the evolution process. The RU fitness function is used to control this desirability.

In addition to the structural properties described above, the desirability of a function structure can be evaluated by whether the desirable means combinations can be found to realize the function structure. The *function-means mapping (FM)* fitness function is introduced to assess this desirability. The following are the four desirability fitness functions.

Function structure dimension (SD). For a given function structure Fs, let N_T be the number of functions in Fs, N_U the number of unique functions in the current population, and k_{sd} is a coefficient, indicating some allowable level of duplication. The exponential ratio is used to measure dimension desirability.

$$ff_{\mathrm{SD}} = \begin{cases} -1 \times e^{N_T/N_U} & \text{if } N_T > k_{sd} \times N_U \\ 0 & \text{otherwise} \end{cases}$$

Structure function variety (FV). For a given function structure Fs, let $N_{U\text{-}Fs}$ be the number of unique functions in the function structure Fs and N_U the number of unique functions in the current population. The different is used to measure function variety:

$$ff_{\mathrm{FV}} = -(N_U - N_{U\text{-}\mathrm{Fs}})$$

Grammar rule usage (RU). For two functions, f_i and f_j, in a given function structure Fs, if flows between f_i and f_j,*flows*(f_i,f_j), are derived from certain grammar rules, *rule_flows*, it is more desirable that these flow relations are kept in child function structures

$$ff_{\mathrm{RU}} = -\sum_{i,j}^{\mathrm{Fs}} unmatch\begin{pmatrix} flows(f_i,f_j), \\ rule\_flows \end{pmatrix};\ unmatch(r_1,r_2)$$

returns 1 if r_1,r_2 are unmatch, 0 otherwise.

Function-means compatibility (FM). For two connected functions f_i and f_j in a function structure Fs and their corresponding means m_i and m_j, it is more desirable if connections between m_i and m_j are compatible with those between f_i and f_j is true for more (i,j) pairs

$$ff_{\mathrm{FM}} = -\sum_{i,j}^{\mathrm{Fs}} incompatible\begin{pmatrix} flows(f_i,f_j), \\ flows(m_i,m_j) \end{pmatrix};\ incompatible\ (r_1,r_2)$$

returns 1 if r_1,r_2 are incompatible, 0 otherwise.

Summarizing the above, we introduce the following overall fitness function for function structure evaluation as the weighted sum of the above fitness function components:

$$ff_{FS} = w_{FC}ff_{FC} + w_{HC}ff_{HC} + w_{GC}ff_{GC} + w_{SD}ff_{SD} + w_{FV}ff_{FV} + w_{RU}ff_{RU} + w_{FM}ff_{FM}$$

We will discuss at length how these fitness functions may impact design concept generation, and how these weighting values should be distributed in a case study later.

3.5 Fitness Functions for Means Selection. HiCED coevolves function structures and their corresponding means combinations. To evaluate the fitness of a given means combination, we need to know whether the means in the combination are compatible with each other and, given that they are compatible, how desirable the means combination is. A thorough evaluation of means compatibility requires much information of the details of each means. In our current research, we evaluate the compatibility by using the function structure information and assume that the compatibility is higher if the flow relations defined by the function structure are compatible with the flow relations that can be realized by the corresponding means. Our coevolution approach provides a platform for using such function relations to evaluate means combination compatibilities. We thus introduce the following measure to evaluate the means compatibility of a combination.

Means connection compatibility (MC). For any two means m_i and m_j in a given means combination Mc and their corresponding functions f_i and f_j in the corresponding function structure Fs, the means in Mc are more compatible if means connections between m_i and m_j compatible with those defined between f_i and f_j are true for more (i,j) pairs

$$ff_{MC} = -\sum_{i,j}^{Fs,Mc} incompatible\begin{pmatrix} flows(f_i,f_j) \\ flows(m_i,m_j) \end{pmatrix};\ incompatible(r_1,r_2)$$

returns 1 if r_1, r_2 are not compatible, 0 otherwise,

Besides the compatibility of means in a means combination, we can also evaluate the desirability by checking if the means can provide better performance and are more preferable. We therefore introduce the following two fitness measures.

Means performance indicator (MP). For a given means combination Mc, let the selected design requirements be $\{q_1, q_2, \ldots\}$, and simple performance measures $\{p_1, p_2, \ldots\}$. The smaller the difference, the more desirable the means combination is

$$ff_{MP} = -\sqrt{\sum_i (q_i - p_i)^2}$$

Means preference indicator (MR). For a given means combination Mc, for each means m_j in Mc, the assigned preference is pr_i. The means combination is more desirable if the total preference is higher

$$ff_{MR} = -\sum_i^{Mc} (1 - pr_i) \quad 0 \leqslant pr \leqslant 1$$

Although much parametric or geometric information may not be obtainable at the conceptual design stage, certain simple performance measures, such as weight, cost, and grade level, can be associated with various means. The MP uses these attributes and measures the difference between these attributes and the stated performance requirements as performance fitness. In addition, designers can assign preferences to individual means based on their confidence [38] on the means or on the frequency of use of the means in the similar design contexts. When preference information is available, the MR function can be used as a desirability fitness measure.

The overall fitness function for evaluating a means combination is a weighted sum of individual fitness functions expressed as

$$ff_{means} = w_{MC}ff_{MC} + w_{MP}ff_{MP} + w_{MR}ff_{MR}$$

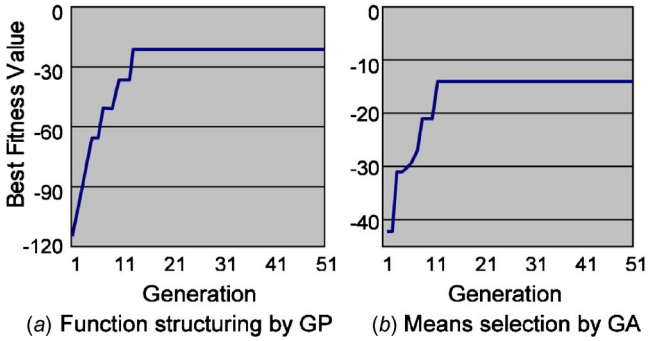


Fig. 6 Examples of convergence curves for coevolutionary design

4 Case Study

The goal of our case study is to investigate (1) how the design results correspond to different design scenarios or requirements, (2) how design solutions are sensitive to the fitness functions, and (3) how design solutions respond to genetic parameter settings. For our case study, we chose to expand Ullman's bicycle design problem [38] to a problem of *designing a self-powered personal transporter*. This expansion creates a bigger design space for HiCED to generate different kinds of personal transporters.

HiCED has been implemented using Java language in a Windows XP environment. The function library was created based on the function structures developed by the mechanical engineering students in their design projects for a senior level design methodology course. The means in the means library were collected from commonly used mechanical vehicles, such as bicycle, scooter, and skating board. As an input to HiCED, we set the top-level function to be ⟨transport⟩⟨X⟩.

Our system runs on a Pentium-4 2.2 GHz PC with 512M memory. Initially there were 14 functions in function library and 41 means in means library. It takes about an hour for HiCED to generate best solutions. Figure 6 shows an example of convergence curves for function structuring and means selection.

4.1 Experiment Design. As shown in Fig. 7, our experiment design has two independent variables, two dependent variables, and two control variables. In this study, the independent variable *design problem* has only one value, i.e., the top-level function ⟨transport⟩⟨X⟩. For *user requirements*, we have three design cases for testing how HiCED can generate different designs in response to different user requirements: (1) design for low cost, (2) design for lightweight, and (3) design for long travel range.

Two parameters were chosen as dependent variables, i.e., the *fitness of design* and the program *running time*. In addition, we also assess the design results manually by examining the design concepts.

The first control variable in Fig. 7 is *genetic parameter setting*. In our experimental study, we are interested in understanding how the mutation rate can help or hinder the results. As will be discussed below, we changed the rate of mutation from 0% to 15%

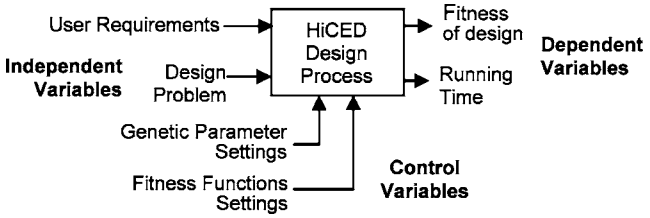


Fig. 7 Experiment design

Table 1 Functions used in case study

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

and gained interesting insights.

The main focus of our case study was to investigate how different distributions of the weights of the fitness functions impact the design results. Through this investigation, we hope to gain insights on what are important for design concept generation. We pay specific attention to the function structure formation and evaluation.

4.2 Function and Means Library. The function definitions in the HiCED function and means library are shown in Table 1 and some means examples are illustrated in Table 2.

In Table 2, *applicable functions* are a set of functions for which the means can be a carrier; *high-level-consistent-means* set contains those means of which this means can be a subtype. The applicable-functions set can help map functions to desired means, and the high-level-consistent-means set can be used to determine whether the corresponding function that this means is supposed to satisfy is still decomposable.

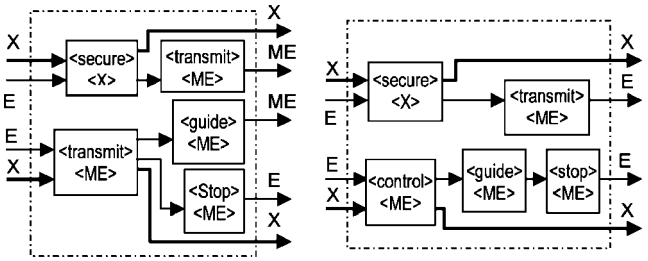
In addition to the means in the library, the constraints that indicate certain relationships between a specific means and other objects, such as another means, are also defined. The following is an example of such constraints:

$$c1 = \begin{Bmatrix} \text{“steering wheel”}, \langle guide \rangle \langle ME \rangle; \text{“pedal brake”}, \\ \langle stop \rangle \langle ME \rangle; \text{“}connect\text{”}, 0 \end{Bmatrix}$$

The expression indicates a relationship that means *steering wheel*, which satisfies function ⟨*guide*⟩⟨*ME*⟩, cannot directly connect to means *pedal brake*, which fulfills function ⟨*stop*⟩⟨*ME*⟩. Usually the constraints are the knowledge derived from design principles, expertise, and experience. Currently, we assume that the knowledge is readily available. The next step of our research will investigate how to acquire the knowledge for design concept generation.

4.3 Analysis of Results. In this subsection, we first show, through the personal transporter example, how HiCED performs coevolution. After that, we discuss how the system responds to the differing user requirements, the changes of fitness function weights, and the settings of genetic parameters.

4.3.1 Coevolution of Functions and Means. Coevolution of function structures and their means combinations is the key process of HiCED. Figures 8(*a*) and 8(*b*) are two alternative function structures generated by the GP algorithm at level 3 of the decomposition hierarchy. Both of them are “equally good” if only functional flow compatibility is considered for evaluation. When the candidate means that implement the functions ⟨stop⟩⟨ME⟩ and ⟨guide⟩⟨ME⟩, shown in Fig. 8(*c*), are taken into consideration, however, the difference becomes obvious. Based on design experience, we know that it is not feasible to connect the means for ⟨guide⟩⟨ME⟩ to the means for ⟨stop⟩⟨ME⟩ directly, as expressed in the example constraint in Sec. 4.2. Once means are considered, function structures are reevaluated based on the new means information. The final result shows that alternative 1 is better than alternative 2 because it does not require direct connection of means between the two functions.



(*a*) Alternative 1 (*b*) Alternative 2

Function	*Means*
<guide><ME>	Steering wheel Guide wheel
<stop><ME>	Cramp brake Pedal brake Friction with ground

(*c*) Available Means for <guide><ME> and <stop><ME>

Fig. 8 Two alternative function structures at level 3

4.3.2 User's Requirements. Three scenarios are used to test how HiCED responds to different user requirements, i.e., “low cost requirement,” “lightweight requirement,” and “long travel range requirement.” “Long travel range” was translated to “choosing the designs that require less power from the rider.” With respect to the requirements of lightweight and low cost, the system came up with the same function structure as shown in Fig. 9 but different means combinations (Fig. 10). As shown in Fig. 10, one of the solutions was close to a skateboard and the other resembles roller skate shoes. For the requirement of long travel range, the system generated a different function structure (Fig. 11) and a different means combination (Fig. 12), similar to a bicycle design.

From Figs. 8–12, it can be seen that the design concepts generated by HiCED are highly conceptual in the sense that there is little physical or embodiment information attached to the concepts. The sketches shown in the figures are only one way to

Table 2 Sample means used in the case study

ID	Means	Applicable functions	High-level consistent means	Weight value	Cost value
…	…	…	…	…	…
m_2	pedal drive	f_2	Φ	n/a	n/a
m_{14}	pedal gear	f_{10}	m_2	3	3
m_{20}	pedal	f_{13}	m_2	1	1
m_{26}	chain	f_{12}	m_2	2	2
…	…	…	…	…	…

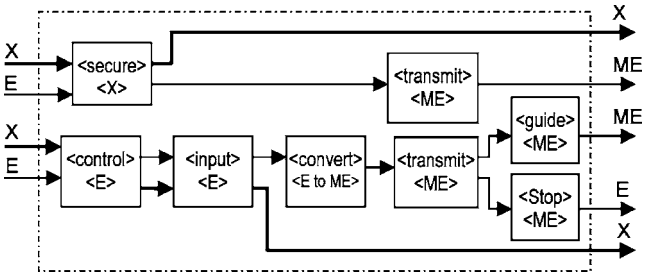


Fig. 9 Function structure regarding weight and cost requirements

interpret the three design concepts. In our research on HiCED, we have strived to avoid requiring embodiment information and to use general design knowledge as much as possible. The advantage of this approach is that the system can generate design concepts in the absence of specific embodiment information so that the designers can quickly examine and evaluate the design concepts generated without having to provide detailed and specific embodiment information; oftentimes, a designer may not be able to provide such information until a design concept is given. The drawback of this approach is that the system generates more feasible design concepts than desirable ones, since there is no sufficient

Function	Minimum Cost		Minimum Weight Solution
	Solution 1	Solution 2	
<secure><X>	board	shoes	shoes
<transmit><ME>	wheels	wheels	wheels
<control><E>	human	human	human
<input><E>	feet	feet	feet
<convert><E to ME>	shaft	shaft	shaft
<transmit><ME>	wheels	wheels	wheels
<guide><ME>	guide wheels	guide wheels	guide wheels
<stop><ME>	friction	friction	friction

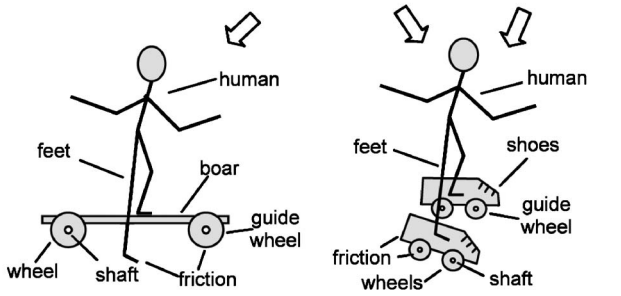


Fig. 10 Solutions with requirements of low cost and lightweight

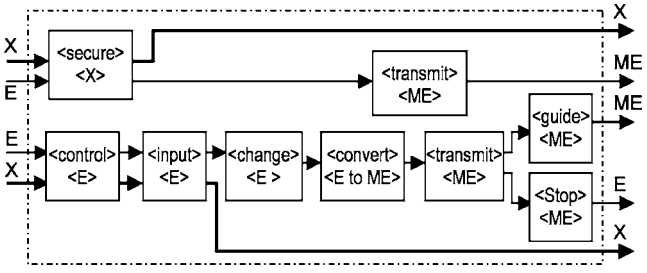


Fig. 11 Function structure for long travel range requirement

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

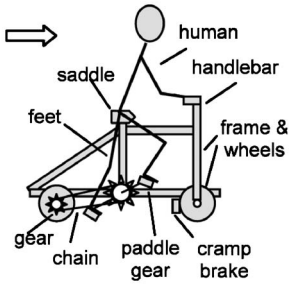


Fig. 12 Means selection with long travel range requirement

information available for the system to "improve" a feasible or a good design concept to make it better or more desirable.

4.3.3 Impact of Fitness Functions. Function structure generation and evaluation has been a focus of our research on HiCED because a function structure is the basis of a design concept. As described in Sec. 3.4, currently seven fitness function components are identified for function structure evaluation. In this subsection, we discuss how these different fitness function components determine the formation of function structures and influence the generation of final design concepts.

Our experiments were conducted in the following procedure. First, we assign each fitness function component an equal medium weight. Then, we test each fitness function component in two situations while keeping all other component weights the same: one with a smaller weight and the other with a bigger weight. By comparing the solutions in different weighting cases, we can investigate how much the function structuring is sensitive to each of the fitness components and develop a best weight distribution for the fitness function. Figure 13 illustrates one such best distribution. The impact and implications of each fitness component can be summarized as follows.

FC is a basic criterion for a valid function structure. Any input/output flow of a function must be an output/input flow of another function, or it must be inherited from the top-level function. Consistent with our predictions, a big enough weight of FC is needed for the system to maintain the feasibility of function structures.

Both HC and RU are related to applying the information of one higher level to the evaluation of the current level function structures. A bigger HC weight contributes to the generation of feasible function structures, and a bigger RU weight allows designers to enforce certain desired functional relations. The hierarchy provides needed contextual information for the process of coevolution. When the weight of HC becomes much bigger than the others, the system tends to exclude certain functions. We suspect that the relative value difference of HC being bigger than FV, de-

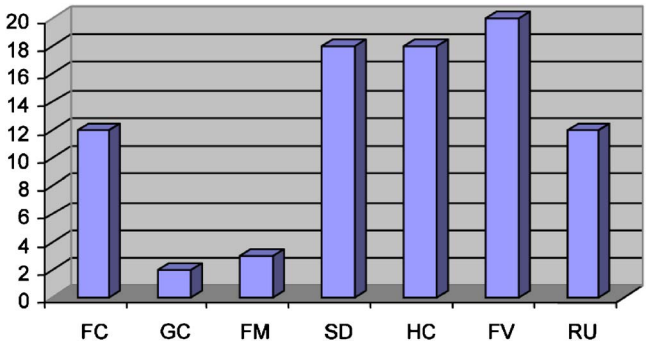


Fig. 13 An effective distribution of fitness function weight

scribed below, had an effect on this behavior.

GC is a criterion to determine whether function structure has the same input and output flows with the top-level function. The results shown in Fig. 13 contradicted our hypothesis that GC is an important factor. Changing the weight of GC did not have significant impact on the results. This behavior can be interpreted as that the FC and HC have already imposed strong feasibility requirements so that GC can usually be maintained. Furthermore, a bigger weight of structure FV, discussed below, also made it easier to satisfy global flow consistency.

SD limitation prevents function structures from being expanded randomly. The experimental results are consistent with our hypotheses that feasible solutions can be generated only when an effective dimension control mechanism is applied. As indicated in Sec. 3.4, the SD fitness function follows an exponential expression, which is the result of our iterative sensitivity study. The resulting size of the function structures was controllable by adjusting the allowable ratio k_{sd}.

The FV fitness function controls the heterogeneity of functions in function structures. In our experiments, we consider function structures with higher variety more desirable. We did not expect to see that after increasing the weight of FV, not only more unique functions were included but also the system became less sensitive to other feasibility related fitness functions, meaning that it became "easier" to generate feasible function structures. The side effect of bigger FV weight is that the computing time increases significantly.

We expected the FM fitness function to play an important role in generating good function structures. Changing the weight of FM, however, had only limited effect on the results. We speculate that this behavior is due to the limited number of functions in the function and means library. We had many more means than functions in the library, so that it was not difficult to satisfy the FM requirement. In comparison, generating feasible function structures with adequate structural dimension and function variety was a major "fight" of the HiCED design process. Our future research will test larger libraries with different function and means ratios.

4.3.4 Impact of Mutation. After setting all the fitness component weights to their best values, as indicated in Fig. 13, we checked how the mutation fraction affects the generation of function structures. In our test, the mutation fraction was changed from 0% to 15%. We observed that few feasible solutions were generated when mutation fraction was less than 1%. From 1% to 10%, with the increase of mutation fraction, more feasible solutions are produced. But, solutions became worse (compared with the "best one" generated) when mutation percentage is greater than 7%.

We expected that mutation could play a crucial role in generating new design concepts. But, the effective range of mutation fraction was higher than what we considered. We speculate the following. The highly conceptual search space of function structures is relatively "sparse." In addition, the lack of evaluation information makes the space "flat" (i.e., less ups and downs) as well. Without sufficient mutation, it is easy for the search to stay within a limited number of potential solutions. Higher mutation percentage encourages "jumping" around to find more unexplored solutions.

5 Concluding Remarks

As design problems become more complex and design lead time more pressing, designers need supporting tools to expand their exploration of the design space and to increase the number and quality of their design concepts. Our research takes a hierarchical coevolution approach to help designers explore design space and develop design concepts by automatically generating design concepts based on the inputs provided by the designers. The approach adopts a zigzag design process. At a given level of abstraction hierarchy, a set of grammar rules is applied to decompose the higher-level functions and create an initial population of function structures. A GA and GP based algorithm is devised to let function structures and their corresponding means combinations coevolve into design concepts. A HiCED prototype system was developed and the case study results demonstrated the effectiveness of the proposed approach and revealed how various fitness function components and genetic parameters might impact the design concept generation process and results.

In many creative conceptual design situations, designers may not have quantitative embodiment information to start with. Still, they need to explore various possible design concepts with their highly conceptual or qualitative knowledge about functions and means. There two reasons for this. One is that the information is not available and the other that applying the detailed embodiment information is not desirable because the early commitment to use such information may misguide the concept generation process. In our research on HiCED, we do not assume the availability, or the application, of the quantitative embodiment information. Definitions of the functions and means in the library are highly conceptual and can be acquired relatively easily. The common strategy in developing a system in this situation is to follow a top-down and rule-based approach, since the semantics can be handled by the rules. The drawback of this approach, however, is that the maintenance of the rules can become very difficult. To avoid this problem, we took a "lean" knowledge strategy by first using a few grammar rules to decompose the functions and generate the initial population of the function structures and then applying a genetic coevolution process to evolve more function structures and their means combinations. Identifying and balancing the fitness function components was a challenge in this research partly because of the spars and flat features of conceptual search space. The case study demonstrated the effectiveness of our strategy and the HiCED system provides a test bed to further our research along this line.

Our current HiCED system is limited by the small size of the function and means library and the limited number of genetic functions. It is conceivable that as the size of the libraries increases and more genetic functions are introduced, the computation time will increase dramatically. Our current research aims to improve the coevolutionary algorithm to deal with the scale-up issue. Furthermore, the HiCED model of design concept generation provides a flexible computational environment for us to investigate how different fitness functions may impact the *novelty, variety, quantity*, and *quality* [39,40] of design concept generation. We are designing new sets of experiments to identify correlations between the fitness functions, contents of the function and means library, and the above mentioned design metrics. Lastly, developing efficient methods to create and enrich the function and means library remains to be a research issue. We are currently exploring an intelligent agent-based approach to address this issue in which agents try to pick up useful functions and means from design logs, databases, and various documents through data-mining and text-mining techniques.

References

[1] Pahl, G., and Beitz, W., 1996, *Engineering Design: A Systematic Approach*, Springer, New York.
[2] Suh, N. P., 1990, *The Principles of Design*, Oxford, New York.
[3] Akao, Y., 1995, *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Productivity, Portland, OR.
[4] Tomiyama, T., 1995, "A Design Process Model That Unifies General Design Theory and Empirical Findings," ASME *Proceedings of 1995 Design Engineering Technical Conference*, pp. 329–339.
[5] Cross, N., Christianns, H., and Dorst, K., 1997, *Analysing Design Activity*, Wiley, New York.
[6] Benami, O., and Jin, Y., 2002, "Cognitive Stimulation in Creative Conceptual Design," ASME *Proceedings of 14th International Conference on Design Theory and Methodology*, Sept. 29–Oct. 2, Montreal, Canada, Paper No. DETC2002/DTM-34023, pp. 1–13.
[7] Jin, Y., and Chusilp, P., 2006, "Stuty of Mental Iteration in Different Design Situations," Des. Stud., **27**, pp. 25–55.
[8] Bracewell, R. H., 2002, *Synthesis Based on Function—Means Trees: Scheme-*

builder, in Engineering Computational Design: Understanding, Approaches and Tools, Springer, New York, pp. 199–212.
[9] Schmidt, L., and Cagan, J., 1995, "Recursive Annealing: A Computational Model for Machine Design," Res. Eng. Des., **7**, pp. 102–125.
[10] Schmidt, L., and Cagan, J., 1997, "GGREADA: A Graph Grammar-Based Machine Design Algorithm," Res. Eng. Des., **9**, pp. 195–213.
[11] Sridharan, P., and Campbell, M. I., 2004, "A Grammar for Function Structure," *Proceedings of ASME 2004 Design Engineering Technical Conferences*, Salt Lake City, UT, Paper No. DETC2004-57130, pp. 1–15.
[12] Maher, M. L., 2001, "A Model of Co-Evolutionary Design," Eng. Comput., **16**, pp. 195–208.
[13] Campbell, M. I., Cagan, J., and Kotovsky, K., 1999, "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment," Res. Eng. Des., **11**, pp. 172–192.
[14] Campbell, M. I., Cagan, J., and Kotovsky, K., 2000, "Agent-Based Synthesis of Electromechanical Design Configurations" ASME J. Mech. Des., **122**, pp. 61–69.
[15] Stiny, G., 1980, "Introduction to Shape and Shape Grammars," Environ. Plann. B, **7**, pp. 343–351.
[16] Li, X., and Schmidt, L., 2000, "Grammar-Based Designer Assistance Tool for Epicyclic Gear Trains," ASME J. Mech. Des., **126**, pp. 895–902.
[17] Starling, A. C., and Shea, K., 2002, "A Clock Grammar: The Use of a Parallel Grammar in Performance-Based Mechanical Synthesis," ASME Proceedings, Paper No. DETC'02/DTM-34026, pp. 1–10.
[18] Stone, R., and Wood, K., 2000, "Development of A Functional Basis for Design," J. Mech. Des., **122**(4), pp. 359–370.
[19] Hirtz, J. M., Stone, R. B., McAdams, D. A., Szykman, S., and Wood, K. L., 2002, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," Res. Eng. Des., **13**, pp. 65–82.
[20] Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Longman.
[21] Koza, J. R., 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge.
[22] Fogel, L. J., Owens, A. J., and Walsh, M. J., 1996, *Artificial Intelligence Through Simulated Evolution*, Wiley, New York.
[23] Parmee, I. C., 1997, "Evolutionary Computing for Conceptual and Detailed Design," *Genetic Algorithms in Engineering and Computer Science*, Wiley, New York.
[24] Bentley, P. J., 1999, *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco.
[25] Bonnie, R. M., and Malaga, R., 2000, "A Co-Evolutionary Approach to Strategy Design for Decision Makers in Complex Negotiation Situation," *IEEE Proceedings of the 33rd Hawaii International Conference on System Sciences*, pp. 1–9.
[26] Lee, C.-Y., Ma, L., and Antonsson, E. K., 2001, "Evolutionary and Adaptive Synthesis Methods," *Formal Engineering Design Synthesis*, Cambridge University Press, Cambridge, pp. 270–320.
[27] Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A., 1999, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," IEEE Trans. Evol. Comput., **1**(2), pp. 109–128.
[28] Vajna, S., and Clement, S., 2002, "Autogenetic Design Theory: An Approach to Optimize Both the Design Process and the Product," ASME *Proceedings of DETC02, ASME 2002 Design Engineering Technical Conferences*, Montreal, Canada, Paper No. DTEC2002/DAC-34038, pp. 1–7.
[29] Fan, Z., Seo, K., Hu, J., Rosenberg, R., and Goodman, E. D., 2003, "System-Level Synthesis of MEMS via Genetic Programming and Bond Graphs," Lect. Notes Comput. Sci., **2724**, pp. 205–206.
[30] Pollack, J., Blair, A., and Land, M., 1996, "Coevolution of a Backgammon Player," *Artificial Life V: Proceedings of the Fifth Artificial Life Conference*, C. Langton and T. Shimohara, eds., Nara, Japan, MIT Press, Cambridge, pp. 92–100.
[31] Ahluwalia, M., Bull, L., and Fogarty, T. C., 1997, "Coevolving Functions in Genetic Programming: A Comparison in ADF Selection Strategies," *Proceedings of the Second Annual Conference on Genetic Programming*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, R. Riolo, eds., Morgan Kaufmann, San Francisco, pp. 3–8.
[32] Kicinger, R., Arciszewski, T., and De Jong, K. A., 2005, "Evolutionary Computation and Structural Design: A Survey of the State of the Art," Comput. Struct., **83**, pp. 1943–1978.
[33] Potter, M. A., and De Jong, K. A., 2000, "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," Evol. Comput., **8**(1), pp. 1–29.
[34] Maher, M. L., and Poon, J., 1996, "Modeling Design Exploration as Co-Evolution," Microcomput. Civ. Eng., **11**(3), pp. 195–210.
[35] Rosenman, M., and Saunders, R., 2003, "Self-Regulatory Hierarchical Coevolution," Artif. Intell. Eng. Des. Anal. Manuf., **17**, pp. 273–285.
[36] Jin, Y., Kunz, J. C., Levitt, R. E., and Winstanly, G., 1992, "Design of Project Plans From Fundamental Knowledge of Engineered Systems," *Working Notes: AAAI Fall Symposium Series*, pp. 149–154, AAAI Press.
[37] Li, W., 2006, "A Hierarchical Co-Evolutionary Approach to Conceptual Design," Ph.D. thesis, University of Southern California, Los Angles, CA.
[38] Ullman, D. G., 2003, *The Mechanical Design Process*, 3rd ed., McGraw-Hill, New York.
[39] Shah, J. J., Vargas-Hernandez, N., and Smith, S. M., 2003, "Metrics for Measuring Ideation Effectiveness," Des. Stud., **24**, pp. 111–134.
[40] Chusilp, P., and Jin, Y., 2006, "Impact of Mental Iteration on Conceptual Design Performance," ASME J. Mech. Des., **128**, pp. 14–25.