A Framework for Computational Design Synthesis: Model and Applications

Jonathan Cagan[1]
e-mail: cagan@cmu.edu
Phone: (412) 268-3713
Fax: (412) 268-3348
Carnegie Mellon University, Pittsburgh, PA

Matthew I. Campbell
University of Texas, Austin

Susan Finger
Carnegie Mellon University, Pittsburgh, PA

Tetsuo Tomiyama
Delft University of Technology,
The Netherlands

The field of computational design synthesis has been an active area of research for almost half a century. Research advances in this field have increased the sophistication and complexity of the designs that can be synthesized, and advances in the speed and power of computers have increased the efficiency with which those designs can be generated. Some of the results of this research have begun to be used in industrial practice, yet many open issues and research challenges remain. This paper provides a model of the automated synthesis process as a context to discuss research in the area. The varied works of the authors are discussed as representative of the breadth of methods and results that exist under the field of computational design synthesis. Furthermore, some guidelines are presented to help researchers and designers find approaches to solving their particular design problems using computational design synthesis. [DOI: 10.1115/1.2013289]

1 Introduction

Possibly the first design synthesis programs written, which were used in industrial practice as well, were a set of expert systems from Westinghouse written in the 1950s that designed electric motors, generators, and transformers. It is known to the world only because Herb Simon spoke of it when talking about the history of Artificial Intelligence [1]. But, arguably, it was Simon's 1969 paper, "The Science of Design," published in his book *The Sciences of the Artificial*.[2], that provided a foundation for the academic pursuit of automated engineering synthesis methods. Ten years after that paper was written, the Computers in Engineering division of ASME was founded and, now 35 years later, we see a strong base of research in the field and the beginnings of the transfer of this work into industrial practice. Some early work on automated synthesis that followed the arguments from Simon included the DOMINIC system [3], a heuristic-based iterative optimizing design process method, and PRIDE [4], an expert system for the design of paper handling paths.

This paper is not a literature review, since the body of work in the area of computational engineering synthesis is vast. This paper is, however, a look at a model for design synthesis automation, a foundation of the field, and an illustration of its effectiveness by mapping our own work onto the model. For those seeking a detailed exploration and literature review into areas and projects in this field, *Formal Engineering Design Synthesis* [5] and *Engineering Design Synthesis* [6] present in-depth investigations into the state-of-the-art of the field.

By formal synthesis we mean the algorithmic creation of designs; the organized, methodological modeling, implementation and execution of design creation on a computer. The goal is to leverage computational speed and depth of calculation to reduce the tedium for human designers and augment the process of searching the space of alternatives for a preferred solution. Synthesis as a method contrasts with traditional optimization in that the goal of synthesis is to more broadly capture, emulate, and/or utilize design decisions made by human designers to create. Ideally, computational design synthesis is invoked in situations in which the human designers are often at a loss of what avenues to pursue, or the best method of achieving a solution requires the generation and evaluation of countless alternatives.

The current supply of synthesis research is mostly produced in academia, yet in only a few cases have these met the demand of industrial design problems. The reason is that the problem is challenging, a complex balance between representation, generation, and search of a design space in pursuit of original design solutions. This paper presents the various aspects of computational design synthesis, in a formal model to elucidate the gap between this supply and demand.

Thus, a generic model of computational design synthesis is presented in which the method is divided into four major activities: representation, generation, evaluation, and guidance [7,8]. Any implemented system that automatically designs must include some semblance of these four steps. In a way, these activities are similar to some important activities that humans follow in their design process: creation of a mental model of the object (representation), creation of the parts and the whole (generation), analysis of how well it meets the design goals and constraints (evaluation), and feedback on improvements to the design for the next iteration (guidance).

In the following sections, we develop these four aspects of the synthesis process. They provide a framework for presenting research in this area, a framework for tackling new research problems in design synthesis and an approach to teaching the field to students. After discussing each area, we highlight a few research projects in the area based on the authors' work, some of which are moving into commercial application, and then discuss challenges to the research community that need to be tackled as the field further progresses.

2 The Foundations of Synthesis

Figure 1 presents a flowchart that highlights the division of tasks into four main steps: representation, generation, evaluation, and guidance. In this section, we discuss this simple flowchart and show that it is a generalization of numerous computational design synthesis methods.

Setting up a problem initially involves declaring constraints and constructing objective functions or design goals. At the top of the flowchart in Fig. 1, the design problem is formulated. The act of formulating or initializing a synthesis process has not received much attention in the literature, since most computational synthesis methods are developed to solve a particular design problem. Currently, there is motivation to generalize methods to handle a

[1]Corresponding author.

Contributed by the Engineering Simulation and Visualization Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received April 25, 2005; revised manuscript received July 8, 2005. Associate Editor: J. Shah.

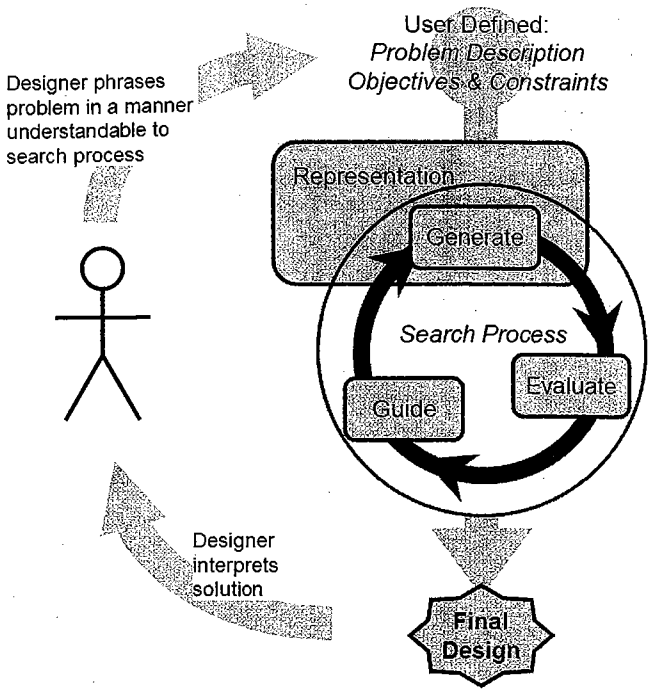

Fig. 1 The generic flowchart for the synthesis of open-ended engineering design problems

broad array of design problems since most computational synthesis methods require a user who is knowledgeable about the intricacies of the algorithm. Just as finite-element analysis has boomed since more accessible pre- and postprocessing have been developed, synthesis methods may follow a similar course. Indeed, recent development in optimization software packages, such as Engineous's iSight [9] and Multistat's Visual Optimizer [10] have added robustness and improved user interfaces in order to streamline the preprocessing and postprocessing of data so a wider range of engineers can easily apply optimization. While computational synthesis addresses more ambitious problems than traditional optimization, their popularity in both industry and academia would likely increase with well defined if not generic user-interfaces.

The *representation* is formulated by the developer of the computational design method to capture the forms or attributes of the design space. For example, in genetic algorithms, the representation is usually a bit-string that represents the key decision variables in the process. Candidate solutions are generated using this representation, in the *generation* task. In genetic algorithms, generation is done by mutating and crossing over existing or parent candidates. Each generated candidate is evaluated in the *evaluation* task to determine how well it meets the objectives and constraints of the design problem. Based on the objectives calculated for the candidates a *guidance* or feedback strategy is implemented to inform the search process to find better solutions in the subsequent iterations. In genetic algorithms, this is the "survival of the fittest" tournament selection where candidates with inferior fitness values are removed from the search process.

Most synthesis methods assume an ordered structure to the design space, whereby each instance within the space is a solution to a common design problem; this instance may be a fully realized design or a more abstract representation. Within such spaces, instances can be organized such that solutions with similar configurations are in close proximity. The principle for this visualization is that designs that require little modification to transform them from one state to another are closer to each other than designs that require larger modification. Therefore, to move about this space of solutions, one makes transformations to designs to arrive at neighboring solutions. Through numerous modifications, one can visit a wide variety of possible configurations. Because the space is infinite and includes past, present, and future design states, this searching through the space becomes analogous to creating, designing, or inventing in design problems. If this space is describable to a computational system, then the challenge is to effectively find in this set the solution that best meets the demands of the design problem.

2.1 Representation. The representation defines the level of detail and focus of the computational search process as well as dictating the range of candidates that can be created. The representation helps determine the appropriate generation or search mechanism and, similarly, the generation mechanism determines the appropriate representation. One could argue that representation is the key challenge in synthesis. Unlike analysis where one uses computational power to find important performance parameters, synthesis creates the details of an engineering model in which the simplicity or complexity within the range of candidate solutions is left to the designer's discretion. In observing the interactions of human designers, one can readily note the ease with which humans are able to develop and compare models of various complexities. Providing this ability to computational systems requires new approaches. Furthermore, this variability and open-endedness in representation makes research in computational synthesis methods more than simply applying optimization or heuristic search methods. In the following sections, we discuss several representation schemes.

Design is often viewed as a transformation from function to form, while the process of synthesis is the creation of a form that meets functional requirements. Most of this paper is devoted to various approaches to computational methods of synthesizing a design from requirements; however, we begin with a brief overview of the representation of function and form.

2.1.1 Function. During the early conceptual stages of design, many design methodologies require the designer to construct functional graphs of the to-be-designed artifact, for example, the function structures of Pahl and Beitz [11]. The function structures formalism represents functions in a block-diagram of energy, material, and information flows; that is the transformation of input to output flows. Block diagrams are useful for designers creating such structures by hand; however, advances in symbolic computation have facilitated symbolic representation of functions. Stone and Wood [12] have developed an extensive language of function to be used within function structures.

An example of a systems engineering approach to function modeling is the bond graph formalism [13,14]. The bond graph formalism represents a dynamic system as a composition of components, such as transformers, sources, and gyrators. Each component deals with power flow and has effort parameters (such as pressure, voltage, and force) and flow parameters (such as flow rate, current, and velocity) at its ports. This technique is an extension of generalized circuit theory and can deal with a variety of dynamic systems, including electric circuits, mechanical systems, and hydrodynamic systems. Bond graphs are a subset of the function definition of Pahl and Beitz, because they deal only with power transformation. Based on the bond graph formalism, Bracewell and Sharpe proposed a practical design platform called Schemebuilder [15].

While the transformational view of function does not require a concrete component or device to perform each transformation function, the qualitative physics community (e.g., de Kleer [16]) has developed Model-Based Reasoning (MBR) technologies based on transformation formalisms. MBR technology reasons about a device's behavior (that is, what a device does) from explicitly represented models of the device's component to the system's behavior and then eventually to function. Functional reasoning adds functional concepts into model-based reasoning technology. In contrast to MBR, functional reasoning deals with "what the device is for." Within qualitative physics, a number of

interesting techniques for synthesis have been developed, although, at least initially, their application domains have been limited to analytical tasks such as fault diagnosis and explanation. Synthesis based on functional reasoning focuses on creating a model of a design object. Examples of these efforts include the Causal Functional Representation Language, a formal representation scheme for the physical behaviors of a system [17]; the Structure-Behavior-Function (SBF) model, a system that represents function and has a design-case memory that represents each case as an SBF model for analogy-based design [18]; and the Function-Behavior-State (FBS) model and its computer-based implementation called the FBS modeler [19–21]. Within the FBS modeling, a function is an association between the designer's intention and a physical behavior of components that realizes the function and is represented in a "to do something" form. This formalization of function is more flexible than the transformational definition of function and allows definitions of function such as "to fix components." FBS has also been applied to more domains. The FBS Modeler has knowledge bases for function prototypes and physical phenomena based on Qualitative Process Theory [22]. The core of the FBS Modeler to support conceptual design synthesis is Qualitative Process Abduction System (QPAS) that performs the core of synthesis to derive structural information, behavioral, and functional descriptions [23].

2.1.2 Form. Most synthesis techniques include representations of the final form of the design object. The representation of form is both domain and solution technique specific. For example, one final representation of form might be simply topology nodes with no geometric detail [24], while another might be a complete specification of the geometry and material [25]. The scope of final representations is too broad to cover in this paper. Where appropriate, the representation of the final design object is covered within the description of the synthesis technique. The interaction between the function and geometric form of a design is a significant challenge to a computational synthesis system.

2.1.3 Vector-Based Optimization. Many engineering design problems have benefited from the application of optimization algorithms. Traditionally used after conceptual design has refined a design to a specific topology, optimization then provides an ideal way to determine what the sizes or parameters of various components should be. However, the synthesis process is not just about determining parameter values, but also about determining the topological structure itself. The selection of the variables and even the objective function and constraints can become dynamic, changing throughout the process.

When the mathematical formulation of the problem is nonconvex, discontinuous, or multimodal, or when the problem formulation is dynamic, nontraditional optimization methods are needed that can handle these situations. These typically include stochastic effects found in optimization methods like simulated annealing [40], genetic algorithms [43], and tabu search [53]. Thus, the encoding of a rich representation can often be accomplished through a fixed set of variables. As the structure of the design moves further from its encoding, the less successful the optimization is in finding optimal designs. The goal of a synthesis system at the topology level is to determine what the variables are, in other words, what the topology is.

2.1.4 Graph Structures. Graph structures are frequently used to represent the design space and the design solutions. A graph is a collection of *nodes* interconnected by *arcs*, which may also include parameters within the node and arc objects that are important to the design problem. Graphs can represent many different types of engineering systems such as electrical circuits [26], roadways [27], and chemical plants [28]. Approaches to constructing optimal graphs include starting with a single node and building up to a full design or starting with a complete graph and removing elements until an optimal structure is reached.

2.1.5 Shape and Graph Grammars. Another representation method is to store only the transitions or production rules for creating a solution, as opposed to storing the solutions themselves. These production, or grammar, rules can be based on shape (as is done in shape grammar formulation [29]) or on function [30–32]. Graph grammars are another approach in which the grammar describes the language of the structure of the topology, possibly at a more abstract level [33]. Such representations can produce a wider variety of candidates since solutions need not have common characteristics, but merely a common starting point. Furthermore, the rules may be developed to include important but elusive hard constraints that define the space of solutions without eliminating the variety of design possibilities.

Shape grammars have been extensively used in architecture [34–37] and engineering (see [38] for a review). In shape grammars the production of geometry or shape has three important characteristics: (1) the rules act directly on the geometry, (2) the shapes are parametric, and (3) emergent shapes can be recognized, supporting the possibility of creativity in the synthesis process.

2.2 Generation. Generation methods can be as naïve as a random number generator or as sophisticated as collaborative agents simulating human thought as a design evolves. The generation process may occur in one iteration or, more likely, over many. Given the challenging task of implementing a process to emulate human creativity or even to reason about sophisticated design configurations, one might tend to prefer naive generate and test methods over knowledge-based methods, particularly given the speed of modern computers. However, if evaluation methods are expensive, one might opt for a generation method somewhere between naive and knowledge-based. Below, we discuss a few of these approaches.

2.2.1 Optimization. Many generation methods use optimization techniques, which range from sequential quadratic programming [39] with its analytical approach to finding new search directions to simulated annealing [40] which uses random perturbations.

Many traditional optimization methods are based on a twofold generation approach, first identifying a productive search direction, and then performing a one-dimensional search along that direction to identify a summit. From this summit, the process again identifies the best local search direction and proceeds. Such approaches are deterministic and efficient on smooth uni-modal spaces; they are also useful when achieving a local optimum is all that is required.

Direct search methods do not require gradients [41,42]. These approaches leap to neighboring states along directions that appear productive in terms of recently visited states. Direct search methods often generate new candidates deterministically as do gradient-based methods. However introducing some randomness in finding neighboring solutions prevents becoming trapped in local neighborhood. Simulated annealing along genetic algorithms [43,44], for example, use direct search in that they do not require gradient information, but depend on stochastic movements to avoid local optima using little domain or search knowledge in the process.

2.2.2 Search Trees. One view of design is that it is a sequence of decisions in which each subsequent decision depends on previous decisions. This sequence can be represented as a search tree as shown in Fig. 2, in which each node is an incomplete solution that must undergo further development until a final design is created. The phrase "back to the drawing board" evokes this process of moving back up the tree to earlier design decisions in order to progress toward a final design by making different decisions. Backtracking often involves repairing prior solutions because constraints and variables often change as the design process progresses.

Various tree traversal methods such as depth-first search or A* are useful in navigating the tree. However, one may not be able to

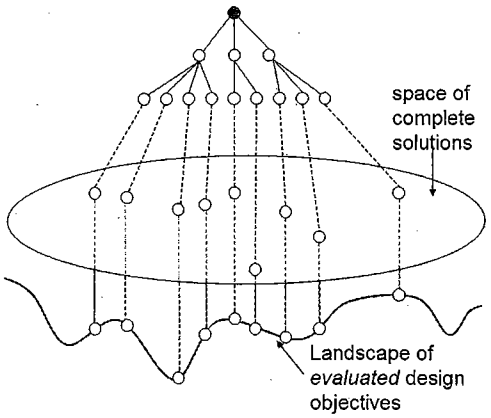

Fig. 2 Generation of candidates sometimes requires a sequence of operations to arrive at a complete design

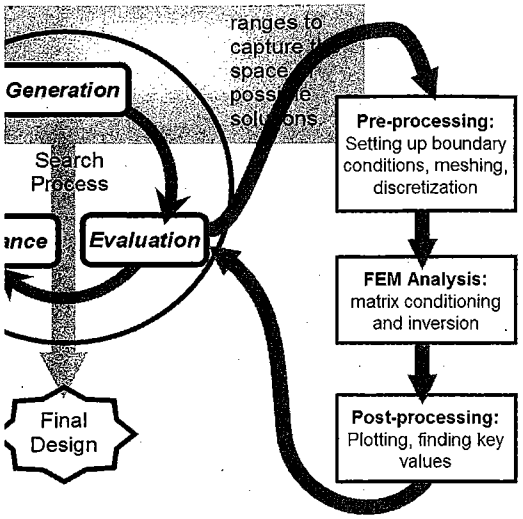

Fig. 3 A flowchart combining simulations tools with computational design synthesis

evaluate the worth of a single design decision until a final design is created. Knowledge-based approaches such as case-based reasoning may be useful to guide the search to successful designs. Completed designs at the leaves of the search tree can be evaluated based on the performance parameters initially defined by the user. As in optimization, one can visualize a projection of the candidates and their performance values. However, the concept of neighboring states is complicated since the process must undo previous decisions and make new ones in order to arrive at the neighboring states.

2.2.3 Agents. The concept of using design agents to substitute for the human ability to make decisions during synthesis has also been explored by several researchers [45–47]. Collaborating agents, each representing a different human expertise or preferences, have been used to synthesize new designs. The collaboration may be ordered (i.e., deterministic), random, or stochastic with agents being invoked as a result of probabilities determined from past successes.

2.3 Evaluation. As is shown in Fig. 1, the third task, evaluation, involves measuring the worth or potential success of a candidate. Although the generation of design concepts alone may provide a benefit to the designer, any automated synthesis system must include an appropriate level of design evaluation to provide feedback to guide the generation process. For some design problems, the evaluation may be a simple analytical expression; however in most design problems, the evaluation often involves simulation of the solution. Finite element methods, computational fluid dynamics, circuit simulation, and other computational analysis tools offer accurate and robust evaluation analyses. Figure 3, which is a detail of Fig. 1, shows that evaluation includes three distinct steps if external simulations are used. Some common difficulties in combining simulation tools within computational synthesis include: (1) since search processes often rely on testing thousands of candidates, the time required to analyze each solution must be kept to a minimum; for example, a finite element simulation that takes an hour could cause the search process to take thousands of hours; (2) in order to perform a simulation, detailed preprocessing often must be done for each simulation; this can involve the time-consuming process of setting up the proper boundary conditions and the resolution of the discretization (e.g., meshing of elements, or time steps) which is often performed by an engineer with experience and knowledge; (3) the postprocessing of the simulation data is complicated by the fact that a single metric of design worth is required and solutions may have defects that can only be found by considering all the data; (4) simulations sometimes fail due to problems in calculations, such as the singularity of a matrix or unbounded iteration. These failures can cause the entire search process to halt prematurely if not handled properly; and (5) engineering design problems often contain more than one measure of design worth, requiring the balancing of multiple objectives; multiobjective optimization continues to be a significant hurdle in computational design synthesis.

These challenges have motivated research in merging computational synthesis methods with computational analysis methods. Hybrid methods might be able to handle invoking and interpreting the simulations required for evaluating candidate solutions. In past synthesis methods [48,49], significant development has been required to create a robust *evaluator* that negotiates these difficulties.

One practical aspect of evaluation is designing search techniques that minimize the number of evaluation calls. Recent advances in computational speed and memory have made this goal less important in some instances, but with the exception of deterministic, one-time generation algorithms, most generation algorithms require multiple iterations, at times on the order of tens of thousands, making computational speed a significant concern. Complex or lengthy simulation analyses can make such design runs impractical. The most effective approach, then, is to develop quick evaluation heuristics for the early stages of the search process to steer the search to productive areas of the design space. As the algorithm progresses, or for the final evaluation, more complex simulations can be used [50,51].

2.4 Guidance. The final task in the synthesis process is to provide feedback to the system. Based on objective function values determined in evaluation, the goal is to find an approach to designing or generating improved solutions. Two approaches are real time iteration and long-term strategy. For real time iteration, the algorithm provides direction by dynamically analyzing the results of the evaluation and directing the algorithm toward improved designs for subsequent iterations. Examples of guidance techniques include greedy search (select the solution that is believed to be closest to the goal), Metropolis criteria (the stochastic approach adopted in simulated annealing), or any type of genetic algorithm selection technique (elitism, roulette wheel approach, etc.) Machine learning techniques are also potential guidance strategies [52]. Guidance is akin to learning from experience. The basis of Tabu search [53] is a guidance strategy that builds on past design experience; this search technique is used in the A-design method discussed below.

Another approach is to learn from the execution of the algorithm so that in subsequent design tasks the system can be more effective and efficient. Learning may be problem or domain specific, or it may focus on the design process itself. While con-

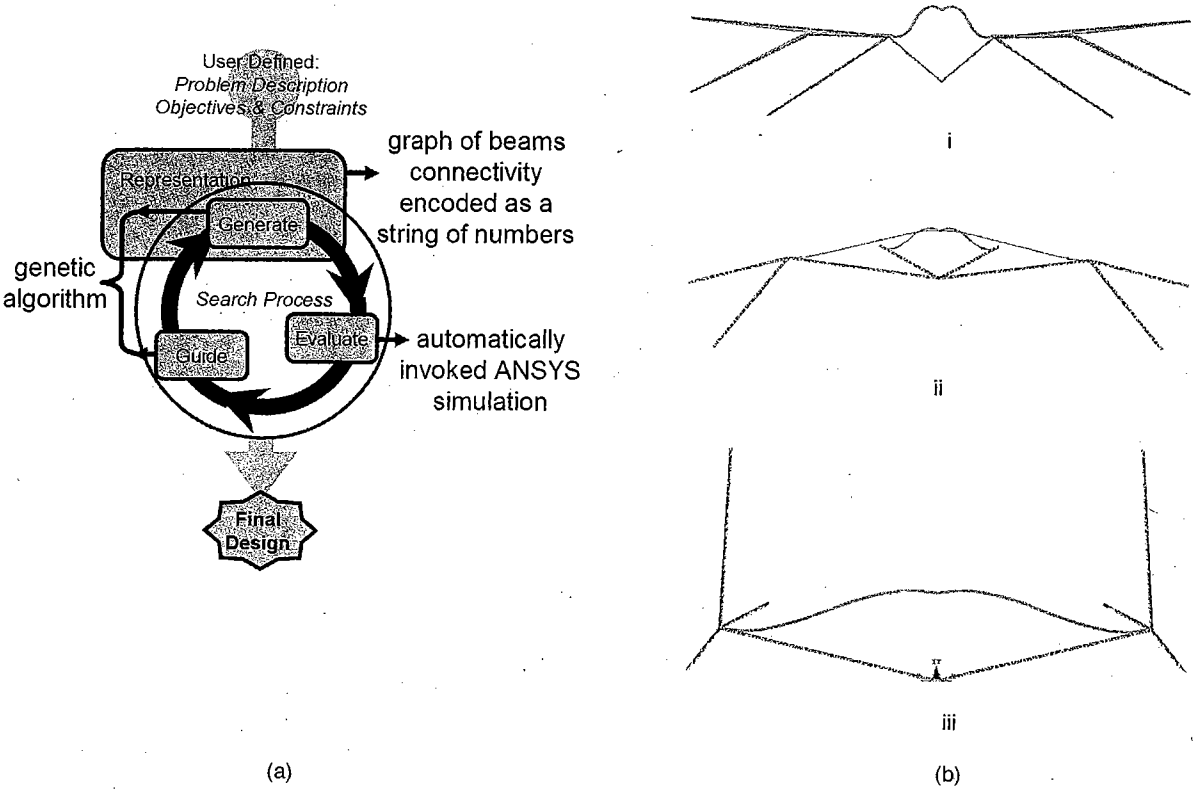

Fig. 4 *(a)* **The flowchart presented in Sec. 2 can be used to describe the synthesis method of bistable microswitches; (*b*) three example solutions that have two stable positions (one shown in black dashed lines and one in solid grey lines) that are 20 μm apart**

strained by the representation, the guidance strategy is pivotal to the effectiveness and efficiency of the synthesis method. Concepts can be borrowed from other approaches to improve a computational synthesis process. Guidance is closely tied to generation. In the optimization approach to generation, the two are often inseparable.

3 Applications

An extensive body of work exists on formal design synthesis. The books *Formal Engineering Design Synthesis* [5] and *Engineering Design Synthesis* [6] present surveys of the field and detailed investigations into projects, theories, and visions for the field. Recent highlights include: research on automatically creating circuits for both passive and active applications [54]; Sims [48] and White et al. [55] evolve animal-like robots configured from actuators and limbs to create walking, jumping, and swimming motions. Peysakhov and Regli [56] build Lego structures that meet given spatial constraints. Shea, et al. [57] construct tensegrity structures composed of interconnected beams and cables. Even the design of wireless internet access points has been automated through invoking a genetic programming approach to positioning and interconnecting hubs [58].

The following subsections present examples from our own work that demonstrate the application of the automated design synthesis model presented in Sec. 2. Many of these applications have been transferred outside academia to industrial use or commercialization.

3.1 Design Synthesis of Multistable Compliant Structures. The design of compliant bistable microswitches presents a difficult challenge in structural and topological optimization [59]. Not only must one design a structure that is compliant within the range of desired displacements, but one must also simultaneously design the structure so that it can maintain new positions with no power or force input. Such devices are useful in high-power integrated circuitry in which discrete switches are necessary to handle high currents. Since the shape of such structures is difficult to conceive, an automation tool is developed to synthesize these structures.

Figure 4(*a*) shows the generation and guidance tasks accomplished by a genetic algorithm (MATLAB's Genetic Algorithm Optimization Toolbox [60]), which is a popular approach used in numerous synthesis methods due in part to its ability to find solutions in highly constrained search spaces. While the most natural representation for the design space would be a family of graphs with nodes representing intersections and edges representing beams, the implemented solution conforms to the needs of the genetic algorithm by representing alternatives as a list of real numbers. Within this list, the dimensions for each beam (length, width, and angle) are interspersed with "keypoint" values that assign the ends of each beam to the particular intersections that develop at the connections of other beams. While the design representation is limited to a vector of fixed length, the keypoints, allows one to create a variety of topologies including all beams connected in series, all beams connected in parallel, or any combination in between. Figure 4(*b*) shows designs synthesized automatically which meet the design goals of having two stable positions 20 microns (10^{-6} m) apart and an actuation force of 1 μN. In these figures the black dashed lines represent the undeformed skeleton of the device and the gray solid lines represent the structure in the second stable position, which is always 20 microns directly above the first. The unintuitive nature of designing such bistable devices on the microscale provides an excellent testbed for computational synthesis [61].

3.2 A-Design: Agent-Based Design. A-design is an agent-based computational design methodology that merges two generation methods, genetic algorithms with knowledge-based search, to synthesize electro-mechanical configurations based on functional

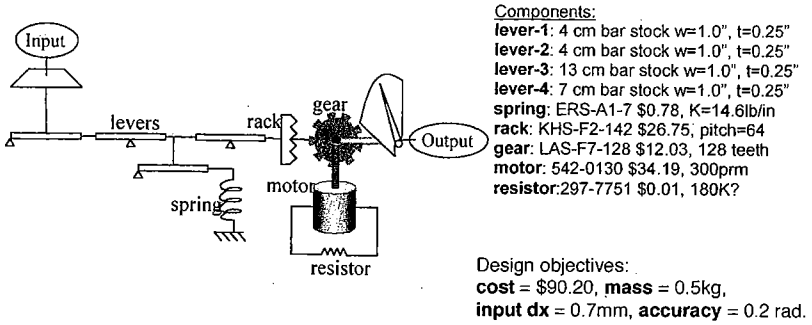

Fig. 5 Weighing machine designed by the A-design (from [46])

reasoning [46,62]. A population of designs is created by agents, with each formation of a population considered a generation. At each generation, designs are evaluated and sorted. Inferior designs are pruned away. The remaining designs, including those forming a Pareto optimum, are the basis from which new solutions are formed and compared, thus providing guidance to the method. The use of populations differs from traditional genetic algorithms in that modifications of the designs are explicitly goal-directed and are heavily influenced by deterministic knowledge about the problem domain. The better designs are fragmented and rebuilt, forming the basis for the next population. The process continues until it converges. It uses recursive information that allows for rapid and flexible design modification in response to changes in design evaluation criteria. The initial work uses a function structure/bond graph representation to build agents.

In the A-design methodology, configuration agents, defined in the work as knowledge-based strategies, combine to produce design solutions in a collaborative, stochastic manner. Additionally, the population of design agents changes over time via a manager-agent that modifies the probabilistic weightings for choosing agent types based on their relative success in producing good designs and on user preferences, which can change over time. Figure 5 shows a weighing machine generated by A-design using its functional representation and a catalog of parts. The system has also been applied to the optimal design of bulk manufacturing processes [63], extending the technique for design optimization applications.

Olson and Cagan [47] extend the A-design approach to model cooperative agents. Unlike the initial approach in which agents collaborate independently, each making a change without regard to preferences of the other agents, this extension frames each member agent within a cooperative team context. That is, each agent acts under the assumption that associated member agents share the same mutual goals, have similar capacities, and intend to follow the designated team processes. Under these assumptions, each agent evaluates and makes suggestions for the best design move according to its particular view, but final design decisions are always made in light of group suggestions and consensus. The collaborative team searches the space more extensively but also more efficiently, generating significantly improved quality in the same run time as a concurrent but noncooperative algorithm. The difference in synthesis approach is a more sophisticated generation and guidance method.

The system has been extended to incorporate new learning and memory capabilities based on cognitive principles [64]. In particular, this work is based on the finding that experts in a domain have a large set of commonly co-occurring linked elements or "chunks." A design chunk is a group of interconnected components that appear in multiple designs. By learning design chunks from a problem it has solved, the A-design system can apply these chunks to new problems. The processes incorporated into the system are not complex and are the simplest processes that still demonstrate chunking based on findings from studies of human expertise. The system can now solve a design problem better and faster using design chunks than without. The system also demonstrates a limited transfer of knowledge across problems when it has learned design chunks from one problem and applied them to another related problem.

3.3 Layout. Determining placement locations for components within a product housing or container is one synthesis problem that has received much attention and success. The placement must adhere to an extensive list of required constraints while optimizing for specified objectives. The layout problem is difficult and time intensive making an automated tool capable of solving such problems extremely valuable. For example, deciding where to place components on the board consumes 50% of the time required to design a printed circuit board (PCB). Currently the PCB layout process is manual, often requiring six weeks to complete a design. There have been no commercially available tools to automate the placement of components on PCBs, in engine compartments of cars, or within aircraft, among other applications. Mathematically the problem is difficult, with multiple local optima and discontinuities in the space, complicated by the high number of components, constraints, and multiple objectives.

Szykman and Cagan [65] formally defined 3D component layout as follows: "Given a set of three-dimensional objects of arbitrary geometry and an available space (possibly the space of a container), find a placement for the objects within the space that achieves the design objectives, such that none of the objects interfere (i.e., occupy the same space), while satisfying optional spatial and performance constraints on the objects."

The nonlinear, multimodal, and discontinuous nature of the typical layout space makes finding a solution challenging. A survey of approaches to solving this problem can be found in [66]. Many of the generation methods have been applied to the 3D layout problem are optimization methods with the exception of a heuristic-based bin packing algorithm [67]. Gradient-based algorithms, which settle in a local minimum [68,69] produce inferior results to stochastic algorithms which inevitably require more computational time. Several stochastic methods include: genetic algorithms (GAs) [70,71] and simulated annealing (SA) algorithms [72–75]. These approaches are effective, once the coding of GAs is determined, but require unacceptably long run times for realistic 3D problems. Hybrid approaches have also been considered with unsatisfactory results [76].

One of the most effective methods to solving the general layout problem uses a stochastic version of the pattern search algorithm called Extended Pattern Search (EPS) [77]. Pattern search algorithms use patterns, which are search directions with varying step sizes, to explore a search space in a greedy manner [78,79]. In 3D component layout, the patterns are the translations and rotations of components and step sizes are the amounts of translations and rotations. The search space is explored starting with large step sizes of the patterns. When patterns with a particular step size no longer improve the solution, the step size is decreased by a common multiplicative factor. Extensions are introduced to the basic pattern search method to help converge to good solutions and

make the algorithm efficient [77]. Alternative patterns have also been explored [80]. Extensions have further refined the EPS algorithm to take into account the relationship between pattern size and component geometry [81,82]. The basic representation as an optimization problem uses an objective function-based formulation by first optimizing the weighted sum of component density while minimizing component overlap. An octree representation [83,84] is used for fast intersection evaluation between components of complex shapes. Evaluation takes place first through geometric analysis based on the octree calculations, with additional performance evaluation added through simulation or heuristics. Guidance occurs through the pattern directions coupled with reduction in step size at the appropriate points in the algorithm's execution.

The EPS technology has been applied to the design of automatic transmissions, the layout of a truck chassis, the packing of trunks to meet SAE specifications, the layout of engine compartments, and the layout of parts to be made within SLA machines. Figure 6 shows an engine compartment placement, a trunk packing, and an SLA packing each automatically laid out with this technology.

The Extended Pattern Search technology for product layout has matured to the point of being commercialized by DesignAdvance™ Systems, Inc., a spin off from Carnegie Mellon University. DesignAdvance has extended the basic technology to layout 2D Printed Circuit Boards (PCBs), its first commercial product, and is in the process of developing a commercial version of EPS for general 3D application to mechanical and electromechanical products.

3.4 Shape Grammars, Interpreters, and the Study of Brand. Shape grammars were introduced by Stiny and Gips [35] in the architecture literature. Shape grammars are a production system of shape or geometry with the properties of parametrics (one shape can represent an infinite number of them) and emergence (you may get out more than you put in). Initial exploration of shape grammars by Stiny focused on describing and recreating architectural styles including Chinese Lattice designs [33], Palladio-style villas [34], and Mughul gardens [87]. Several other applications followed, most notably recreating the prairie homes of Frank Lloyd Wright [36].

Beyond architecture, shape grammars have applications in engineering and industrial design. For example, Agarwal and Cagan [88] introduced the coffeemaker grammar; Brown et al. [89], the lathe grammar; Shea and Cagan [90]), the truss grammar; Agarwal, et al. [91], the MEMS resonator grammar, McCormack and Cagan [92], the inner hood panel grammar; Pugliese and Cagan [93], the Harley motorcycle grammar; and McCormack et al. [94], the Buick Grammar. The focus on the last two references was on the representation of product brand. The thesis is that classes of products could be broken into discrete attributes that could then be modeled in a shape grammar. The advantage is not only articulating the grammar but enabling a tool to guarantee that products are generated that meet a brand identity. In the Harley grammar, brand was captured through constraints applied to a grammar that generated motorcycles while in the Buick grammar, the essence of Buick is captured within the shapes that define the rules of the grammar themselves (see Fig. 7).

Shape grammars are a representation of design descriptors that can be acted on by a generative system. For example, an optimization-based generative scheme can be used to search the space of design options described by shape grammars. In the shape annealing method, Shea et al. [95] use a simulated annealing algorithm with a shape grammar to design optimal truss and frame structure configurations (see Fig. 8). Evaluation is done with a finite element method, made possible because the translation of design topology to finite element model was relatively straightforward. Guidance was provided through Hustin move sets within the simulated annealing algorithm to encourage larger moves initially and smaller ones later in the run [96]. Shea and

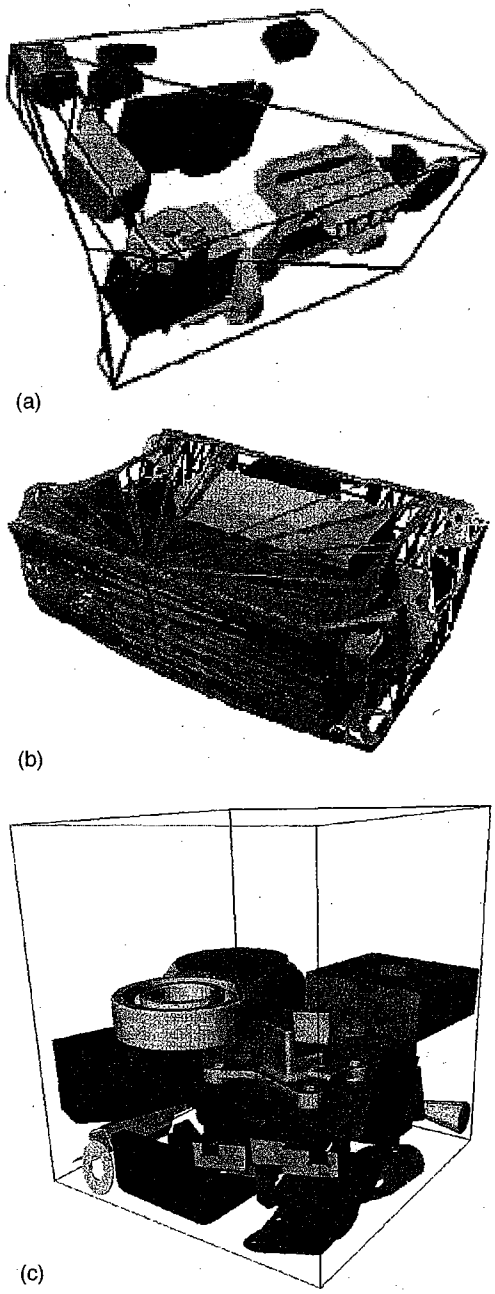

Fig. 6 Layout configurations from Extended Pattern Search (*a*) from [77]; (*b*) from [85]; (*c*) from [86])

Smith [97] applied the technique to the design of transmission towers. Shea and Zhao [98] developed an industrial version of the method called eifForm and used the method to develop a cantilever structure called the "Paternoster Square Noon Mark," built and permanently installed in London on the London Stock Exchange building in Paternoster Square.

General implementation of shape grammars requires a system that can interpret shapes, seeking new forms that emerge out of a configuration of lines (straight or curved). McCormack and Cagan [99] introduced a framework that provides a means to implement and use shape grammar technologies in design practice via the a shape grammar interpreter. A user working with the grammar through an interactive loop chooses which of the applicable rules to apply and where to apply it. Alternatively, an optimization routine can make the choices presented by the interpreter concerning rule selection, application location, and parameters during instantiation. After each rule application, the design is examined and if

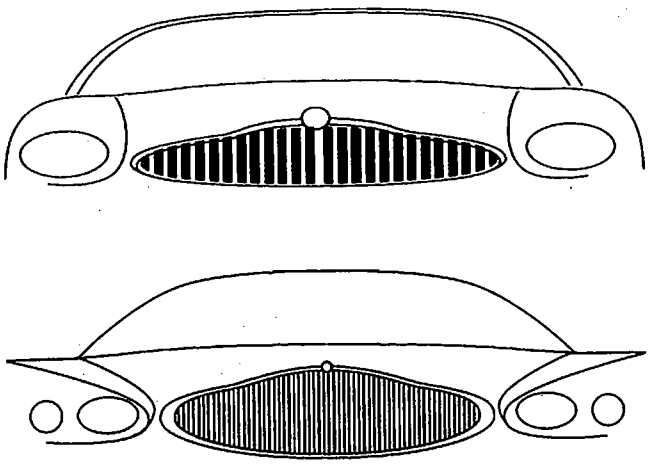

Fig. 7 Novel Buicks generated from shape grammars (from McCormack et al. [94])

the design is deemed completed, it is presented to the user, or, if not, the design is modified by the user or the optimization routine.

4 Discussion

The model of the synthesis process consists of four steps: representation, generation, evaluation, and guidance. Any research endeavor that focuses on the synthesis process should, whether explicitly or implicitly, address these four steps. In solving new problems, there are a number of issues to consider. Below is a list of questions that one should consider in tackling new problems.

4.1 Representation.

4.1.1 What are the Basic Building Blocks in the Design Problem? The first step in the representation of design is to determine the list of building blocks from which the design (feasible or infeasible) can be created. It is to be noted that generally with the increase in the number of basic building blocks the design solution space expands exponentially. On the other hand lowering the number of basic building blocks leads to lesser exploration of design space.

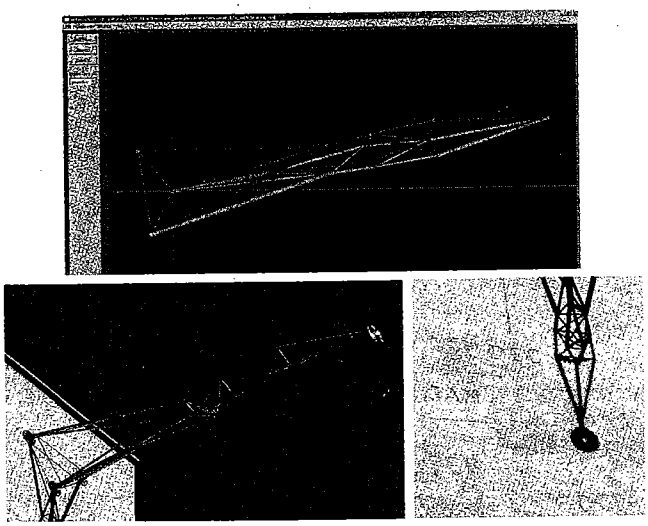

Fig. 8 "Paternoster Square Noon Mark" truss structure designed with shape annealing (eifForm) by Shea and Zhao [98]. Model and actual structure are shown.

4.1.2 How are Interrelations Between Basic Building Blocks Going to be Handled? Constructing designs from fundamental elements becomes complicated when the fundamental elements are not all of the same type. Having different building blocks gives rise to differing interrelations between these blocks and how they affect the entire design. A method for defining the way building blocks interact together is required.

4.1.3 Can You Live With a Representation That Leaves Out a Large Set of Feasible Solutions? or (Can You Live With a Representation That Includes Numerous Infeasible Solutions?) Constructing the rules of a representation rarely perfectly delineates the search space into feasible and infeasible solutions. If a clear penalty can be assigned to infeasible designs then perhaps allowing these to exist is acceptable since it allows for much complete search. However, if much of the search is wasted on infeasible designs, one might consider a representation that is more focused at the expense of overlooking a portion of feasible design.

4.1.4 Is it Possible to Capture Different Technological Solutions to the Same Subproblem? In engineering, one often simplifies a phenomenon to a simpler model both for analysis and design purposes. However, since diverse approaches to solving a design problem should be examined, it would be ideal to construct a representation that can test different approaches within the same search. For example, if one were to create a heating system (i.e., oven) for a specific application, it would be nice for the computational design synthesis to include radiative, convective and microwave approaches within the same search.

4.2 Generation.

4.2.1 Can a Completely Random Decision be Made for Each Design Step? If each decision in the generation process can be made randomly, chances of finding "creative" solutions increases. However, complete randomness also brings with itself the curse of time-consuming search as the design space explored becomes vast. A careful tradeoff needs to be made to balance the exploration versus time-consumption.

4.2.2 Should Design Constraints be Hard vs Soft? A careful generation technique can easily take into account some of the design constraints without affecting the design generation time. However, the decision to introduce the constraints within generation needs to be made judiciously because in some cases constraints may prevent a thorough search of the design spaces. Including constraints in generation makes them hard constraints since the search will not explore candidate which it is prevented from generating. Another effective method of handling constraints in design is to use "penalty functions" within the evaluation stage. These are deemed soft constraints since the search process is allowed to explore these regions despite their infeasibility. One must be judicious in selecting which constraints are to be handled in the generation stage and which are to be handled trough the use of penalty functions. Hard constraints tend to reduce computing time but also reduce the effectiveness of an algorithm.

4.3 Evaluation.

4.3.1 Are There Multiple Objectives to Handle? The nature of objectives being evaluated during the evaluation stage also affects the selection of appropriate techniques for optimization. Most of the algorithms/techniques work for a single objective. Multiobjective problems are handled mainly in two ways, the first is to convert the multiobjective into a single objective (for example through use of weighting method), the second is to consider the "Pareto-optimality" approach. Conversion of multiobjective into single objective enables the use of most of the techniques, however such a conversion is highly debated in "Pareto-optimal" approach literature. Only special types of algorithms like Multi-Objective Genetic Algorithms, A-Design, A-Teams, etc. are able to take into account the "Pareto-optimality" concept. Hence, it