Evolutionary computation,
engineering design,
manufacturing

Andrew KUSIAK[*]

Evolutionary Computation in Design and Manufacturing

Biology and information technology will have an impact on the design of future products and processes. In this paper, the product and process design challenges are discussed. Evolutionary computation is introduced and its potential applications in design and manufacturing are discussed.

1. INTRODUCTION

The rapid developments in information technology and biology may have a profound impact on engineering design and manufacturing. However, there the exact magnitude of possible changes in industry is difficult to predict. The changes in design and manufacturing are bounded by the following two views:

- Skeptic's View
- Optimist's View

The two views are discussed next.

The Skeptic's View
All products and manufacturing systems are unique.

The Optimist's View
All products across different technologies and manufacturing systems share some commonality.

The two views are illustrated in Figure 1, where the items on the left are essentially products of different degree of granularity and the items on the right are different scale manufacturing systems.

[*] Andrew Kusiak, Intelligent Systems Laboratory, 4312 Seamans Center The University of Iowa, Iowa City, Iowa 52242 - 1527, andrew-kusiak@uiowa.edu,

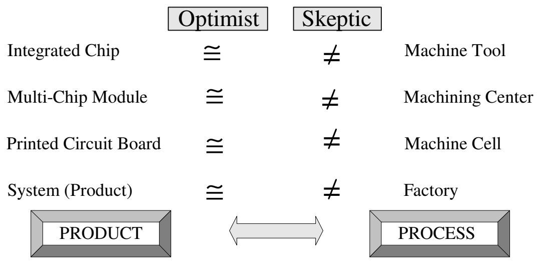
Optimist
Skeptic
Integrated Chip ≅ ≠ Machine Tool
Multi-Chip Module ≅ ≠ Machining Center
Printed Circuit Board ≅ ≠ Machine Cell
System (Product) ≅ ≠ Factory
PRODUCT
PROCESS

Figure 1. The product-process relationship

The two views can be synthesized into the following product-process commonality paradigm.

Product-Process Commonality Paradigm: Products and processes can built from the same constructs defined by their functions and interfaces.

The product-process commonality paradigm states that processes and products can be constructed from the same set of constructs, which seem to hold in industry. For example, a machine tool is a product for the corporation that manufactures it and a process for the corporation that is using it to manufacture other products.

The set of contracts to be used to design and manufacture products and processes makes up the design periodic table illustrated in Figure 2.

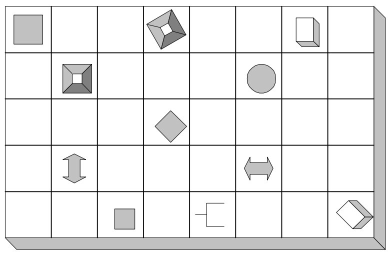

Figure 2. Design periodic table

Such a periodic table could be created if a sufficient number of designs would be described with a common language and made available to the designers. This leads to the need for the development of the Design Description Language (DDL). Some elements of such a language have emerged in electrical design in the form of the Hardware Description Language (HDL).

In the last ten years the industry has undertaken massive efforts to integrate engineering design, manufacturing, and some services, e.g., supply chain. Many organizational and logistics barriers have been removed to improve the communication between product designers, manufacturing engineers, quality engineers, marketing specialists, and other experts that could contribute to the design of products meeting all the requirements, e.g., customer requirements represented by marketing experts or the customers themselves.

It appears that largely due to the progress in information technology, the trend of the last decade aiming at the integration of various business areas will be replaced with the information and financial integration over the organizational integration as illustrated in Figure 3.

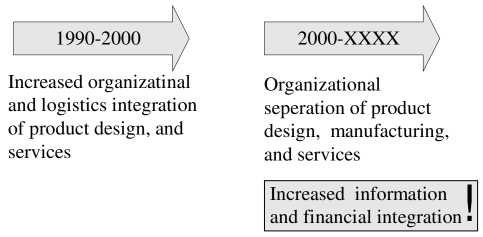
1990-2000
Increased organizatinal and logistics integration of product design, and services
2000-XXXX
Organizational seperation of product design, manufacturing, and services
Increased information and financial integration !

Figure 3. The system integration paradigm in industry

System Integration Paradigm: A corporation of the future is likely to be a system of autonomous components (e.g., product design, manufacturing, marketing) distributed in space and over different time zones with close information and financial integration (Kusiak 2000).

Process-Product Paradigm: Processes are likely to become a valuable subject of increased economic activity in different forms, for example:

- Processes may be sold in the future the same way today's products are sold, e.g., a disease diagnosis process may be sold by a bioinformatics company;
- Sold as product-process bundles, e.g., an airplane entertainment system and its diagnostic system (process), or manufacturing tools and a reconfigurability service for the manufacturing system;
- Benchmarks for expansion, e.g., analysis of business process will proceed a virtual merge transition of business units belonging to different corporations;
- Medium providing competitive advantage, e.g., innovation process.
- Process models may become integrated with products, e.g., a software program managing a supply chain.

The design and manufacturing is undergoing a tremendous change that will be accelerated in the future. As knowledge is becoming the most valuable commodity, the tools for capturing and processing information are of great importance. Evolutionary computation offers formalisms and tools for coping with these challenges.

2. EVOLUTIONARY COMPUTATION

The basic concepts, typology, and literature review of evolutionary computation are presented.

Evolutionary Computation; Problem solving systems that use computational models of evolutionary processes as the key elements in design and implementation. A number of evolutionary computational models have been developed, including evolutionary algorithms, genetic algorithms, the evolution strategy, evolutionary programming, and artificial life.

Evolutionary Algorithm (EA); Algorithm incorporating aspects of natural selection or survival of the fittest. An evolutionary algorithm maintains a population of structures (initially randomly generated) that evolves according to rules of selection, recombination, mutation and survival, referred to as genetic operators. A shared "environment" determines the fitness or performance of each individual in the population. The fittest individuals are more likely to be selected for reproduction (e.g., retention, duplication), while recombination and mutation modify those individuals, yielding potentially superior ones.

EAs differ from genetic algorithms (GAs). A GA generates each individual from some encoded form known as a "chromosome". Chromosomes are combined or mutated to breed new individuals.

EAs are useful for optimization when other techniques such as gradient descent or direct, analytical discovery unsuccessful. Combinatorial and real-valued function optimization, in which the optimization surface or fitness landscape is 'rugged', possessing many locally optimal solutions, are well suited for evolutionary algorithms. The background on various implementations of evolutionary algorithms is provided in Fonseca and Fleming (1995), Bäck (1996), Coello (1999), and Van Veldhuizen and Lamont (2000). The last paper provides a comprehensive typology of EAs. The coevolutionary algorithm is a variation of EA where each individual represents only a partial solution to the problem (see Horn et al. 1994 and Moriarty and Miikkulainen 1998). It is a promising alternative in solving difficult and dynamic problems.

Evolution Strategy (ES); An algorithm where individuals (potential solutions) are encoded by a set of real-valued 'object variables' (the individual's 'genome'). For each object variable an individual has a 'strategy variable' which determines the degree of mutation to be applied to the corresponding object variable. The strategy variables mutate, allowing the rate of mutation of the object variables to vary. An ES is characterized by the population size, the number of offspring produced in each generation and whether the new population is selected from parents and offspring or only from the offspring. ES were proposed in 1963 by Ingo Rechenberg and Hans-Paul Schwefel at the

Technical University of Berlin while searching for the optimal shapes of bodies in a flow.

Eiben and Bäck (1997) presented an extension of the evolution strategy approach to multiparent recombination involving a variable number of parents to create an individual offspring. The extension was experimentally evaluated on a test suite of functions of different modality and separability and the regular/irregular arrangement of their local optima. Multiparent diagonal crossover and uniform scanning crossover and a multiparent version of intermediary recombination are considered in the experiments.

Olafsson (1996) demonstrated the use of evolutionary game theory for allocation of service requirements on to an ensemble of heterogeneous network components. By incorporating differentiated pricing structures into a system utility function, network agents were encouraged to increase their usage of those components, which were poorly utilized. It was demonstrated how this approach could enhance network utilization. The work also reported some new results regarding evolutionarily stable strategies in non-linear evolutionary games.

A road network usually has to fulfill two requirements: (i) it should as far as possible provide direct connections between nodes to avoid large detours; and (ii) the costs for road construction and maintenance, which are assumed proportional to the total length of the roads, should be low. The optimal solution is a compromise between these contradictory demands.. The road optimization problem belongs to the class of frustrated optimization problems. Schweitzer et al. (1997) applied Boltzmann and Darwin and mixed strategies to find differently optimized solutions (graphs of varying density) for the road network, depending on the degree of frustration. They showed that the optimization process occurs on two different time scales. In the asymptotic limit, a fixed relation between the mean connection distance (detour) and the total length (costs) of the network exists that defines a range of possible compromises. Furthermore, they investigated the density of states, which describes the number of solutions with a certain fitness value in the stationary regime. The authors found that the network problem belongs to a class of optimization problems in which more effort in optimization certainly yields better solutions. An analytical approximation for the relation between effort and improvement was derived.

Artificial Life: Non-biological systems, such as computer programs, which evolve to greater levels of fitness by means analogous to natural selection. The process usually requires the algorithms to meet a fitness test before they reproduce, thus improving the fitness of each subsequent generation. The programmer may in some cases define a goal for the competing algorithms (e.g., 'solve this equation').

Genetic Programming (GP): The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs in the scheme computer languages as the solution. A genetic algorithm creates a string of numbers that represent the solution.

Genetic programming, uses four steps to solve problems:

1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3) Create a new population of computer programs.
   i) Copy the best existing programs
   ii) Create new computer programs by mutation.
   iii) Create new computer programs by crossover (sexual reproduction).
4) The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming (Koza 1992, Benzhaf et al. 1998).

Genetic programming is distinguished from other evolutionary algorithms in that it uses a tree (hierarchical) representation of variable size rather than of linear strings of fixed length. The flexible representation scheme is important because it allows the underlying structure of the data to be discovered automatically. One primary difficulty, however, is that the number of solutions may become excessive without any improvement of their generalizabilty. Zhang and Muehlenbein (1995) investigated the fundamental relationship between the performance and complexity of the evolved structures. The essence of the parsimony problem is demonstrated empirically by analyzing error landscapes of programs evolved for neural network synthesis. They considered genetic programming as a statistical inference problem and applied the Bayesian model-comparison framework to introduce a class of fitness functions with error and complexity terms. An adaptive learning method was introduced that automatically balanced the model-complexity factor to evolve parsimonious programs without losing the diversity of the population needed for achieving the desired training accuracy. The effectiveness of this approach was empirically demonstrated on the induction of sigma-pi neural networks for solving a medical diagnosis problem as well as other tasks.

Iba et al. (1995) introduced a new approach to genetic programming by integrating a GP-based adaptive search of tree structures and a local parameter tuning mechanism employing statistical search. In traditional GP, recombination can cause frequent disruption of building blocks, or mutation can cause abrupt changes in the semantics. To overcome these difficulties, they supplemented traditional GP with a local hill-climbing search, using a parameter tuning procedure. More precisely, the authors integrated the structural search of traditional GP with a multiple regression analysis method and establish their adaptive program called STROGANOFF (i.e., STructured Representation On Genetic Algorithms for NOnlinear Function Fitting). The fitness evaluation was based on a minimum description length (MDL) criterion, which effectively controls the tree growth in GP. The authors demonstrated its effectiveness by solving several system identification (numerical) problems and compare the performance of ROGANOFF with traditional GP and another standard technique (radial basis functions). Thereafter they extended STROGANOFF to symbolic (non-numerical) reasoning, by introducing multiple types of nodes, using a modified MDL-based selection criterion, and a pruning of the resultant trees. The effectiveness of this numerical approach to GP is demonstrated by successful application to symbolic regression problems.

Parsimony pressure, the explicit penalization of larger programs, has been increasingly used as a means of controlling code growth in genetic programming. However, in many cases parsimony pressure degrades the performance of the genetic program. Soule and Foster (1998) showed that poor average results with parsimony pressure are a result of "failed" populations that overshadow the results of populations that incorporate parsimony pressure successfully. Additionally, they showed that the effect of parsimony pressure can be measured by calculating the relationship between program size and performance within the population. This measure can be used as a partial indicator of success or failure for individual populations.

Yao et al. (1999) proposed a fast evolutionary programming algorithm that that uses Cauchy instead of Gaussian mutation as the primary operator. The proposed algorithm is efficient in search of a large neighborhood. The authors showed the relationship between the search step size and the probability of finding a global optimum.

The feasibly of the ideas discussed in this paper is illustrated with three examples from the evolutionary computation literature. Lohn and Colombano (1999) presented an evolutionary search method for automatic generation of circuit designs. They used a set of circuit primitives that were synthesized in valid circuits. The algorithm allows for the evolution of the circuit size, circuit topology, and device values. Thompson et al. (1999) presented evolutionary strategy to design a reconfigurable controller. The designed product exhibit better properties than the one designed with conventional constraint based methods. Moriarty and Miikkulainen (1998) applied coevolutionary search to design a neural network. The deigned network was robust due to neurons assuming overlapping roles as well as more diversity.

3. EVOLUTIONARY PROCESS DESIGN

Some of the evolutionary computation concepts discussed in this paper are illustrated with the design of a process. Process modeling involves two notions (see Figure 4):

- Horizontal, and
- Vertical

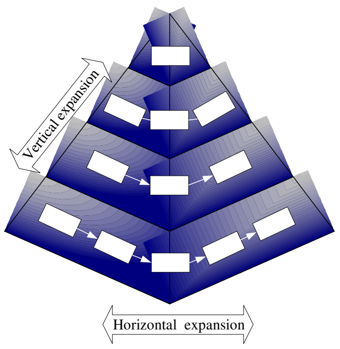

Figure 4. Horizontal and vertical expansion of process model

A process model is seldom developed at one level rather it is built horizontally and vertically. The top node in the hierarchy denotes the overall process that is decomposed into lower level components. The highest granularity model is usually a network of activities (the horizontal notion).

To support the horizontal notion of process modeling concepts from evolutionary computation will be applied. The feasibility of applying evolutionary computation, in particular genetic programming, is illustrated with the IDEF3 methodology (Kusiak 1999). The crossover operator applied to the original process model in Figure 5(a) produces the model in Figure 5(c) by using the submodel in Figure 5(b).

The crossover operator shown in Figure 5 is one of many operators defined in genetic programming that can be applied. The selection of operators could be enhanced with intelligent search strategies.

The concepts from evolutionary computation support the horizontal notion of process modeling. To illustrate the use of evolutionary computation in horizontal process modeling consider the following simplistic algebra that includes examples three activity operators:

- Specialize
- Generalize
- Mutate

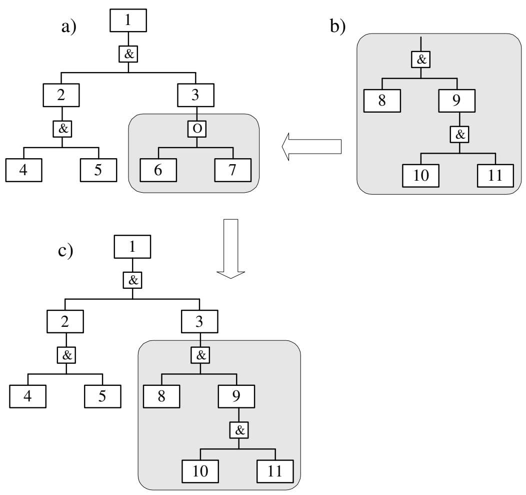

Figure 5. IDEF3 tree involved in crossover

To illustrate the there operators consider the model in Figure 6 that is represented in a simplified form in Figure 7(a).

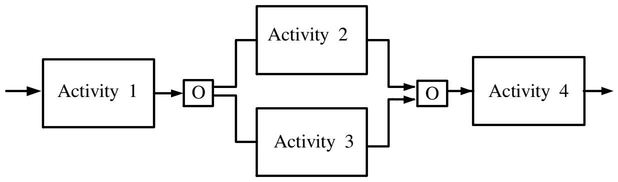

Figure 6. Simple process model

The generalization operator transforms the model in Figure 7(a) in the model in Figure 7(b) by incorporating activity 5. Similarly the specialization and mutation operations are illustrated in Figure 7(b) and 7(c).

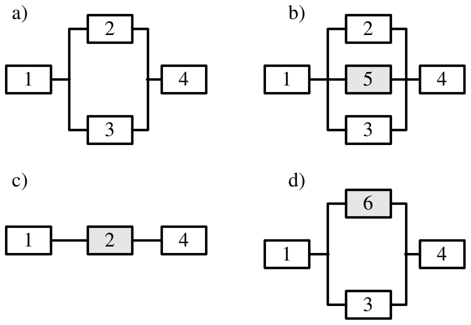

Figure 7. Activity model operators: (a) reference model, (b) generalization, (c) specialization, (d) mutation

In addition to the activity operators, algebra for inputs, outputs, controls, mechanisms, and logical connectors can be defined. For example, the generalize operator applied to an Exclusive OR connector would transform it into an OR-connector as illustrated in Figure 8(b). An add operator applied to output-input O-I1-4 inserts this input between activities 1 and 4 as shown in Figure 8(c).

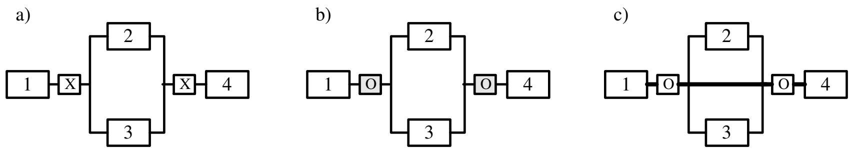

Figure 8. Logical junction and input operators: (a) base model, (b) generalization of Exclusive OR junction, (c) Add Input-Output1-4 operator

4. CONCLUSION

In this paper, three paradigms governing the design and manufacturing of the future were defined. The background of evolutionary computation and examples of its use in design and manufacturing were discussed.

REFERENCES

[1] Bäck, T. (1996), Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York.
[2] Benzhaf, W., Nordin, P., Keller, R.E., and F.D. Francone (1998), Genetic Programming; An Introduction, Morgan Kaufmann, San Francisco, CA.
[3] Coello, C.A.C. (1999), A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge and Information Systems, Vol. 1, No. 3, pp. 269-308.
[4] Eiben, A.E. and T. Bäck (1997), Empirical investigation of multiparent recombination operators in evolution strategies, Computational Intelligence, Vol. 5, No. 3, pp. 347-365
[5] Fonseca, C.M. and P.J. Fleming (1995), An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation, Vol. 3, No. 1, pp. 1-16.
[6] Horn, J., Goldberg, D.E., and K. Deb (1994), Implicit niching in a learning classifier system: Nature's way, Evolutionary Computation, Vol. 2, No. 1, pp. 37-66.

[7] Iba, H., deGaris, H., and T. Sato (1995), A numerical approach to genetic programming for system identification, Evolutionary Computation, Vol. 3, No. 4, pp. 417-452.

[8] Koza, Z. (1992), Genetic Programming, MIT Press, Cambridge, MA.

[9] Kusiak, A (2000), Computational Intelligence in Design and Manufacturing, John Wiley, New York.

[10] Kusiak, A. (1999), Engineering Design: Products, Processes, and Systems, Academic Press, San Diego, CA.

[11] Lohn, J.D. and S.P. Colombano (1999), A circuit representation technique for automated circuit design, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 3, pp. 205-219.

[12] Michalski, R.S., Mozetic, I., Hong, J., and N. Lavrac (1986), The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, Proceedings of the 5th National Conference on Artificial Intelligence, AAAI Press, Palo Alto, CA, pp. 1041-1045.

[13] Moriarty, D.E. and R. Miikkulainen (1998), Forming neural networks through efficient ad adaptive coevolution, Evolutionary Computation, Vol. 5, No. 4, pp. 373-399.

[14] Olafsson, S. (1996), Resource allocation as an evolving strategy, Evolutionary Computation, Vol. 4, No. 1, pp. 33-55.

[15] Schweitzer, F., Rosé, H., Ebeling, W., and O. Weiss (1997), Optimization of road networks using evolutionary strategies, Evolutionary Computation, Vol. 5, No. 4, pp. 419-438.

[16] Soule, T. and J.A. Foster (1998), Effects of code growth and parsimony pressure on populations in genetic programming, Evolutionary Computation, Vol. 6, No. 4, pp. 293-309.

[17] Thompson, A., Layzell, P., and R.S. Zebulum (1999), Exploration in design space: Unconventional electronics design through artificial evolution, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2, pp. 167-196.

[18] Van Veldhuizen, D.A. and G.B. Lemont (2000), Multiobjective evolutionary algorithms: Analyzing the state-of-the-art, Evolutionary Computation, Vol. 8, No. 2, pp. 125-147.

[19] Yao, X., Liu, Y., and G. Lin (1999), Evolutionary programming made easier, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2, pp. 82-102.

[20] Zhang, B.-T. and H. Muehlenbein (1995), Balancing accuracy and parsimony in genetic programming, Evolutionary Computation, Vol. 3, No. 1, pp.17-38.