INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

EXPLORING THE WORTH OF AUTOMATICALLY GENERATED DESIGN ALTERNATIVES BASED ON DESIGNER PREFERENCES

Tolga Kurtoglu[1], Matthew I. Campbell[2]

[1]Graduate Research Assistant, The University of Texas at Austin, Austin, TX, USA, 78712
[2]Associate Professor, The University of Texas at Austin, Austin, TX, USA, 78712

ABSTRACT

This paper introduces a tool called the DPM (Designer Preference Modeller) that analyzes the designer's decision making during concept evaluation, and constructs a designer preference model to be used for evaluation of automatically generated design alternatives. The method is based on establishing an interaction between a designer and a computational synthesis tool during conceptual design. The synthesis software generates design alternatives using a catalogue of design knowledge formulated as grammar rules which describe how electromechanical designs are built. DPM carefully selects a set from these alternatives and presents it to the designer for gathering evaluation feedback. The designer's evaluations are translated into a preference model that is subsequently used for automatically searching the solution space for best designs. Application of the method to the design of a consumer product shows DPM's range of capabilities.

Keywords: concept generation, design automation, sampling, design selection, grammar rules.

1 INTRODUCTION

Design generation and evaluation are two tightly interconnected processes of conceptual design. Finding a good solution usually requires an in-depth search of the design space, often necessitating generation of as many design alternatives as possible. These alternatives are then evaluated against various design requirements, constraints, and objectives to determine which alternatives are most useful for advancing towards successful designs and which alternatives have little or no design worth. Often, these two processes occur over many iterations until a satisfactory design solution is found. A closer look at the traditional and computer-based conceptual design methods reveals that computers and humans have distinct characteristics regarding design generation and evaluation [1]. Humans are very good at comparing and evaluating solutions of various complexities by using sophisticated reasoning methods, however, it is impractical for them to generate all solution alternatives in a design space. In contrast, computers are well suited for generating numerous design alternatives, thanks to their computational speed, but they lack the judgement employed in human decision making to be able to effectively evaluate them.

In this paper, we introduce a method that brings the designer and the computer together in order to leverage the strengths of both in generating and evaluating conceptual design alternatives. The objective of this research is to establish an interaction between a designer and a computational synthesis tool so that the designer's decision-making during concept evaluation can be analyzed, modelled, and later used for faster search of larger design spaces. Accordingly, we have developed a tool called DPM (Designer Preference Modeller) that facilitates communication between a designer and a computational synthesis software as shown in Figure 1. The computational synthesis tool generates design alternatives using a catalogue of design knowledge formulated as grammar rules which describe how electromechanical designs are built. The designer, on the other hand, gets involved in the process by evaluating a prescribed set from these design alternatives. In order to get the synthesis software and the designer to interact, DPM carefully selects this set from the population of candidate designs and presents them to the designer for gathering evaluation feedback. This selection is made by following a heuristic that aims to simultaneously reduce the number of required designer evaluations and capture the variety in the design solution space. The designer's feedback is

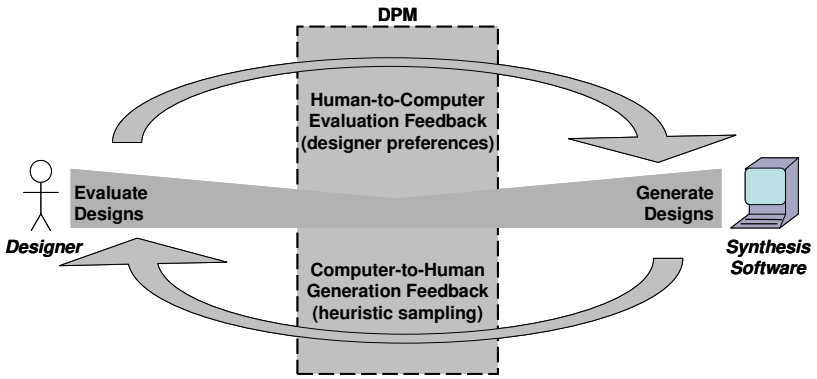

Figure 1. DPM brings the designer and the computational synthesis tool [2] together so that the designer's decision-making during concept evaluation can be modeled.

translated into a preference model that is used to automatically search for best designs. Because the preference model is built directly from the evaluations of the designer, it reflects the designer's reasoning and judgement under specific design objectives and constraints. The main advantage of the presented method is that it allows the construction of a designer preference model from limited number of designer evaluations, which then can be used for evaluating large number of design alternatives.

The research presented here extends our previous work that integrates empirical knowledge acquisition techniques and graph-grammar-based methods into a framework that facilitates the automation of conceptual design of electromechanical systems [2]. The basis of this automated design generation method is that knowledge is extracted from past designs, stored as procedural rules, and then employed in building solutions to conceptual design problems. In the research presented here, we use the DPM tool to evaluate these automatically generated design alternatives. The rest of this paper presents the state of the art in design generation and evaluation (Section 2) and explains the approach to our automated concept generation. (Section 3) This is followed by a description of the method used by the DPM (Section 4) and a presentation of its use through an illustrative example (Section 5). The results of this example are discussed in Section 6, and the paper concludes with a summary of the method.

2 RELATED WORK

2.1 Generation Methods in Design

Concept generation research has traditionally focused on developing methods that improve the quality and variety of ideas generated. These methods are often kept simple and efficient such that designers are not burdened by the details or limitations of the method. The most common concept generation method is known as brainstorming [3]. The term brainstorming is frequently applied to any idea generation technique. Brainstorming as a specific method requires a group of individuals to follow the basic rules of (1) avoiding criticism, (2) welcoming "wild ideas", (3) building on one another's ideas, and (4) preferring more ideas than dwelling on specific ones. Designing by analogy is a well accepted approach to arrive at novel design solutions. It can be accomplished by first generalizing the design problem to a set of functions (or a graph of functions as in the function structure representation). Then one can look for or conceive analogous products or components that perform the same set of functions [4,5]. Function-means trees and Morphological Analysis [6] are similar methods in which solutions to individual functional requirements are first sought and then synthesized together. Apart from these approaches, one widely used method is the Theory of Inventive Problem Solving [7]. This method provides a tabulated representation of a large number of solution principles that have been extracted from existing patents. Another approach is "catalog design" where concepts are generated purely through browsing a catalog of physical elements (components, assemblies, etc.). The results are evidently limited by the breadth of the catalog; however, the benefit lies in the presentation of design knowledge that falls outside the designer's expertise memory [4].

Various approaches have been attempted to solve conceptual design problems with computational methods. Examples include the agent-based system presented in the A-Design research [8] and the catalog design method used in Chakrabarti and Bligh [9]. In both these approaches, input-output characteristics of component elements are used to synthesize a system level design by integrating individual component models according to high-level design requirements. Some methods, on the other hand, rely on representations that manage and manipulate functional descriptions (detailed description of what a design should do) which are later converted into configurations of components. As is shown in Pahl and Wallace [10], a representation of functions allows for a richer set (or a form-independent set) of design principles to be captured. Bracewell and Sharpe [11] used the bond graph formalism as a foundation in their "Schemebuilder" tool to automatically explore alternative conceptual schemes and appropriate allocation of function between electromechanical components. Following a similar function-based approach, Bryant, et al. [12] developed a concept generation technique that utilizes a repository of existing design knowledge and a set of matrix-manipulation algorithms. Finally, graph grammar based methods provide a flexible yet structured approach to engineering design synthesis [13]. The concept of a grammar is that a set of rules is constructed to capture specific domain knowledge about a certain type of artifact. For example, the rules can encapsulate a set of valid operations that can occur in the development of a design. Grammar techniques create a formal language for generating and updating complex designs from a simple initial specification, or a seed. Our aforementioned approach to computational design generation [2] follows the grammar formalism, and combines it with function based synthesis methods [14]. Accordingly, it uses a functional description of a design as a seed and seeks multiple configuration-based solutions that address the functional requirements.

2.2 Evaluation and Selection Methods in Design

Design evaluation and selection is an important part of the design process and has received great attention in the design literature. In this section, we provide a review of various decision-based design techniques [15] and describe traditional and computer-based methods for evaluating a pool of conceptual solution alternatives and selecting the ones most likely to produce best solutions.

Perhaps, the most common concept evaluation method is the Pugh Concept Selection Charts [16]. Pugh charts use a minimal, qualitative evaluation scale and compare design alternatives in a matrix format against a number of performance criteria. Pugh Charts provide an effective known tool for preliminary concept selection when there is minimal information available about the potential design solutions. Numerical concept scoring/weighting, and decision matrices [17] are similar methods and only vary in the representation of the nature (qualitative vs. quantitative) and the resolution of their evaluation scales. The Analytic Hierarchy Process (AHP) [18] is another multi-criteria decision making technique that uses hierarchically related performance metrics. Similar in spirit to AHP, some researchers have proposed methods that are inspired from the multi-attribute utility theory [19]. The general method for these techniques is to assign a value for each performance metric, weight the value by the importance of the metric, and then aggregate the weighted scores to convert multiple metrics into a single metric. Application of these methods to design evaluation include formulation of non-linear utility functions capturing multi-attribute aspects of engineering problems [20], physical programming [21], and set-based techniques that rely on fuzzy logic to represent imprecision in design and to conduct design evaluation [22]. Finally, at later stages of design where simulation data is available, computational analysis tools such as finite-element-methods (FEM), computational fluid dynamics (CFD), etc. offer accurate and robust evaluation of design alternatives. The integration of these different analysis tools into a single, robust evaluator which can negotiate multiple attributes of a design problem remains a challenging topic and is tackled by the field of multi-objective design optimization.

2.3 Learning-Based Methods in Design

Examples of learning-based computational design tools include the Learn-It [23] and its temporal extension Learn-It-II [24] systems which observe a designer's actions and use an instance-based technique to learn the design strategy employed. Both these systems are intended for parametric design problems in which the designer iteratively adjusts the parameters of a design to meet specific design requirements. The learned strategies are later used to automatically generate design solutions when the design requirements change. Myers et al. [25] have created a system that monitors a

designer's interactions with a CAD tool in order to automatically produce design documentation. This system however, does not perform automated design. Moss et al. [26] integrated a learning mechanism to the agent-based computational design tool called A-Design [8] in order to enable the system to learn from its own design generation experiences. However, A-Design does not involve the designer in the process and its learning scheme is aimed at improving the quality of designs generated by its agents.

3 A GRAPH GRAMMAR FOR AUTOMATED CONCEPT GENERATION

In this section, we explain our approach to automated concept generation and describe how we use a graph grammar to automatically create new design configurations starting from a set of functional requirements. A detailed description of this work can be found in Kurtoglu et al. [2].

In developing this method, we took the common view of design which models the design process as a transformation of function to form. Accordingly, the framework includes representations that capture the designs at these two levels of abstraction. In this framework, a design's *function* is represented using function structures [10], whereas its *form*, or configuration, is represented by the use of configuration flow graphs (CFG's) [2]. A CFG is a representation that shows the connectivity or topology of components in a design. In a CFG, nodes of the graph represent a design's components, and arcs represent energy, material or signal flows between them. For flow naming, the functional basis [27] terminology is adopted, while the components of the graph are named using the component types of a user-defined taxonomy of electromechanical components [28]. These component types can be thought of as generic abstractions of common component concepts (gear, shaft, wire, dc motor, battery, etc.) without specific geometric details. The construction of a CFG allows a conceptual design to be expressed as a configuration in a graphical, topology-based format. (A CFG can also be interpreted as a conceptual sketch, or schematic.) The creation of a function structure (FM) and the corresponding configuration flow graph (CFG) captures a direct mapping between the functional and the structural architecture of a design and constitutes the foundation of the graph grammar that represents the transition or production rules for creating conceptual configurations from functional specifications.

In our computational approach to concept generation, we utilize an expanding online repository that contains knowledge about past designs [29], and we derive design rules from it that capture the decisions of the original designer in mapping functional specifications to component solutions. In deriving the rules for the graph grammar, we first build an existing design's CFG and its function structure. This is illustrated in Figure 2. The figure shows the function structure and the configuration flow graph of a past design from the repository. We, then capture the mapping between the two graphs. Each mapping represents a potential grammar rule that shows how a functional requirement was transformed into an embodied solution in the actual design. Some of the rules derived from the analysis of the aforementioned design are also shown at the end of Figure 2. In the first rule, the rule states that the functional requirements of "convert rotational mechanical energy to translational mechanical energy" and "transfer translational mechanical energy" are addressed in the design by the use of the component "link". Similarly, the last rule shown in Figure 2 indicates that the function "transfer RME" in the function structure is solved by a "driveshaft" and a "rotational coupler" in the actual design. Following this procedure, we currently have defined 189 rules derived from 23 products.

The grammar provides a method to generate design configurations through the execution of rules that create feasible solutions to the design problem. These solutions are encoded as configuration flow graphs. Our automated synthesis method is to perform a graph transformation of the initial function structure into one or more configuration flow graph. To perform this graph transformation, the grammar rules are defined to add components to the CFG that maintain a valid connection of components as well as meet specific function requirements specified with the function structure. Each of the rules developed are modelled after basic grammar conventions where rules are compromised of a left hand side (LHS) and right hand side (RHS) as illustrated in Figure 2. The left-hand side contains the state that must be recognized in the function structure and the right-hand side depicts how the design is transformed to a new configuration by the addition of new component(s). The basic generation process for a set of rules is to first recognize which rules can be applied given the current state of a design, then choose one of the applicable rules, and finally to apply the rule as a step towards

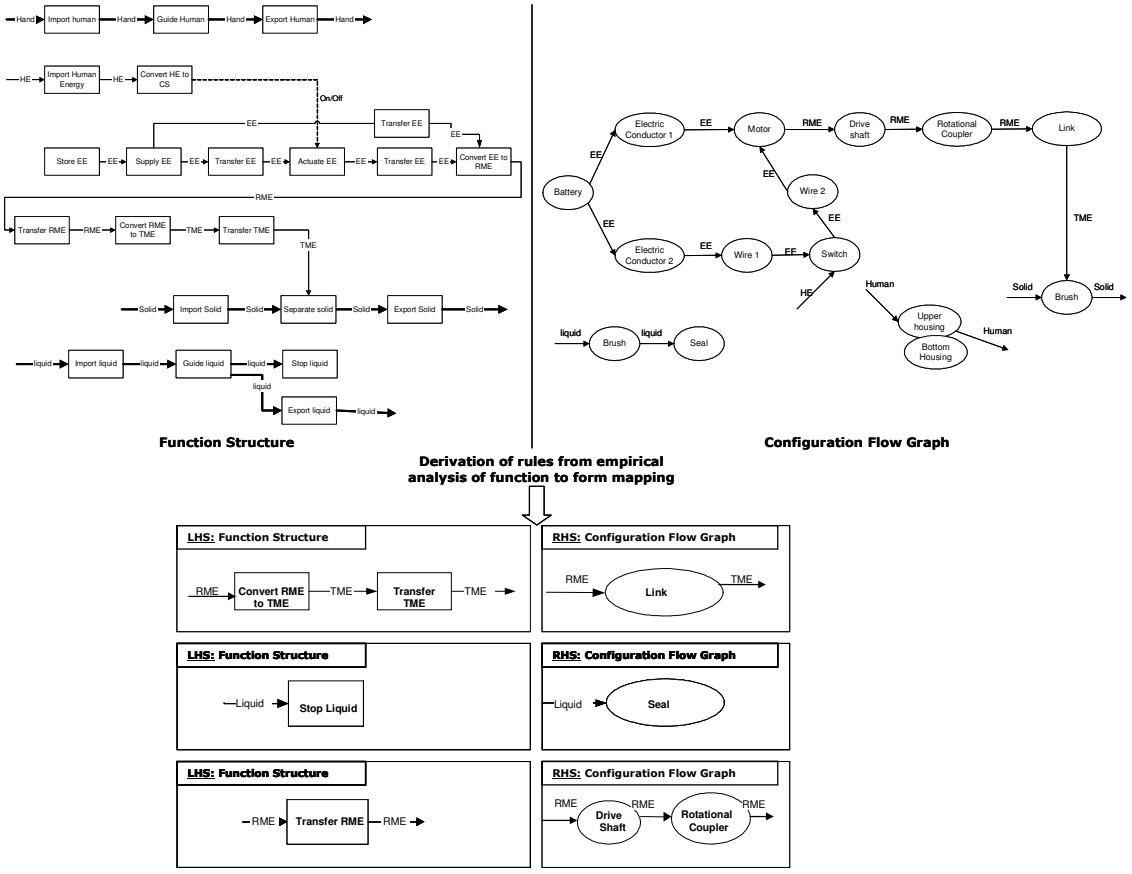

Figure 2. The function structure and the configuration flow graph of a past design, along with three grammar rules derived from its analysis.

constructing an updated configuration, and a new design state. This cycle is repeated until no further rules can be recognized, indicating that all functional requirements given by the function structure are addressed and that the configuration synthesis is complete. This process is shown in Figure 3 for the design of a "bread slicer". At the beginning, none of the sub-functions of the initial function structure are mapped to components. As the rules are applied, the CFG is incrementally updated by the addition of new components. This is illustrated in the figure with four snapshots between start to finish. In between the snapshots, we have listed the grammar rules that are applied. The result of this process is the completed design configuration shown at the end of the figure. Each design configuration is represented by a list called the *recipe* that captures the sequence of rules that are used in the construction of that particular configuration. For example, the recipe for the CFG shown at the bottom of Figure 3 is {39, 31, 13, 11, 28, 25, 38, 43}. Depending on the grammar rules chosen to create the CFG, the recipes of different solutions may have potentially different lengths to their lists. The use of the recipes by the DPM is explained in the next section.

4 CONCEPT EVALUATION USING THE DPM

Design evaluation provides a means of determining the 'value', 'usefulness', or 'strength' of a design solution with respect to a given objective [14]. An evaluation requires a comparison of concept variants, or a rating of solutions based on an idealized design solution. DPM facilitates the evaluation of design concepts using an interactive approach that involves the designer in this critical process. In terms of our specific implementation, the designer's preferences are captured as preference scores for rules that are used during the creation of designs. In other words, DPM models how much the designer prefers a particular rule over others in the grammar, or synonymously how much the designer likes a particular function-to-component mapping as described by that rule.

DPM carries out the construction of this model by employing a sampling strategy that selects and presents the designer with a small-set of proposed solutions. It performs this sampling by following an

algorithm that aims to simultaneously reduce the number of required designer evaluations and capture the variety in the design solution space. The goal of the sampling is to find the minimum set of solutions, recipes of which represent the most diverse collection of the rules (i.e. solution principles, or components). After completing the sampling, DPM presents the selected solutions to the designer for evaluation. The designer is asked to make pairwise comparisons between the selected designs. These solutions are then rated by the designer based on specific design objectives and constraints. After the solutions are scored, the solution ratings are projected on to the rules. Using this projection, a "preference score" is assigned to each rule. The collection of these preference scores constructs a model of designer preferences. Finally, DPM uses the model of designer preferences to guide the search for finding the "best" configurations in the population of generated solutions. To achieve that, preference scores of rules are propagated over each recipe to compute the overall worth of each design in the population. Before we present the details of the process, we provide some definitions that are used in the formulation of the method.

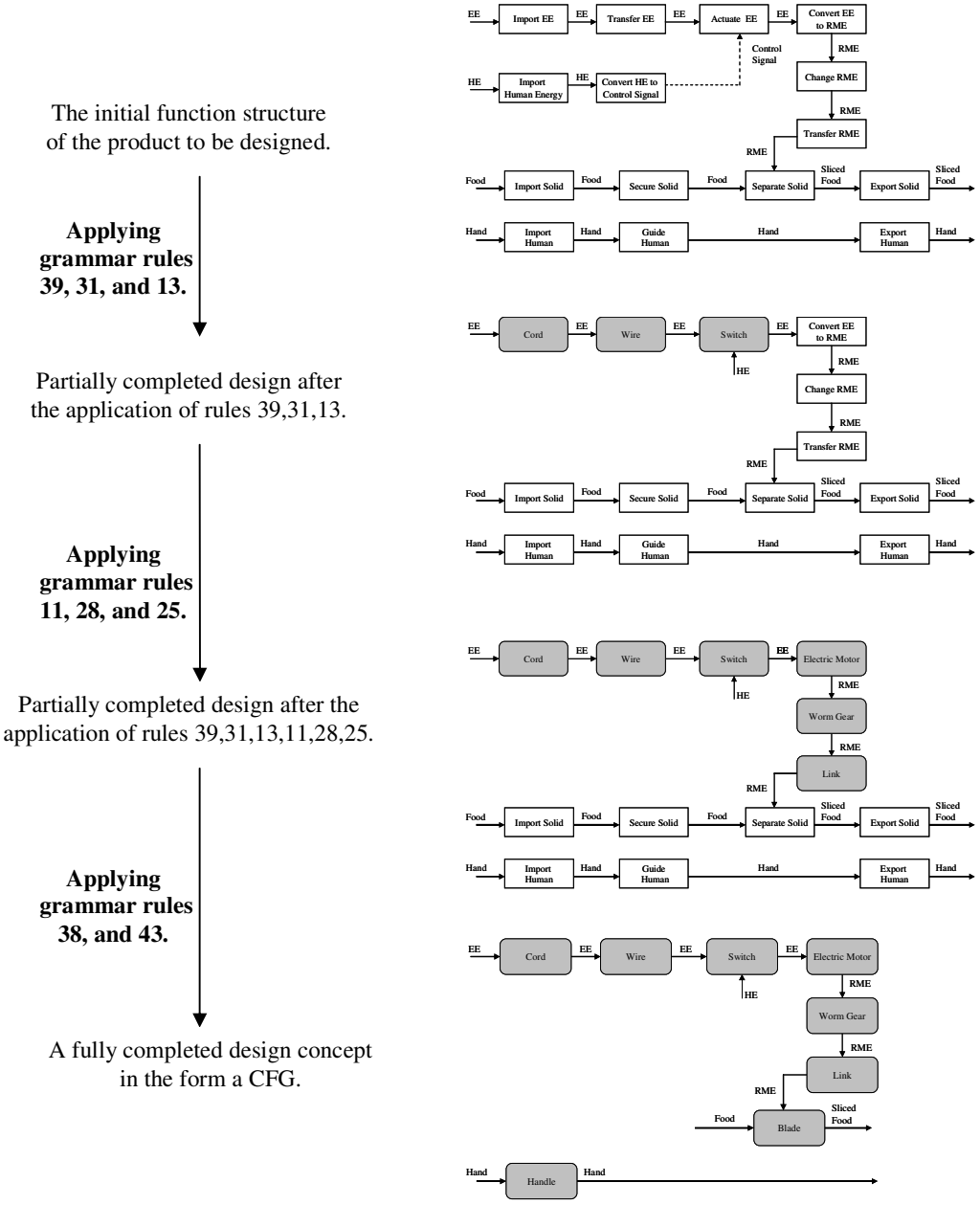

Figure 3. A pictorial illustration of building a CFG from a function structure using the grammar rules.

$c_i \in CP$ is a *candidate*:

An automatically generated configuration (CFG) as a solution to the design problem.

$CP = (c_1, c_2, ..., c_i, ...c_n)$ is the *candidate pool*:

The collection of all candidates where *n* is the number of candidates generated.

$DFS = (c_i, c_j, ..., c_p)$ is the *designer feedback set:* where $DFS \subseteq CP$

The set of candidates selected to be presented to the designer for evaluation feedback.

r_j is a *rule*:

A grammar rule (function-to-form mapping) used in the generation of a candidate.

$\forall c_i \in CP : \exists! R_i$ where $R_i = (r_k, r_l, ..., r_s)$ is the *recipe* of candidate c_i:

The collection of rules used in the generation of candidate c_i.

$RP = R_1 \cup R_2 \cup R_3 ..... \cup R_n$ is the *recipe pool*:

The union of recipes of all candidates where *n* is the number of candidates generated.

$CDFS = [s_i, s_j, ..., s_p]$ is the *cumulative designer feedback score* vector*:* where $s \in \Re$

The vector of cumulative scores of candidates in DFS after designer evaluations.

$RPS = [s_{r1}, s_{r2}, ..., s_{rp}]$ is the *rule preference score* vector*:* where $s_{r_j} \in \Re$

The vector of preference scores of rules in RP.

$CS = [cs_i, cs_j, ..., cs_p]$ is the *candidate score* vector*:* where $cs \in \Re$

The vector of candidate scores of candidates in CP.

4.1 Sampling Strategy

The goal of the sampling is to select a set of candidates from the candidate pool. This set called the DFS will be presented to the designer for obtaining evaluation feedback. As one might expect, there may be countless possibilities for choosing the DFS. Here, DPM employs a *heuristic* sampling strategy that strives to strike a balance between two objectives. First, it aims to choose candidates that will keep the number of designer evaluations to a minimum. It accomplishes this by taking advantage of the commonality of candidates in the candidate pool. Accordingly, DPM selects candidates with a high "commonality measure". Simultaneously, it targets candidates that will best represent the variety in the solution space. To achieve the latter, DPM keeps track of a "variety measure" for the rules that constitute the recipes of the candidates in the DFS and only allows addition of the candidates that increase the variety of rules in the DFS. By monitoring these two measures, DPM continues to select candidates from the candidate pool until all rules in the recipe pool are represented in the DFS.

To better understand the sampling approach, consider the example shown in Figure 4.a. This is a real design example illustrating the synthesis of a switch assembly in a soda mixer. The design tree shows the generation of the candidates. The arc labels in the tree correspond to the grammar rule that is applied during the transition between any two nodes of the tree. The basic steps in applying the heuristic sampling strategy are as follows:

1. Compute the candidate pool (CP), and the recipe pool (RP).

As shown in Figure 4.a., the final design configurations, i.e. candidates, are represented at the leaves of the tree and numbered as C1-C9. The recipes of the nine candidates and the recipe pool are summarized in Figure 4.b.

2. Create a histogram of the frequency of rules in RP.

Figure 4.c shows the "frequency of appearance" of the rules at each iteration. Rule#3 is used in the construction of all nine designs and has the highest frequency with 9 in the first iteration.

3. Calculate the commonality measure of each candidate by summing frequencies of rules in the recipe of the candidate.

Figure 4.d shows the "commonality measure" of the candidates at each iteration. In the first iteration, C4-C9 has the highest commonality measure with 21.

4. Select the candidate with the highest commonality measure and add it to the DFS,

Among the four candidates with the highest commonality measure, C9 is (randomly) selected and added to the DFS.

5. Remove the rules of the selected recipe from the recipe pool.

After the selection of C9 to the DFS, the rules in the recipe of this candidate (3, 10, 31, 35) are removed from the recipe pool. As can be seen in Figure 4.c., the frequencies of the rules 3, 10, 31, and 35 following the first iteration are zero. This ensures that there will be minimum overlap between C9's recipe and the recipes of future selections. This modification is a necessary step for minimizing the number of candidates included in the DFS at the end of the sampling.

6. Repeat steps 1-5 until no rules remain in the recipe pool.

Following a similar scheme, C6 is selected in the second (with a commonality measure of 6), and C1 is selected in the third (with a commonality measure of 6) iteration.

The result of the sampling process is the DFS containing C9, C6, and C1. This DFS satisfies the two objectives of the heuristic sampling of finding a set with minimum number of candidates, recipes of which represents the most diverse collection of the rules. Accordingly, the generated DFS captures all the rules in the recipe pool by the selection of only three candidates.

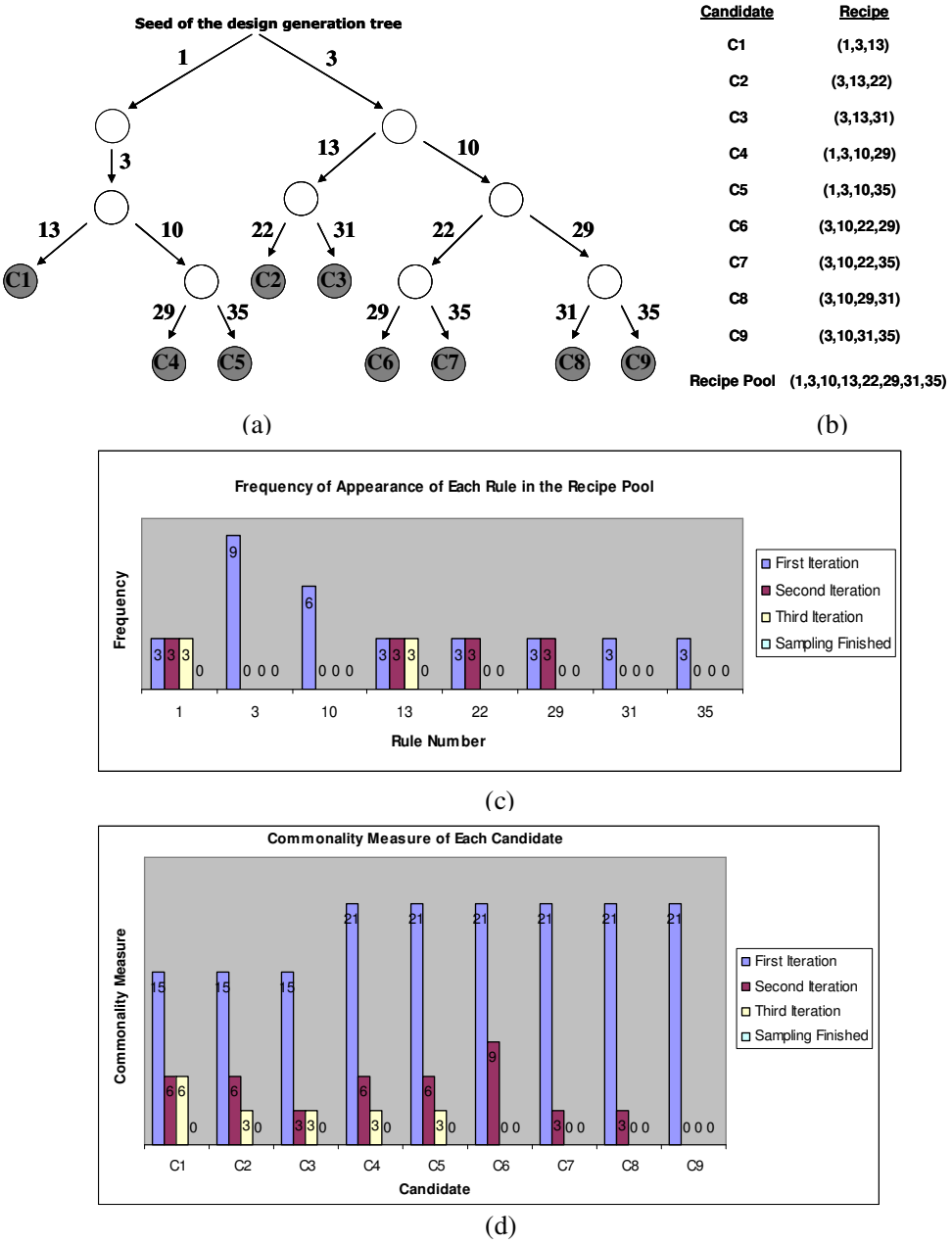

Figure 4. An example of the application of the sampling strategy.

4.2 Designer Feedback

The sampling is followed by gathering the designer's feedback. To achieve this, DPM presents the candidates in the DFS to the designer for evaluation. The designer is asked to make pairwise comparisons between the selected candidates. The evaluation scale utilized during this concept scoring is similar to the one used in decision matrices [17], and contains the ratings {-3, -2, -1, 0, 1, 2, 3}. When element *i* compared to *j* is assigned one of the aforementioned ratings, then element *j* compared to *i* is assigned its negative. The GUI for this step is shown in Figure 5. The overall worth of each candidate as a result of this evaluation, called the cumulative designer feedback score (CDFS), is calculated by simply adding the values in the cells of the column that corresponds to that particular candidate.

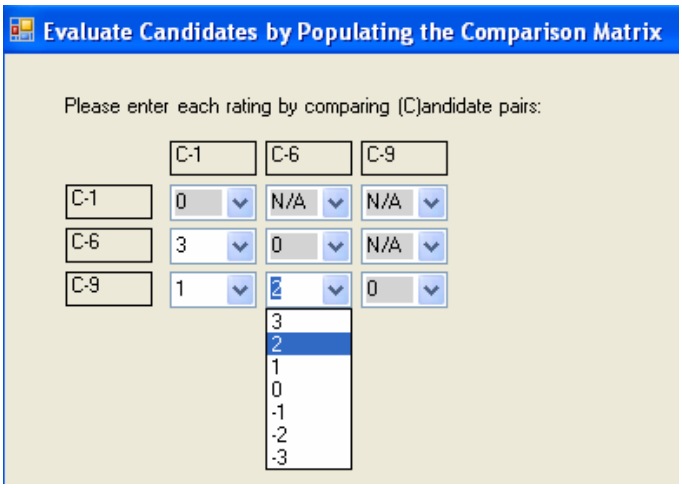

Figure 5. The GUI of the pairwise comparison matrix used during designer evaluations.

Considering the values in Figure 5, the cumulative designer feedback score vector (for the candidates C1, C6, and C9) becomes [4, -1, -3].

4.3 Construction of the Designer Preference Model

After the designer completes the pairwise comparisons between the candidates, the candidate ratings are projected onto the rules in the recipe pool to compute a "preference score" for each rule. The collection of these preference scores constructs a model of designer preferences as represented by the rule preference score (RPS) vector. The basic steps in the computation of the RPS vector are as follows.

1. Calculate the cumulative designer feedback score s for each candidate in the DFS by summing each column in the comparison matrix,
2. For each rule in the recipe of a candidate, add the cumulative designer feedback score of that candidate to the rule score,
3. Repeat step 8 for each candidate to calculate the final rule preference score s_r,
4. Aggregate the rule preference scores to construct the RPS vector.

Considering the same example shown in Figure 4 and taking the cumulative designer feedback scores calculated in the previous section, the value of 4 is added to the rule score of the rules in the recipe of C1 (rules 1,3, and 13). Similarly, the value of -1 is added to rule score of the rules 3,10,22,and 29 and the value of -3 is added to the rule score of the rules 3,10,31,and 35. Aggregating these values defines the RPS vector, which then becomes: RPS=[4, 0, -4, 4, -1, -1, -3, -3] corresponding to the "preference scores" for rules 1, 3, 10, 13, 22, 29, 31, 35. The RPS vector represents a model of the designer's preference for each rule in the recipe pool. In this particular example, the designer prefers rule#1 and rule#13 the most, and rule#10 the least.

4.4 Automated Concept Scoring Using the Designer Preference Model

DPM uses the model of the designer preferences to automatically search for the best designs in the population of generated candidates. This is performed by propagating the preference scores of rules over each candidate recipe in the population. Specifically, by summing over the individual rule preference scores in a recipe, the worth of each candidate can be computed. Finally, the aggregation of these scores constitutes the candidate score (CS) vector. After all candidates are evaluated, they can be ranked or sorted for identifying the best or the worst designs in the population.

Returning to the switch assembly design, and using the RPS vector shown before, all of the nine candidates in the population can be evaluated. The CS vector for this example then becomes: CS=[8, 3, 1, -1, -3, -6, -8, -8, -10] corresponding to the "candidate scores" of candidates C1-C9. The best three designs in this population are C1, C2, and C3. Note that in evaluating the candidates, DPM preserves the designer's original ranking among the candidates in the DFS, i.e. the relation C1 > C6 > C9 holds true after the DPM evaluates the candidates using the designer preference model.

5 ILLUSTRATIVE EXAMPLE: DESIGN OF A BREAD SLICER

We have tested DPM's performance on three electromechanical design problems: design of a bread slicer, design of a wall climber toy product, and the design of a bottle capping machine. In this section, we consider the first of these problems in detail to demonstrate DPM's capabilities.

The design process starts with the specification of the function structure of the system to be designed. This is accomplished through a graphical user interface that allows the designer to quickly draft a function structure. The function structure that is used for the example problem is the same one shown at the top of Figure 3, and includes 15 sub-functions, and 19 flows. After the concept generator initiates the design generation process, the sub-functions are replaced with components through incremental application of the grammar rules until the design space is exhaustively searched for all possible design configurations. This generation process creates 594 valid design configurations.

After the candidate pool is generated, DPM employs its sampling algorithm and selects 12 candidates from the candidate pool. These candidates are added to the DFS and are presented to the designer for evaluation feedback. The designer evaluates the 12 candidates based on the criteria of *quality of concepts.* Finally, the rule preference score (RPS) and the candidate score (CS) vectors are computed by the DPM. The result of this process is summarized in Table 1, and the best and the worst candidates determined by the evaluation are shown in Figure 6.

Looking at these two candidates, one can see that the design on the left has a number of superior features when compared to the design on the right. First, it includes a *knob* component to activate the switch which can be used for setting the speed of the slicer. Second, the power transmission from the electric motor to the blade is much better developed in this design. It includes a *gear box* for speed reduction, and a *rotational coupler* that secures a *blade* to the *output shaft* of the gear box. Moreover, in this design, the *blade* is supported by an outer *housing* that helps with importing and securing of the bread into the device. This was a feature that the designer particularly favoured while providing evaluation feedback for reasons regarding to the customer needs of "safety" and "ability to accommodate different slice sizes". On the contrary, in the other design, the importing and securing of the bread are addressed by the blade itself, similar to how an electric kitchen knife functions.

As this case illustrates, DPM is capable of generalizing a small number of designer evaluations into a preference model, which it later uses to evaluate a large range of automatically generated design alternatives.

6 CONCLUSIONS

We have described a tool called the DPM (Designer Preference Modeller) that analyzes the designer's decision making during concept evaluation, and constructs a designer preference model to be used for evaluation of automatically generated design alternatives. The method is based on establishing an interaction between a designer and a computational synthesis tool during conceptual design.

The main novelty of the DPM approach is the combination of computing and human reasoning. Computational design tools can generate numerous solutions to a design problem by conducting an in-depth search of the design space. But, they typically require very detailed specifications of part shapes and dimensions in order to evaluate certain performance parameters. At the conceptual stage of design, however, such detailed models of system components and design parameters are simply not available and the evaluation of conceptual design alternatives remains a stumbling block in computational

Design Problem			
Example Problem	*Bread Slicer*		
Evaluation Criteria	Quality of Concepts		
Design Generation and Sampling			
# of candidates generated	**594**		
# of rules in the recipe pool	30		
# of candidates in DFS	12		
Design Evaluation			
Recipe Pool	1,7,8,10,11,12,13,15,17,20,21,22,23,24,25,26, 28,29,30,31,33,34,35,36,38,39,40,41,42,43		
RPS	-3,3,14,3,0,6,-3,7,-21,-7,15,3,24,-9,-14,7, -3,3,-9,0,2,-9,0,3,0,0,9,9,9,-9	***Recipe***	***Score***
Best Candidate	C155	10,11,15,22,23,29,38,39,40,41,42	**67**
Worst Candidate	C498	1,8,11,13,17,38,39,43	**-50**

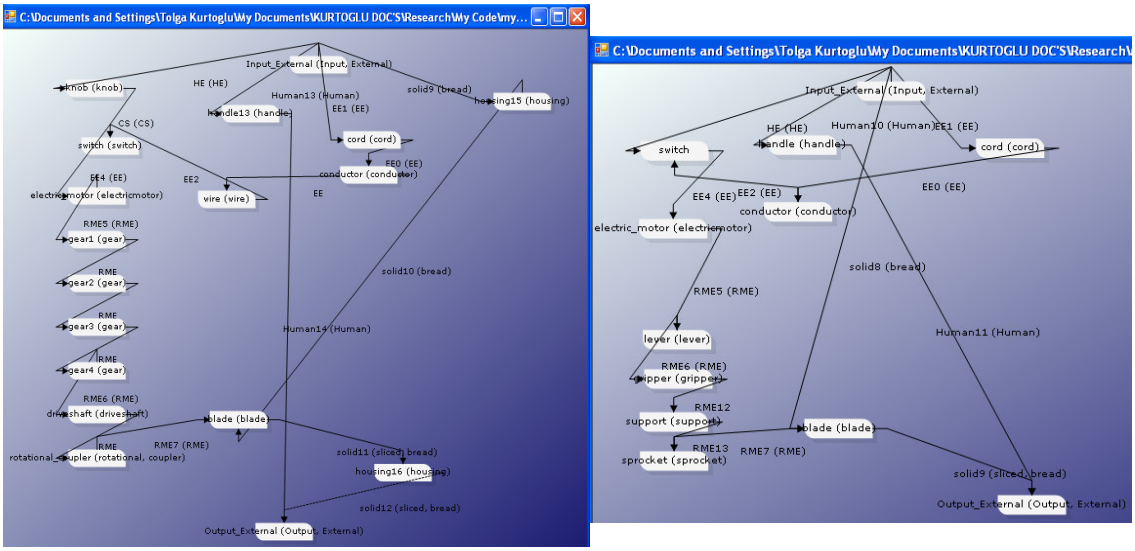

Figure 6. Summary of DPM's evaluation results and the best (left) and the worst (left) candidates in the population.

synthesis. Humans, on the other hand, can employ sophisticated reasoning techniques in making decisions for ruling out or approving design alternatives, however they are bounded by the limits of their own rationality [1]. This makes it impractical for them to process and evaluate more than a handful of alternatives. The heuristic followed by the DPM takes advantage of the commonality of solutions in a design space and ensures that the required number of designer evaluations is kept to a minimum, consistent with the principle of bounded rationality. By establishing a proper level of interaction between the designer and the synthesis software, DPM combines the strengths of computational search with that of human decision-making. Moreover, since the preference model is directly derived from the evaluations of the designer, it reflects the designer's reasoning and judgement under specific design objectives and constraints which is easily generalized and used for faster search towards the best designs in large design spaces.

REFERENCES

[1] Simon, H. A., 1969, The Sciences of the Artificial, The MIT Press, Cambridge, MA.

[2] Kurtoglu, T., Campbell, M.I.,Gonzales, J., Bryant, C.R., McAdams, D.A., Stone, R.B., ,2005, "Capturing Empirically Derived Design Knowledge for Creating Conceptual Design Configurations," Proceedings of DETC2005, Sept. 24-28, Long Beach, California.

[3] Osborn, A., 1957, Applied Imagination. Scribner, New York, NY

[4] McAdams, D. and Wood, K., 2000, "Quantitative Measures for Design By Analogy," DETC2000/DTM-14562, Proceedings of DETC2000, Balitmore, MD.

[5] Linsey, J.S., Green, M.G., Murphy, J.T., Wood, K.L., Markman, A.B.,2005, "Collabrating to Success: An Experimental Study of Group Idea Generation Techniques"DETC2005, Long Beach, CA.

[6] Zwicky, P., 1969, Discovery, Invention, Research through Morphological Analysis, McMillan, New York.

[7] Altshuller, G., 1984, Creativity As An Exact Science, Gorden and Breach, Luxembourg.

[8] Campbell, M., J. Cagan and K. Kotovsky, 2000, “Agent-based Synthesis of Electro-Mechanical Design Configurations,” Journal of Mechanical Design, Vol. 122, No. 1, pp. 61-69.
[9] Chakrabarti, A., Bligh, T. P., 1996, "An Approach to Functional Synthesis of Mechanical Design concepts: Theory, Applications, and Merging Research Issues," AIEDAM, Vol.10, 313-331.
[10] Pahl, G. and Wallace, K., 2002, Using the Concept of Functions to Help Synthesize Solutions, Springer, London.
[11] Bracewell, R.H.,and Sharpe, J.E.E., "Functional Descriptions Used in Computer Support for Qualitative Scheme Generation—Schemebuilder," AIEDAM, Vol. 10, No. 4, 1996, pg. 333-346.
[12] Bryant,C, Stone, R., McAdams, D., Kurtoglu, T., Campbell, M., 2005, “A Computational Technique for Concept Generation”, ASME IDETC’05. Long Beach, CA.
[13] Cagan, J., 2001, “Engineering Shape Grammars,” Formal Engineering Design Synthesis, Antonsson, E. K., and J. Cagan, eds., Cambridge University Press.
[14] Pahl, G., and Beitz, W., 1996, Engineering Design—A Systematic Approach, 2nd edition, Springer, London.
[15] Hazelrigg, G. (1996). Systems Engineering: An Approach to Information-Based Design. Prentice Hall, Upper Saddle River, NJ.
[16] Pugh, S. (1991). Total Design: Integrated Methods for Successful Product Engineering. Addison-Wesley Publishing Company, Workingham, UK.
[17] Ullman.D. 1995.The Mechanical Design Process. NewYork:McGraw-Hill.
[18] Saaty, T. (1980). The Analytic Hierarchy Process, McGraw-Hill.
[19] Keeney, R. L., and Raiffa, H., 1976. Decisions with Multiple Objectives: Preferences and Value Tradeoffs, John Wiley & Sons, New York.
[20] Thurston, D. L., 1991, “A Formal Method for Subjective Design Evaluation with Multiple Attributes” Research in Engineering Design, Vol. 3, pp. 105-122.
[21] Messac, A. (1996). Physical programming: effective optimization for computational design. AIAA Journal, Vol. 34(1), 149–158.
[22] Wood K.L, Antonsson, E.K., Beck, J.L. 1990, “Representing Imprecision in Engineering Design: Comparing Fuzzy and Probability Calculus”, Research in Engineering Design, 1990.
[23] Stahovich, T.F., 2000, “LearnIT: an instance based approach to learning and using design strategies,” ASME Journal of Mechanical Design, Vol 122, No.3, pp.249-256.
[24] Rawson, K., Stahovich, T.F., 2006, “A Method for Inferring Design Rules with Explicit Bounds of Applicability” ASME IDETC’06. Philadelphia, PA.
[25] Myers,K.L., Zumel, N.B., Gracia, p., 1999, “Automated Rationale Capture for the Detailed Design Process,”, In: 11’th Conference on Innovative Applications of AI.
[26] Moss J., Cagan, J., Kotovsky k., 2004“Learning from Experience in an Agent Based Design System”, Research in Engineering Design, Vol 15, pp:77-92.
[27] Stone R. and Wood K., 2000, “Development of a Functional Basis for Design,” Journal Mechanical Design, Vol. 122 pp.359-370.
[28] Kurtoglu, T., Campbell, M., Bryant,C, Stone, R., McAdams, D., 2005, “Deriving a Component Basis for Computational Functional Synthesis” Proceedings of ICED’05, Melbourne, Australia.
[29] Bohm, M. and Stone, R., 2004, Product Design Support: Exploring a Design Repository System, Proceedings of IMECE’04, IMECE2004-61746, Anaheim, CA.

Contact: Tolga Kurtoglu
Department of Mechanical Engineering – University of Texas at Austin
1 University Station, C2200, Austin, TX 78712-0292, USA
Phone: +1-512-471 7347
Fax: +1-512-471 7682
e-mail: tolga@mail.utexas.edu