A Generalization of Computational Synthesis Methods in Engineering Design

Matthew I. Campbell and Rahul Rai

Department of Mechanical Engineering
The University of Texas at Austin
1 University Station, C2200
Austin, TX 78712-0292

Abstract

This paper presents a systematic way of thinking about computational synthesis methods in engineering design. By studying how various researchers present their work and through personal experiences, the authors have found that by dividing these problems into representation, generation, evaluation, and guidance challenges, one can more easily tackle difficult problems. This paper presents this division along with a brief literature review of papers that either implicitly or explicitly coheres to this division.

1 Introduction

Many engineering design problems have benefited from the application of optimization algorithms. After conceptual design has refined a design to a specific topology, optimization can provide an ideal way to determine what the sizes or parameters of various components should be. In the basic optimization model,

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{subject to:} \\ &\quad \mathbf{g}(\mathbf{x}) < 0 \\ &\quad \mathbf{h}(\mathbf{x}) = 0 \end{aligned} \qquad (1)$$

a breadth of parametric problems can be solved. In this model, f represents a predefined objective function that is based on a fixed number of decision variables arranged in a vector, **x**. The vectors **g** and **h** are inequality and equality constraints respectively that indicate violations within the variables in **x**.

One of the main difficulties is when the vector, **x**, is not clearly defined as a fixed set of known decision variables. For more than two decades, engineers have noted this shortcoming and have looked to other computational techniques in solving these more open-ended design problems. Applications of genetic algorithms, function and shape grammars, expert systems, case-based reasoning have proven to be successful in engineering as well.

As many readers will note that being a researcher in this field can be very rewarding. One is allowed to design ways in which the computer can aid in our design tasks. Given the numerous approaches to addressing design problems in this area, it is possible that there are underlying commonalities that need to be drawn out from our combined consciousness so that future endeavors in this area can build more readily on these past accomplishments. A similar presentation of the commonalities in this area of research is the ten steps in applying evolutionary computation as presented by Bentley and O'Reilly (2001).

In the following section, we will present a division of computational design synthesis (CDS) techniques into four main challenges: representation, generation, evaluation, and guidance. We believe that this division can be useful for three distinct purposes. Firstly, the division could provide a clear framework for presenting research in this area. Many times a reader of a paper in this area can be left feeling that a particular project was presented insufficiently. This is mainly due to the fact that such research often involves the production of new programming code that cannot be presented in a research paper in a concise way. Thus, it is nearly impossible to present a body of research while addressing all the concerns a reader might have about intricacies of the implemented method. The questions in Section 3 present a checklist for a writer that would allow them to better anticipate a reader's concerns.

Secondly, the division is useful in tackling new problems. As mechanical engineers, the authors find many traditional problems and problem-solving techniques in engineering could benefit from a computational design synthesis method. However, getting started on *designing* the computational system to design new solutions for a given engineering problem can be a daunting task. This division facilitates an approach to tackling new design problems. The questions in Section 3 can aid the researcher in approaching new problems in a consistent manner.

Thirdly, this division is useful in teaching this exciting field of study to students new to the idea of computational design synthesis. The division helps students see the benefits amongst different approaches, for example using simulated annealing versus genetic algorithms.

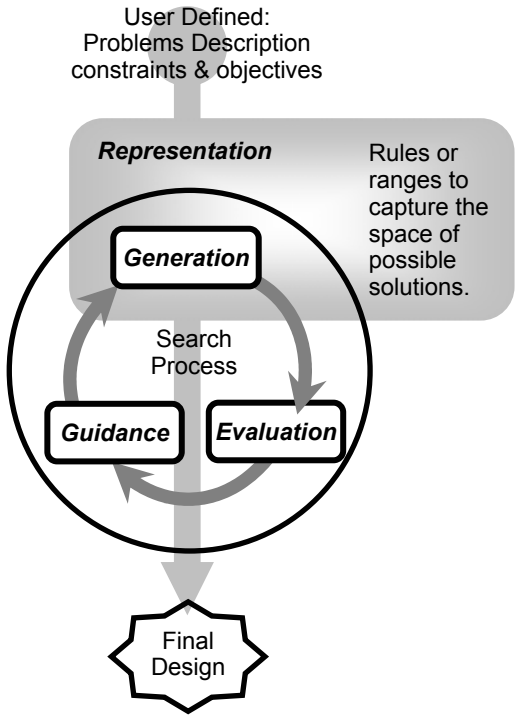

Figure 1: The generic flowchart for the synthesis of open-ended engineering design problems.

2 Method

Figure 1 presents a flowchart that highlights the division of tasks into four main areas: representation, generation, evaluation, guidance. In this section, we will describe how this simple flowchart is a generalization of numerous computational design synthesis methods such as traditional optimization, tree searching, or machine learning (Mitchell, 1997).

Initially, setting up a problem involves declaring constraints and constructing objective functions. At the top of the flowchart in Figure 1, the design problem is formulated. From a traditional optimization standpoint, this involves casting the problem in the *negative null form* shown in equation 1. However, in more ambitious problems, one is confronted with a description of the problem that might not clearly fit in this division.

The *representation* is formulated by the programmer 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. Upon this *representation*, candidate solutions are generated in the *generation* task. In genetic algorithms, this is done by mutating and "crossing over" existing or parent candidates. From the generated candidates, each one 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* strategy is implemented to inform the search process of how to find better solutions in the subsequent iterations. In genetic algorithms, this is the "survival of the fittest" tournament selection where candidate with inferior fitness values are removed from the search process. This example shows how the representation can be mapped into genetic algorithms, but it is believed that nearly every computational design synthesis method has at least these four parts.

It is generally accepted that *designing* as performed by humans is similar to *searching* as performed by computers. We tend to envision a space of design instances whereby each state within the space is a solution to a common design problem. Within such spaces, instances can be organized such that solutions with similar configurations are close in 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 in its organization and includes past, present and future design states, this "searching" through the space becomes analogous to "creating," "designing," or "inventing" in real 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. Computational search can happen at much greater speeds than human search process can as is evidenced by the Kasparov vs. Deep Blue chess match (Campbell, 1999). However, the computer has yet to match human abilities in *representing* design concepts, *generating* good solutions, *evaluating* the worth of such solutions, and using information from failed attempts to *guide* the creation of new designs. These activities are the distinguishing traits of the experienced engineering designer. In this section, we discuss the details and difficulties of implementing these activities computationally.

2.1 Representation

The representation provides a foundation that dictates the range of candidates that can be created. One can choose to represent a problem as a fixed set of decision variables (as is done in traditional optimization), a bit string genome as is done in traditional genetic algorithms (Holland, 1992), a string of executable code as in genetic programming (Koza, et al., 1996), or as grammar or production rules. This last type introduces an interesting distinction for representations. Earlier techniques represent the characteristics manifest in the complete candidate, whereas grammar rules represent the sequence of operations that describe the candidate. Grammar or production rule systems are developed by researchers to encapsulate a set a valid operations that can occur in the development of a candidate. These rules can be based on shape (as is done in shape grammar formulation; Stiny, 1980) or on function

(Soman and Campbell, 2002; Schmidt and Cagan, 1998). Such representations can produce a wider variety of candidates since solutions need not have common characteristics, but merely a common starting point.

2.2 Generation

Based on a representation, one must develop a method for generating new alternatives. Often, the generation task is the simplest to implement as computer speed is leveraged such that even randomly generated solutions can lead to improved solutions. However, depending on the complexity of the design problem, generation might happen all at once, or through a series of steps. Often, a candidate represented as a point in the search space can be perturbed to create neighboring states, thus making the generation simply the set of operations that transport one about the design space. In Figure 2, a more complex view of generation is presented. Here, a generation tree is traversed from an initial point to create valid solutions. Such approaches require the generation to encapsulate tree searching algorithms like depth-first search or A*. Furthermore, this approach requires there to be a metric (heuristic) for determining the distance to the complete design as well as design objectives that measure the quality of a design (as is discussed in the next section on evaluation).

Generation techniques can be any type of decision-making algorithm that operates upon the prescribed representation. Such approaches could include case-based reasoning functions or agents as the "builders" of design solutions (see example in Campbell et al., 2000).

2.3 Evaluation

The third task, evaluation, introduces a number of complications. In ambitious design problems, the evaluation is often not as simple as a fixed analytical expression (as is usually the case in defining f(x) in equation 1). Coincidentally, the related field of computational analysis offers accurate and robust approaches to determine a solution's worth. Approaches such as finite-element analysis, computational fluid dynamics, and circuit dynamic simulations could be incorporated within computational synthesis methods. In Figure 3, we present a flowchart for how such simulation tools could be incorporated with the evaluation stage. Unfortunately, there are a number of issues that currently complicate the merging of these two domains of computational design and computational analysis: time constraints, setting up proper input files, parsing output files, handling simulation errors, etc. In addition to simulation tools, evaluation can also include multiobjective design problems, predator-prey models, and fixed objective functions.

Despite the difficulties, evaluation is easily separable from representation unlike generation and guidance. The theoretical challenge is simply to translate or decode the representation into a form that is compatible with the evaluation method.

2.4 Guidance

Based on objective values determined in evaluation, the final task is to strategize an approach to redesigning or regenerating new and better solutions. Examples of guidance technique 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). In fact, one can view a variety of machine learning techniques as potential guidance strategies. Guidance is akin to our own "learning from experiences." The basis of Tabu search (Glover, 1989) is an effective guidance

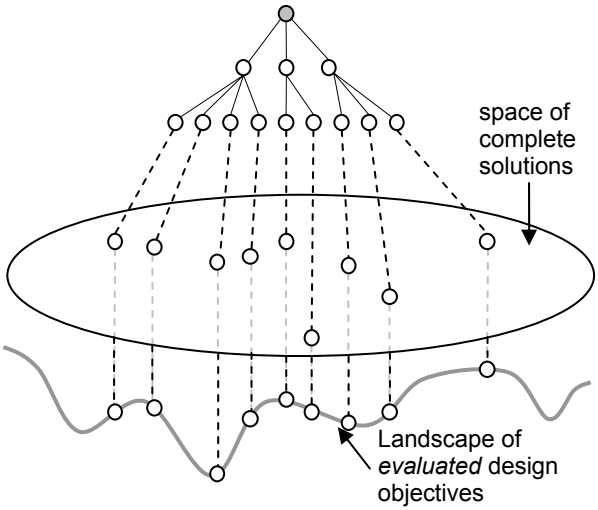

Figure 2: Generation of candidates sometimes requires a sequence of operations to arrive at a complete design.

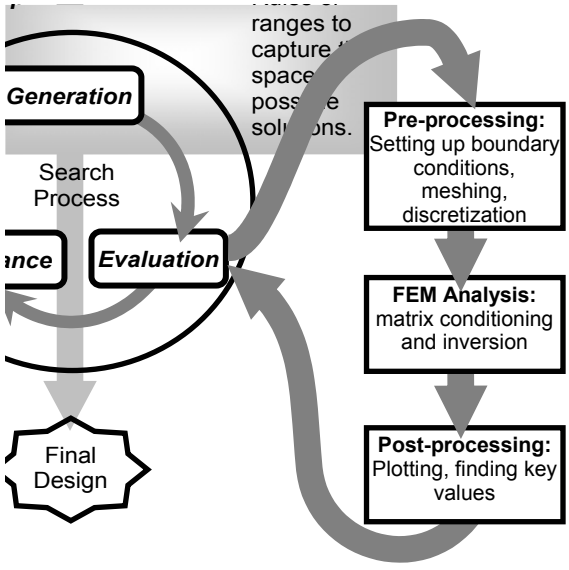

Figure 3: A flowchart combining simulation tools with computational design synthesis.

strategy that builds upon past design experience. While somewhat constrained by the representation, the guidance strategy is pivotal to the effectiveness and efficiency of the synthesis method. One can often borrow concepts from various different approaches to improve their automated design process.

3 Questions to ask when tackling new design problems.

As mentioned in the introduction, this division can be useful in tackling new problems. In this section, we present a checklist of questions that one should consider in tackling new problems.

3.1 Representation

What are the basic building blocks in the design problem?
The first step in the representation 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.
Furthermore, with the same problem, the basic building blocks can be constructed in different ways (Bentley and Kumar, 1999). The type of basic building blocks used for design generation inherently affects the complexity and variation in the candidate designs which is in turn reflected in the effectiveness and time of the search process. Of the four areas, representation is the most influential in directing the design quality; therefore, the utmost decisions in selecting the basic building blocks must be done with the utmost care.

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. In Campbell (2000), individual mechanical components (e.g. gears, springs, shafts) are modeled as the building blocks of the design. Because of the different attributes of these elements, the rules governing their connections are quite complex.

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. However, if much of the search is wasted on infeasible designs, one might consider a representation that is more focused at the risk of overlooking a portion of feasible design.

Can a fixed set of variables be used to represent candidates?
The set of variables in the design also affects the representation process. Generally, it is easier to represent a solution if the set of variables is fixed. However, if the numbers of variables can themselves vary; special attention is required in the representation stage. Jakobi (1998) presents how neural nets controllers of varying sizes for evolutionary robotics simulation could be evolved through genetic algorithms.

Is it possible to capture different technological approach with the same search space?
Diverse approaches to solving a design problem should be examined, and therefore 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 CDS to include radiative, convective and microwave approaches within the same search.

How to relate genotype to phenotype?
One of the important questions in evolutionary computation is how the "genotypes" are related to "phenotypes." A phenotype can be represented through different genotypes. However, these differing genotypes will evolve differently within the search thus having implication on the quality of the final solutions.

3.2 Generation

Can a completely random decision be made for each design step?
If each decision in the generation process can be made randomly, the 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.

Is there a different way to structure the problem such that randomness is used to avoid infeasible solutions while still guaranteeing full coverage?
Complete randomness during the generation stage can be time consuming. This is especially true if the randomness explores a large part of the infeasible space. It is sometimes possible to come up with ways in which we can structure the randomness such that the exploration of the infeasible space is reduced if not eliminated as in the example from Rai and Allada (2003) where an AND/OR graph-based generation technique is utilized.

Should design constraints be hard vs. soft?
A carefully designed 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 space. Including constraints in generation makes them hard constraints since the search will not explore candidates, 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 through the use of penalty functions. Hard constraints tend to reduce computing time but also possibly reduce the effectiveness of an algorithm.

Is the design space smooth/discontinuous?
The characteristics of the design space also affect the generation process. If the design space is smooth and continuous, then methods like SQP tend to work better. However, if the design space is rugged and discontinuous then it is intuitively better to use stochastic techniques, as they overcome the deficiency of traditional optimization. In fact, techniques such as Nelder-Mead simplex method (1965) overcome the limitations inherent in approaches that require a smooth or continuous objective function.

3.3 Evaluation

Are there multiple objectives to handle?
Most algorithms work on a single objective function. Multiobjective problems are mainly handled in two distinct ways. The first is to convert the multi-objective into a single objective (for example through use of weighting method). The second is to consider the "Pareto-optimality" approach. Conversion of multi-objective into single objective enable the use of most of the techniques, however the effectiveness of such a conversion is debated by those adopting pareto-optimal approaches.

Can a simple model be created which captures the true design challenge without external simulation?
Generally, external analysis is time-consuming and only tenuously linked to the search process. In such cases, it is sometimes beneficial to develop an internal evaluation model that captures the true design challenges. This could require extra effort in creating robust and streamlined simulation techniques, but it will be more customized to the problem and potentially lead to higher quality solutions.

How can data transfer with external simulations be managed?
However, if one finds use of external analysis necessary, then care must be taken to link the data transfer between external simulations and the search process. Basically, the quality of solutions is limited to the accuracy of evaluation. However, if the evaluation is taking too long, simplifications must be made.

Will there be a need to handle 'no evaluation' situations?
It is sometimes necessary to handle situations where evaluation information is unreliable (not all candidates may produce valid evaluation information). In such cases, it is necessary to provide for means in the search procedure that can handle these situations.

Does it make sense to minimize or maximize what is being evaluated?
It does not always make sense to maximize or minimize an objective function. The goal of the design problem could be more of *satisficing* (Simon, 1986) where constraint-programming approaches might prove more useful than optimization or other search methods.

3.4 Guidance

Is gradient information available and/or useful?
In many situations where the objective function space is smooth one can easily extract the gradient information and use it through the application of appropriate algorithms to find the optimal solution. However, if the objective function is not smooth then one has to resort to heuristics methods for searching the objective function space.

How well is the space characterized?
Different algorithms work better in different types of space characteristics For example, Sorkin (1991) postulates that simulated annealing works better in fractal-like spaces. Information about search space characteristics could help in preparing better guiding strategies or selecting appropriate algorithms and heuristics whose performance is mainly based on the inbuilt guiding strategies. If the design space is multi-modal in nature, special provisions should be provided in the search process to confront the peaks and crests in the objective function values.

How much memory/experience to utilize? (or How to balance time vs. space?)
Algorithms, which developed 30 years ago in an era when computational memory was costly, tend to avoid accumulating information about past candidates. In general, for all algorithms there is some compromise between the usage of computational memory and time required to solve the problem. In such cases, it is generally suggested that the use of computational memory be leveraged to offset the computational time since computational memory is much cheaper within current computational hardware.
Some algorithms (e.g., Tabu search) generally tend to capitalize on the in-built memory of the algorithm. In using these algorithms, one needs to make careful decisions about how much memory to use. Recalling the quality of past candidates can make a guidance strategy more efficient, but using a higher amount of memory can possibly inhibit novel solutions.

Exploration versus exploitation?
Exploration generally refers to search that constantly diverges into new areas of search space, while exploitation means to focus more locally in the search for optimal candidates. A successful search algorithm should strike a balance between these two. Furthermore, it is generally

Table 1: A comparison of various computational design synthesis papers on the degree to which they describe the representation, generation, evaluation, and guidance aspects of their research.

	Representation	Generation	Evaluation	Guidance
Storn, 1995	*****	***	*****	***
Ongkodjojo, Tay, 2002	*****	***	*****	**
Cinquini, Venini, Nascimbene, Tiano, 2001	**	**	*****	*
Li, Antonsson, 1998	*****	**	*****	***
Rogalsky, Derksen, Kocabiyik, 2000	****	***	*****	***
Jain, Dhar, Kaushik, 1994	**	**	*****	*
Da, Sadler, Jawahir,1996	**	**	*****	*
Fichter, 2000	***	**	***	***
Srinivasan, Tettamanzi, 1996	****	**	****	**
Queipo, Devarakonda, Humphrey, 1994	****	****	***	****
Lam, Delosome, 1988a	****	**	****	***
Corno, Prinetto, Rebaudengo, Sonzareorda, 1997	****	****	***	*****
Talkudar, 1993	***	*****	**	*****
Storn, Price, 1995	***	*****	***	*****
Lam, Delosome, 1988b	**	**	**	*****
Bendsoe, Kikuchi, 1988	***	*****	***	*****

accepted that early in the search process more exploration of the search space occur. Then, when the search begins to converge, it should concentrate or locally exploit the search to find the best local candidate.

4 Results

The computational design synthesis (CDS) approaches present in literature could be broadly classified into two major areas i.e., 1) application driven and 2) algorithm development. The focus of the former is to use already existing algorithms to solve a problem for a particular application. The second category focuses on the development of suitable algorithms or heuristics that are generic enough to be applied to a variety of application problems. The degree to which the four previously discussed elements are presented in each paper may differ. Extracting trends in the focus of these papers could have implications on how the major areas of the CDS framework could or should be presented. In order to extract such information a small literature survey was performed by the authors (16 papers), which comprised literature from both application driven (first 12 papers in Table 1.) and algorithm development (last 4 papers in Table 1.) categories. Each paper was rated on a scale of 1-5 for each of the four parts of CDS framework subjectively by the authors. The overall result of the literature survey is presented in Table 1.

From the table 1, there appears to be a clear distinction in how application driven research is presented compared to algorithmic development research. Generally, application driven research focuses heavily on the *representation* and *evaluation* parts of the CDS framework, while algorithm development research focuses more on *generation* and *guidance*. This makes sense since in developing new applications for existing algorithms, one would need to concentrate on how the candidate solutions are represented and how they are to be evaluated. The generation and guidance strategies tend to be unchanged by different applications, since the strategies in solving them are intrinsic to a given algorithm.

To a certain extent, it is possible to mix and match the procedures from each of the four tasks. In actuality however, there are some limits to recombining these algorithms since the *representation* provides a foundation that dictates the range of *generation* and *guidance* methods. Approaches like Asynchronous Teams (Talukdar, 1993) manage to combine multiple algorithms under a common representation and evaluation scheme, thereby allowing different algorithms to bring together the strengths of their different generation and guidance strategies to find the best solution.

5 Discussion

This paper seeks to generalize a body of techniques collectively known as computational design synthesis (CDS) techniques. The division of computational design synthesis into the four areas of representation, generation, evaluation, and guidance can be a useful approach in

solving complex design problems. Every problem domain has unique challenges that are not always best solved by a single synthesis method. Dividing the problem into these four tasks is a rigorous approach to establishing effective computational synthesis methods.

The degree to which design activities in engineering can be replaced with computational approaches is yet to be determined. Incorporating machine learning, optimization, and other computational methods into engineering complements the traditional view of "computer as analyzer." Furthermore bridging the fields of engineering design with computer science is to marry two fields that each has substantial terminology problems. We, as researchers, have difficulty in describing, presenting, teaching, and ultimately applying techniques when the terminology is so elusive. This paper attempts to generalize this body of work, and distill from it a set of common principles that can be used in presenting past research and in solving new problems.

References

Bendsoe, M.P.; Kikuchi, N. 1988. Generating Optimal Topologies in Structural Design Using A Homogenization Method. *Computer Methods in Applied Mechanics and Engineering* 71: 197-224.

Bentley, P. J.; O'Reilly, U. M. 2001. Ten Steps to Make a Perfect Creative Evolutionary Design System. *GECCO 2001 Workshop on Non-Routine Design with Evolutionary Systems*.

Bentley, P. J.; Kumar, S. 1999. Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. *Genetic and Evolutionary Computation Conference (GECCO '99),* July 14-17 Orlando, FL.

Cadence Design Systems, Inc. 2000. Spectre® Circuit Simulator Advanced Simulator for Complex Circuits. *Datasheet. http://www.cadence.com/datasheets/spectre_cir_sim.html*, San Jose, CA.

Campbell, M. Knowledge discovery in Deep Blue. *Communications of the ACM*. Vol. 42, No.11:65-71.

Campbell, M.; Cagan J.; Kotovsky, K. 2000, Agent-based Synthesis of Electro-Mechanical Design Configurations. *Journal of Mechanical Design*, Vol. 122, No. 1:61-69.

Campbell, M. I. 2001. An Automated Approach to Generating Novel MEMS Accelerometer Configurations. *TEXMEMSIII: Texas-Area MEMS Workshop*, Richardson, TX, June, 7.

Cinquini, C.; Venini, P.; Nascimbene, R.; Tiano, A. 2001. Design of a River-Sea Ship by Optimization. *Structural Multidisciplinary Optimization* 22:240-247.

Corno, F., Prinetto, P., Rebaudengo, M., and Sonzareorda, M., 1997. SAARA: Simulated Annealing Algorithm for Test Pattern Generation for Digital Circuits. *SAC97: 12th Annual ACM Symposium on Applied Computing* San Jose, CA, February 1997, pp. 228-232

Current, J., Min, H., and Schilling, D., 1990."Multiobjective analysis of facility location decisions", European Journal of Operation Research, Vol. 49, pp. 295-307

Da, Z.J., Sadler, J.P., and Jawahir,I.S., 1996,"Multiple Criteria optimization of Finish Turning Operations Based on a Hybrid Model" Proceedings ASME Design Engineering Technical Conferences, DETC96/DAC-1105, August 18-22, Irvine, CA.

Fichter, D. P. 2000, Application of Genetic Algorithm in Portfolio Optimization for the Oil and Gas industry. *SPE Annual Technical Conference*, October 1-4, Dallas, TX.

Glover, F. 1989. Tabu Search-Part 1. *ORSA Journal on Computing*, Vol. 1, No. 3:190-206.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Reading, MA.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems* The MIT Press, Cambridge, MA, 2nd edition.

Jain, S., Dhar, P.L.; Kaushik, S.C., 1994. Optimal Design of Liquid Desiccant Cooling Systems"*American Society of Heating, Refrigrating and Air-Conditioning Engineering Transaction*, pp. 79-86.

Jakobi, N. 1998. *Minimal Simulations For Evolutionary Robotics* PhD. Thesis, University of Sussex.

Kirkpatrick, S.; C. D. Gelatt Jr.; M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* Vol. 220:671-679.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

Lam, J., and Delosme, J.M., 1988, Simulated Annealing: A Fast Heuristic For Some Generic Layout Problems *25th ACM/IEEE Design Automation Conference*. 510-513.

Li, H.; Antonsson, E.K. 1998. Evolutionary Techniques in MEMS Synthesis. *Proceedings ASME Design Engineering Technical Conferences*, DETC98/MECHE-5840, September 13-16, Atlanta, GA.

Mitchell, T. M., 1997, *Machine Learning*, McGraw-Hill, New York.

Nelder, J. A.; Mead, R. 1965. A Simplex Method For Function Minimization *Computer Journal*, 7:308-313.

Ongkodjojo, A.; Tay, F.E.H. 2002 Global Optimization and Design for Microelectromechanical Systems Devices based on Simulated Annealing Journal *of Micromechanics and Microengineering* 12:878-897.

Queipo, N.; Devarakonda, R.; Humphrey, J.A.C. 1994. Genetic Algorithm for Thermosciences research: Application to the Optimized Cooling of Electronic Component. *International Journal of Heat and Mass Transfer* 37:893-908.

Rai, R.; Allada, V. 2003. Modular Product Family: An Agent Based Pareto Optimization Approach. *Submitted to International Journal of Production Research*.

Rogalsky, T.; Derksen, R.W.; Kocabiyik, S., 2000. Differential Evolution in Aerodynamic Optimization. *Canadian Aeronautics and Space Journal*. 4b:183-190.

Schmidt, L.C.; Cagan, J. 1998. Optimal Configuration Design: An Integrated Approach Using Grammars. *Journal of Mechanical Design*. 120:2-9.

Simon, H.A. 1986 Rationality in Psychology and Economics. *The Journal of Business*. Vol. 59, No. 4:209-224.

Soman, A.; Campbell, M. I. 2002. A Grammar-Based Approach To Sheet Metal Design. *ASME Design Engineering Technical Conferences*, Montreal, Quebec, Canada, October 1-3, DAC02-34087.

Sorkin, G.B. 1991. Efficient Simulated Annealing on Fractal Energy Landscapes. *Algorithmica*. 6:367-418.

Srinivasan, D.; Tettamanzi, A. 1996. A Heuristics-Guided Evolutionary Approach To Multiobjective Generation Scheduling. *IEEE Proceedings. C, Generation, Transmission, and Distribution*, Vol. 143, No. 6:553-559.

Stiny G. 1980. Introduction To Shape And Shape Grammars. *Environment and Planning B: Planning and Design*. 7:343-351.

Storn, R. 1995. Differential Evolution Design of an IIR-Filter with Requirements for Magnitude and Group Delay. *IEEE International Conference on Evolutionary Computation ICEC 96*, pp. 268-273,.

Storn, R.; Price, K. 1995. Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. *Technical Report TR-95-026,* University of California, Berkeley, CA

Talukdar, S. 1993. Asynchronous Teams. *Fourth International Symposium on Expert Systems Applications to Power Systems*, La Trobe University, Australia, Jan 4-8.