Principles of modularity, regularity, and hierarchy for scalable systems

Hod Lipson*

Mechanical & Aerospace Engineering, and Computing & Information Science, Cornell University, Ithaca, 14850 NY, USA

Scalability of open-ended evolutionary processes depends on their ability to exploit functional modularity, structural regularity and hierarchy. This paper offers a number of observations about properties, dependencies and tradeoffs among these principles and proposes a formal model where such elements can be examined.

Keywords: evolutionary computation, hierarchy, modularity, regularity, tiling problem

1. INTRODUCTION

Scalability of open-ended evolutionary processes depends on their ability to exploit functional modularity, structural regularity and hierarchy. Functional modularity creates a separation of function into structural units, thereby reducing the amount of coupling between internal and external behaviour on those units and allowing evolution to reuse them as higher-level building blocks. Structural regularity is the correlation of patterns within an individual. Examples of regularity are repetition of units, symmetries, self-similarities, smoothness, and any other form of reduced information content. Regularity allows evolution to specify increasingly extensive structures while maintaining short description lengths. Hierarchy is the recursive composition of function and structure into increasingly larger and adapted units, allowing evolution to search efficiently increasingly complex spaces.

The existence of modular, regular and hierarchical architectures in naturally evolved systems is well established [7, 4]. Though evolutionary processes have been studied predominantly in biological contexts, they exist in many other domains, such as language, culture, social organization and technology [1], among many others. But principles of modularity, regularity and hierarchy are nowhere as dominant as they are in engineering design. Tracing the evolution of technology over generations of products, one can observe numerous instances of designs being encapsulated into modules, and those modules being used as standard higher-level building blocks. Similarly, there is a pressure to reduce the information content in designs, by repeating or reusing the same modules where possible, using symmetrical and regular structures, and standardizing on components and dimensions. These and other forms of regularity translate into reduced design, fabrication and operation costs. The organization of engineering designs, especially as complexity increases, is typically hierarchical. The hierarchy is often organized such that the amount of information is distributed uniformly across levels, maintaining a 'manageable' extent of information at each stage. These principles of modularity, regularity and hierarchy are cornerstones of engineering design theory and practice [e.g. 6]. Though these principles are well established, there is—like with biological evolution—still a lack of formal understanding of how and why modular, regular and hierarchical structures emerge and persist, and how we can computationally emulate these successful principles in design automation processes.

In this short paper I would like to highlight a number of observations about properties, dependencies and tradeoffs among these principles and propose a formal model where such elements can be examined.

2. DEFINITIONS AND METRICS

Functional modularity is the structural localization of function.

In order to measure functional modularity, one must have a quantitative definition of function and structure. It is then possible to take an arbitrary chunk of a system and measure the dependency of the system function on elements within that chunk. The more that the dependency *itself* depends on elements outside the chunk, the less the function of that chunk is localized, and hence the less modular it is. If we represent dependencies as second derivatives of function with respect to pairs of parameters (i.e. the Hessian matrix of the fitness), then modules will be collections of parameters that can be arranged with lighter off-diagonal elements [9].

Structural regularity is the compressibility of the description of the structure.

The more the structure contains repetitions, near-repetitions, symmetries, smoothness, self similarities, etc, the shorter its description length will be. The amount of regularity can thus be quantified as the inverse of the description length or of its Kolmogorov complexity.

* E-mail: hod.lipson@cornell.edu

Journal of Biological Physics and Chemistry **7** (2007) 125–128
Received 20 December 2007; *accepted* 31 December 2007

Hierarchy of a system is the recursive composition of structure and/or function.

The amount of hierarchy can be quantified given the connectivity of functional or structural elements (e.g. as a connectivity graph). The more the distribution of connecting path lengths among pairs of elements approximates a power law distribution, the more hierarchical the system is. For example, in a perfectly balanced binary tree of depth d, there are about twice as many paths of length $2d$ between leaves than there are paths of length $2d–2$, implying that path lengths between leaves are distributed exponentially. Conversely, in a fully connected graph, all path lengths are of constant length.

3. OBSERVED PROPERTIES

The following sections describe some observations on the interplay between modularity, regularity and hierarchy.

3.1 Modularity and regularity are independent principles

Principles of modularity and regularity are often confused in the literature through the notion of *reuse*. Indeed, modularity has several advantages, one of which is that modules can be used as building blocks at higher levels, and therefore can be *repeated*. Nonetheless, it is easy to imagine a system that is composed of modules, where each module appears only once. For example, opening the hood of an (old) car reveals a system composed of a single engine, a single carburettor, and a single transmission. Each of these units appears only once (i.e., is not reused anywhere else in the system), but can be considered a module because its function is localized. Its evolutionary advantage is that it can be adapted more independently, with less impact of the adaptation on the context. A carburettor may be swapped for a newer technology, without affecting the rest of the engine system. We thus have modularity without regularity.

Similarly, there are instances of regularity without modularity: The smoothness of the hood of the car, for example, reduces the information content of the structure but does not involve the reuse of a particular module.

Though these principles are independent, they often appear in tandem and hence the confusion: we tend to speak of useful modules being reused as building blocks, and indeed recurrence of a pattern may be an indication of its functional modularity, though not a proof of it.

3.2 Tradeoffs between modularity and regularity

An inherent tradeoff exists between modularity and regularity through the notion of *coupling*. Modularity, by definition, reduces coupling, because it involves the localization of function. But regularity *increases* coupling because it reduces information content. For example, if a module is reused in two different contexts, then the information content of the system has been reduced (the module needs to be described only once and then repeated), but any change to the module will have an effect on both places. Software engineers are well aware of this tradeoff: as a function is encapsulated and called from an increasing number of different contexts in a program, so does modifying it become increasingly difficult because it is entangled in so many different functions.

The tradeoff between modularity and reuse is also observed in engineering as the tradeoff between modularity and optimality. Modularity often comes at the expense of optimal performance. Systems that are less modular, i.e. more integrated, can be more efficient in their performance as information, energy and materials can be passed directly within the system, at the expense of increased coupling. Software engineers are familiar with ‘long jumps’ and ‘global variables’ that have this effect; similarly, mechanical products will often achieve optimality of performance or cost through integration of parts into monolithic components wherever possible. The increased performance gained by reduction of modularity is often justified in the short term, whereas increased modularity is often justified over longer time scales where adaptation becomes a dominant consideration.

3.3 Properties of the problem or the solution?

It is not clear whether modularity, regularity and hierarchy are properties of the system being evolved (i.e. the ‘solution’), or of the target fitness specification (i.e. the ‘problem’). It may well be that there is a duality between these viewpoints. The evolutionary computation literature contains several instances of test functions that are themselves modular (separable, e.g. Royal Roads [5]), hierarchical (e.g. Hierarchical-IFF [8]), and regular (e.g. One-Max). The Royal Roads test problem contains a concatenation of fixed-size blocks of bits, where each block can be optimized independently. Hierarchical-IFF is a binary tree composition of pairs of blocks starting from single bits, where the optimal configuration of each block at each level depends on the value of its sibling block at the same level. One-Max is a test problem that requires the entire bitstring to be identical. It is not surprising then to see corresponding algorithms that are able to exploit these properties and find the solutions to these problems.

Engineers often go to great lengths to describe design goals in a way that is solution-neutral, i.e., that describes target functionality while placing the least constraints on the solution. Indeed engineering design is

notorious for having multiple—even many—solutions to any given problem, without any solution being clearly superior. The fact that modular, regular and hierarchical solutions are more attractive may be because the design process itself tends to prefer them for reasons of scalability. It is therefore plausible that in a search for scalable algorithms for synthesizing solutions bottom up, we should avoid test functions that have an inherent modular or hierarchical reward, and have these solution properties emerge from the search process itself.

4. AN ABSTRACT MODEL

As a matter of research methodology it is convenient to have an abstract and simple domain where the principles in question can be examined quickly, accurately and transparently. We would like a problem domain in which

- Problems and solutions can be open-ended: be of unbounded complexity;
- There is a clear interpretation for both structure and function;
- Problems and solutions can be synthesized from basic building blocks;
- Problems can be specified in a solution-neutral way;
- Modularity, regularity, and hierarchy can exist and can be quantified;
- Problems have multiple, valid solutions that exhibit varying degrees of modularity, regularity and hierarchy.

One such suitable domain is the *geometrical tiling problem* [3]. Tiling problems are an area of extensive mathematical interest and have been studied extensively over centuries. They are subject to elaborate mathematical analysis and proofs, and yet are simple and intuitive to understand as puzzles. The tessellating tile problem has been used as a test function for evolution once before [2], but here it is proposed that the tiling problem itself has properties amenable to the analysis of modularity, regularity, and hierarchy.

Tiling problems can be specified at any dimension, and with any number of primitive tiles, but for the purpose of illustration let us consider the case of a two-dimensional tiling problem using a single tile.

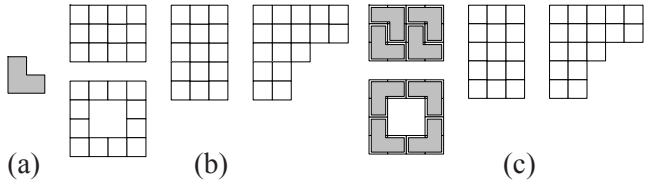

Figure 1. Tiling problem model of synthesis. (a) A primitive tile; (b) a problem: an area to be tiled; and (c) a solution: tiling of the area. Some problems cannot be solved, and some can be solved in many different ways, some more and some less modular, regular, hierarchical.

Consider the example shown in Figure 1. A single L-shaped tile constitutes the building block of this domain. It can appear in any of four orientations. The 'problem' is to tile a given area completely, such as the area shown in Figure 1b. The actual tessellation of the area is the 'solution' to the problem, as shown for some of the problems in Figure 1c.

In tiling problems, the arrangement of tiles is the structure of the solution, and the geometry of the covered area is the function of the solution. Thus an area can specify a target function (a problem) without describing the structure of the solution. Some problems may be simple, some may be difficult, and some may be impossible. For example, the two untiled problems in Figure 1c are impossible: The right grid is impossible to tile because it has an area that is not an integer multiple of the tile area, and so it is 'outside the scope of the search space'. The 3×5 grid *is* an integer multiple of the tile area, but still is not tilable. It is not always clear what makes a given area easy or difficult to solve.

Multiple solutions with different structures but identical function may exist. For example, Figure 2 shows a 4×6 function that is solved using two different structures. Clearly, the left solution is more regular than the solution on the right. The solution on the left is probably easier to come by if the highlighted set of building blocks is discovered early and encapsulated as module.

Some tiling problems have only nonperiodic solutions, and are similar to a needle in a haystack. Other tiling problems, such as the one shown in Figure 2b, have a self-similar solution. The highlighted set of building blocks in Figure 2b is geometrically similar to its basic building block. It can be shown that in contrast with periodic tiling, in self-similar tiling it is not possible to determine the next tile to place using only local information.

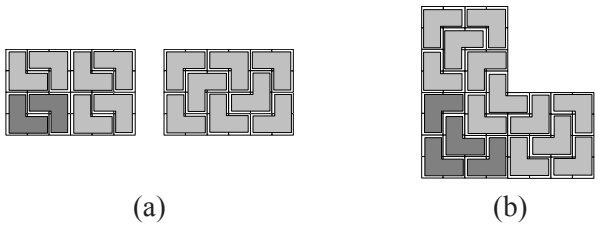

Figure 2. Tiling problems: (a) two different structures achieve the same function, but one is less regular and modular. (b) A self-similar hierarchical tiling. Highlighted elements constitute a potential module.

5. CONCLUSIONS

This paper proposes some quantitative definitions for the concepts of modularity, regularity and hierarchy. A number of observations about the properties, dependencies and tradeoffs among these principles have been discussed:

modularity and regularity are distinct, independent properties that embody a tradeoff due to coupling. They are desired, though not necessary properties of the solution, not the problem. Geometrical tiling problems are one domain where such issues can be studied.

ACKNOWLEDGMENT

This paper is an adaptation of a previously published position statement delivered at a workshop on modularity in the 2004 Genetic and Evolutionary Computation Conference.

REFERENCES

1. Basalla, G. *The Evolution of Technology*. Cambridge: University Press (1989).
2. Bentley, P.J. & Kumar, S.Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. In: *Genetic and Evolutionary Computation Conference (GECCO '99)*, 14–17 July 1999, Orlando, Florida. pp. 35–43. RN/99/2.
3. Grünbaum, B. & Shephard, G.C. *Tilings and Patterns*. New York: W. H. Freeman (1987).
4. Hartwell, L.H., Hopfield, J.H., Leibler, S. & Murray, A.W. From molecular to modular cell biology. *Nature* **402** (1999) C47–C52.
5. Mitchell, M. *An Introduction to Genetic Algorithms*. Cambridge, Mass.: MIT Press (1996).
6. Suh, N.P. *The Principles of Design*. Oxford: University Press (1990).
7. Wagner, G.P. & Altenberg, L. Complex adaptations and the evolution of evolvability *Evolution* **50** (1996) 967–976.
8. Watson, R.A. & Pollack, J.B. Hierarchically-consistent test problems for genetic algorithms. In: *Proc. 1999 Congress on Evolutionary Computation (CEC 99)* (eds Angeline, Michalewicz, Schoenauer, Yao and Zalzala), pp.1406–1413. IEEE Press (1999).
9. Wyatt, D. & Lipson, H. Finding building blocks through eigenstructure adaptation. In: *Genetic and Evolutionary Computation Conf. (GECCO '03),* 12–16 July 2003, Chicago, Illinois. pp. 1518–1529.