

Network motifs in computational graphs: A case study in software architecture

Sergi Valverde¹ and Ricard V. Solé^{1,2}

¹*ICREA-Complex Systems Lab, Universitat Pompeu Fabra, Dr. Aiguader 80, 08003 Barcelona, Spain*

²*Santa Fe Institute, 1399 Hyde Park Road, New Mexico 87501, USA*

(Received 1 July 2004; revised manuscript received 3 June 2005; published 8 August 2005)

Complex networks in both nature and technology have been shown to display characteristic, small subgraphs (so-called motifs) which appear to be related to their underlying functionality. All these networks share a common trait: they manipulate information at different scales in order to perform some kind of computation. Here we analyze a large set of software class diagrams and show that several highly frequent network motifs appear to be a consequence of network heterogeneity and size, thus suggesting a somewhat less relevant role of functionality. However, by using a simple model of network growth by duplication and rewiring, it is shown the rules of graph evolution seem to be largely responsible for the observed motif distribution.

DOI: [10.1103/PhysRevE.72.026107](https://doi.org/10.1103/PhysRevE.72.026107)

PACS number(s): 89.75.Fb, 89.20.Ff, 87.80.Vt

I. INTRODUCTION

Many natural and artificial systems are describable as networks of interacting components [1–4]. The network is a medium that allows resource sharing often involving an efficient transport of energy (metabolism, power grid), matter (highways, airport webs), or information (cellular communication, Internet). The architecture of complex networks can be explored at different scales, from the overall properties defined by average measures such as path length or clustering, correlations, or degree distributions to the more fundamental features displayed by small subsystems. In this context, it has been shown that some special, small subgraphs—so-called *motifs*—seem to be particularly relevant in describing the architecture of complex networks [5]. Motifs have been suggested to be the functional building blocks of network complexity. Are some subgraphs more common than others because their functional relevance?

An alternative view is that the rules of network growth can by themselves favor some subgraphs with no special relation to the underlying functionality. Actually, this seems to be the case for the structure of the protein-protein interaction map. In spite of the fact that proteins perform functions, the overall architecture of the protein network is easily reproduced by means of a simple model of node duplication plus rewiring [6]. Such properties include scale freeness, small-world features, and even hierarchical organization and protomodularity [7]. Mounting evidence suggests that many key features of complex networks (including motifs) might be strongly tied to the global network structure [8].

If functional constraints to network architecture have to be considered, one particularly relevant aspect of network complexity is associated with the presence of some underlying computational process. Computation is a key ingredient of any complex adaptive system (CAS). By storing and processing information, CAS's are able to predict (and adapt to) external fluctuations. Computation occurs in both natural and artificial systems [9], although the building process that creates the computational structure is different. This is actually one of the most important points here: are the rules of designed and evolved systems completely different? Biological networks are largely originated through tinkering [10–12]: new components are obtained by re-using old ones, mainly

by duplicating them. In spite of the apparent limitations of such mechanisms, it allows one to discover good designs [13]. More complex computations can be developed as the network size is increased and new functions can emerge.

How is computation linked to network structure? Tentative answers, to be developed here, can be obtained by looking at a very important class of computationally driven networks: software systems. They offer a unique opportunity of exploring different levels of complexity with well-defined functional traits. As opposed to most examples of evolving computational networks, extensive databases storing software evolution registers exist and involve a high degree of detail. Here we analyze the largest data set of software maps explored to date (83 different systems). The main goal of our study is to see if functionality, as opposed to network evolution, is a main constraint to the distribution of network motifs in real graphs. The paper is organized as follows. In Sec. II an overview of the software systems analyzed here is given. In Sec. III, the statistical patterns of network motifs in a large set of software systems are presented and the presence of scaling relations and the size-dependent frequency of motifs analyzed. In Sec. IV a model of duplication and rewiring is used in order to reproduce the structure of motifs of a large software map. In Sec. V a general discussion is presented.

II. SOFTWARE NETWORKS

Programming languages describe software systems [14]. Every computer program has a textual representation following syntactic rules dictated by a programming language [15]. The program is decomposed in a number of simpler software entities or logical elements, which are given a unique name. Software entities include things as data objects, instructions, subprograms, or modules. A hierarchy or natural ordering between software entities is prompted by modern programming languages. At the lowest level, a program is viewed as a sequence of simple machine instructions. Sequences of related instructions are enclosed in subprograms. At the highest level, there are modules or logical containers grouping simpler software entities. Often, modules are defined as functional blocks but there are no strict principles driving module composition.

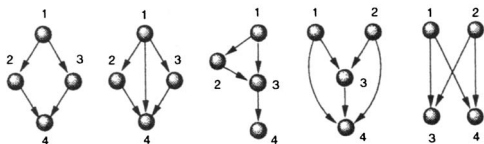


FIG. 1. Examples of common network motifs with $n=4$ elements found in software graphs. Here each node is a class and arrows indicate static dependences among classes (see text).

It is useful to depict the complex structures defined in computer programs by means of a graph [16], where nodes represent software entities and links represent syntactical relationships between modules, subprograms, and instructions (see Fig. 1). In this paper, we will focus on a particularly interesting subset of software entities: the collection of modules and their static interactions (also called software architecture). While it is widely acknowledged that software evolution depends on its architecture, very little is known about the cause and effect relationships between design practices and evolution outcomes [17]. In order to understand the relation between structure and artificial evolution, we have envisaged a network model of software architecture, hereafter called the software map or software network. Here, we show how the evolution of computer programs can be understood by recovering and analyzing their software networks at different stages of development.

Following [16,18], Fig. 2 shows the text of an incomplete C++ [19] computer program [see Fig. 2(a)] and its corresponding software map [see Fig. 2(b)]. The program text reads from left to right and top to bottom. The software network $\Omega=(V,L)$ of this C++ program is recovered by means of a very simple lexical analysis. First, we identify the vector of all module names (also class in C++) given by $W=(w_i)=(\text{point, chessmen, point, move, point, point, pawn, chessmen, move})$. Name ordering is important when recovering module dependences (see below). Names w_i that appear in the head of a module declaration provide the set of

network nodes $V=\{v_i\}$. These names (hereafter called module definitions) are easily identified because they are preceded by the C++ keyword *class*. Remaining names are called module references. In this example, we have four ($N=|V|=4$) unique module definitions: point (w_1), chessmen (w_2), move (w_4), and pawn (w_7). This defines a mapping from names $w_k \in W$ and network nodes $v_i \in V$ in the software network.

The design of any nontrivial function involves the interaction of at least two modules [20]. Static module interactions can be depicted from relative positions of names in W . Let us assume that $w_k \in \{w_1, w_2, w_4, w_7\}$ is a module definition associated with node v_i and $w_l \notin \{w_1, w_2, w_4, w_7\}$ is a module reference associated with node v_j . A directed link $\{v_i \rightarrow v_j\} \in L$ signals a dependence from module definition w_k to module reference w_l . Link directionality reflects name ordering in the C++ program—that is, $k < l$. There are two types of module dependences: association (also “has a” relationship) or inheritance (or “is a” relationship). The purpose of these dependences is to establish a logical organization of the system. However, our analysis is centered on the study of topological patterns and does not take into account detailed relationship semantics. In an association, referenced node v_j is nested in the C++ block of module v_i . This block is always bracketed by the symbols {and}. In an inheritance, referenced node v_j always follows the C++ sequence: public after the referencing node v_i [see Fig. 2(a)]. Repeated links are not considered in the following analysis.

Software maps capture the topology of complex software systems. In particular, these maps provide a quantitative approach to the evolution of technology. They are actually evolving entities and somewhat inhabit an intermediate zone between computing machines and neural structures. We have shown software networks to be scale free and small world [16,18,21]. Software networks can be described under a statistical physics perspective.

III. SOFTWARE MOTIFS

In this section, we extend our previous topological studies by analyzing software networks at the level of network subgraphs, or subsets of connected nodes in a network. The statistics of subgraphs provides important information about network structure. It has been claimed that overrepresented subgraphs (i.e., motifs) signal key building blocks of networks [5]. This might be the case for regulatory networks, where specific subgraphs (i.e., feed-forward loops) perform information processing functions [22]. A particular class of subgraphs, cycles, have received considerable interest. Cyclical dependences in software maps imply that a module is related to itself, which may be acceptable, unacceptable, or required [23]. Ambiguity in the functional meaning of cycles suggests that subgraphs in software graphs are not strictly related to well-defined functions. The ubiquity of subgraphs in software networks seems to be a consequence of top-down mechanisms of software organization and not a consequence of selective pressures.

Following the method outlined in previous section, we have recovered and analyzed a large dataset of software

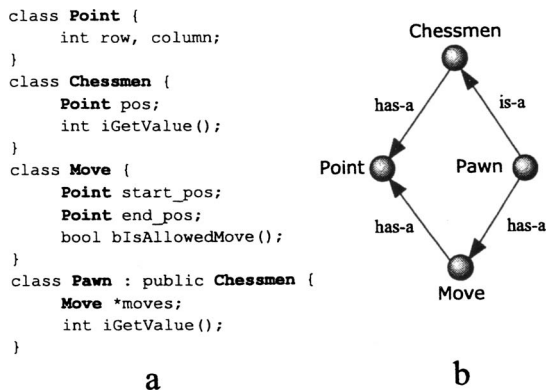


FIG. 2. (a) A piece of C++ code from a chess-playing program and (b) the corresponding software map or network model displaying the collection of modules and their logical dependences. The only information required to recover the software map is the set of module names [highlighted in bold in (a)] and their relative locations in the C++ program (see text). Notice how nodes are labeled with their names and links are decorated with relationship type.






















Network	Nodes	Edges	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}
Software Networks (medium)			 FFL S38			 Bi-parallel S904			 FFL extended S344								
Fairuse	106	180	41	11.6±3.3	8.94	18	7.3±3.5	3.01	33	9.5±5.5	4.25						
Aime	143	319		n/a		230	30.8±19	10.46	55	31.8±9.3	2.49						
Filezilla221a	186	331	77	29.4±6.1	7.86	30	10.2±4.5	4.38		n/a							
Aztec	255	391	68	26±5.4	7.82	25	10.6±4.6	3.15	68	14±5.9	9.08						
Exult	261	504	107	56.3±8.4	6.01	86	30.2±8	6.75	182	80.2±19.6	5.18						
Software Networks (large)			 Bi-fan S204			 Bi-parallel S904			 Multi-X S2252			 Multi-Z S206			 Multi-Y S2190		
blender226	495	834	486	138±30.3	11.4	33	16±5.2	3.2	123	7.8±5.8	20	22	3.7±3.6	5.04	18	4.2±3.2	4.37
gtk221	748	1147	644	189±33.8	13.4	119	25±7.5	12.5	173	26±16.7	8.8	21	2.0±1.7	10.9	19	4.2±2.7	5.41
vtk	771	1362	512	262±39.9	6.27	295	69±14.3	15.8	41	6.3±3.2	10.7	193	27.7±14.7	11.2	122	12.8±5.8	18.7
java2	1364	1947	816	189±35.5	17.7	173	48±10.8	11.5	345	18.4±14	23.3	22	2.2±2.1	9.5	17	3.8±2.2	5.99
prorally	1993	4987	22750	1840±171	122	3848	322±34.2	103.1	1080	144±50.6	18.4	210	28.7±9.2	19.8	1318	55.5±14.7	85.9
Software Networks (large)			 FFL S38			 S472			 Multi-X incomp. S2186			 Multi-X incomp. S408			 FFL extended S344		
blender226			126	33.8±6.1	15		N/A		1976	766±162	7.43	436	196±65	3.7	94	26.2±8.6	7.88
gtk221			118	47.7±9.6	7.31	15	3.1±2.3	5.06	4177	1941±398	5.6	1462	748±261	2.73	188	68.1±13.7	8.73
vtk			229	81.6±10.6	13.9	30	13±6.7	2.53	707	388±61.6	5.17	333	217±44	2.62	718	212.1±49.1	10.3
java2			176	46.2±9	14.4	8	1.8±1.4	4.57	10212	6346±1180	3.2	2494	1397±522	2.1	257	52.5±17.6	11.6
prorally			1169	272±21	42.6	282	30.8±10.4	24.2	25099	15482±1750	5.5	5742	4163±752	2.1	2909	736±101.4	21.4
Gene Regulation (transcription)			 FFL S38			 Bi-fan S204											
E. coli	424	519	40	7±3	10	203	47±12	13									
Neurons			 FFL S38			 Bi-fan S204			 Bi-parallel S904								
C. elegans	252	509	125	90±10	3.7	127	55±13	5.3	227	35±10	20						
Electronic Circuits (forward logic chips)			 FFL S38			 Bi-fan S204			 Bi-parallel S904								
s15850	10383	14240	424	2±2	285	1040	1±1	1200	480	2±1	335						

FIG. 3. Network motifs with $n=3, 4$ elements found in software graphs. The numbers of node and edges for each network are shown. The most frequent motifs were classified in distinct rows for each type of system: medium software systems, large software systems, gene regulatory nets, neural networks, and digital electronic circuits. For each motif, we display the number of occurrences in the network (N_{real}), the number of occurrences ($N_{rand} \pm SD$) in a set of 100 randomized network versions, and a qualitative measure of its statistical significance as given by the Z score (see text). Medium and large software networks share a large amount of motifs but we found larger variability in the medium data set. A remarkable difference is motif 2190 (the last motif in the second row), which appeared only in the context of large software systems.

maps $\{\Omega\}$ from 83 reverse-engineered C++ software systems. A given graph can be characterized by a degree sequence. For the whole graph Ω each node has a degree sequence given by the in-degree list $\{K_i\}$ and the out-degree list $\{R_i\}$ (with $i=1, \dots, N$). The lists would be completed by the so-called mutual edges $\{M_i\}$ —i.e., cases where there is a pair of edges in both directions between two nodes. For each subgraph $\Omega_i \subset \Omega$ of size n (here $n=3, 4$), another degree sequence would be provided by two new lists, now $\{k_j\}$ and $\{r_j\}$ for the in- and out-degrees, respectively. For example, for the left subgraph in Fig. 1, we would have $\{k_j\}=\{0, 1, 1, 2\}$ and $\{r_j\}=\{2, 1, 1, 0\}$.

Network motifs are defined in terms of subgraphs which appear much more often than expected from pure chance. Specifically, they occur with a high frequency compared with the expected from an ensemble of randomized graphs with identical degree structure [5]. The random networks are generated by means of the switching rule. For every pair of links $i \rightarrow j$ and $u \rightarrow v$ in the original software network, we add the

pair $i \rightarrow v$ and $j \rightarrow u$ in the randomized network. This rule keeps intact the in- and out-degree sequences but destroys degree-degree correlations. The statistical significance of a given subgraph Ω_i is described by its Z score [5], defined as

$$Z(\Omega_i) = \frac{N_{real}(\Omega_i) - \langle N_{rand}(\Omega_i) \rangle}{\sigma[N_{rand}(\Omega_i)]}. \quad (1)$$

Here $N_{real}(\Omega_i)$ is the number of times the subgraph appears in the network, whereas $\langle N_{rand}(\Omega_i) \rangle$ and $\sigma[N_{rand}(\Omega_i)]$ refer to the mean and standard deviation (SD) of its appearances in the randomized ensemble, respectively. In order to be significant, it is required that $|Z(\Omega_i)| > 2$. When $Z(\Omega_i) > 2$ [$Z(\Omega_i) < -2$] the motif [antimotif] is considered to be more [less] common than expected from random. In Fig. 3 the results from our analysis are shown for some typical software networks. A handful of these subgraphs appear to be present in all software systems analyzed and also in both electronic circuits and biological networks involving compu-

tation. This is the case of Bi-parallel (S904), Bi-fan (S204), the feed-forward loop (S38), and its close variants (such as S2186 and S408). Such a common point might be easily interpreted in functional terms: similar subgraphs are abundant because they are selected or chosen to perform a given function or task. As shown below, no evidence from statistical patterns supports such view.

Assuming sparse graphs ($\langle K \rangle \ll N$), the probability of a given subgraph Ω_i can be estimated. Following Itzkovitz *et al.*, [24] we can see how this is calculated using the first subgraph in Fig. 1. Here we have $\{K_j\}=\{2, 1, 1, 0\}$ and $\{R_j\}=\{0, 1, 1, 2\}$. The idea is to compute the different probabilities associated with each directed edge linking all pairs of nodes. For example, the probability of having a directed link from node 1 to node 2 (for $K_1 R_2 \ll N \langle K \rangle$) is approximately

$$P(1 \rightarrow 2) = \frac{K_1 R_2}{N \langle K \rangle}, \quad (2)$$

which can be interpreted as follows [24]: we perform K_1 attempts for the first node to connect to the target node with a probability $R_2/N \langle K \rangle$. Similarly, we would have

$$P(1 \rightarrow 3) = \frac{(K_1 - 1) R_3}{N \langle K \rangle} \quad (3)$$

being the approach used for all edges. The average number of appearances of Ω_i is finally computed by averaging. Itzkovitz *et al.* [24] have shown that the average number of appearances $\langle G \rangle$ of a given subgraph is given by a product of moments of different orders of the in-degree, out-degree, and mutual degree distributions:

$$\langle G \rangle \sim N^{n-g_a-g_m} \langle K \rangle^{g_a} \langle M \rangle^{g_m} \prod_{j=1}^n \left\langle \binom{K_i}{k_j} \binom{R_i}{r_j} \binom{M_i}{m_j} \right\rangle_i, \quad (4)$$

where g_a and g_m are the number of single and mutual edges. The approximation assumes uncorrelated, sparse networks ($\langle K \rangle \ll N$). Both conditions are met by software maps [25]. These mean-field quantities can be used as a null model estimate of the number of motifs and, eventually, to detect stray, significant deviations from randomness. Since different motifs are found in different systems [5], they can actually allow us to identify the basic functional blocks for a given class of networks.

By exploring our collection of software graphs, we determined $\langle G \rangle$ for real nets (indicated as N_{real} in Fig. 2) and compared them to N_{rand} . Here software maps with a size $N > 10$ have been analyzed. Two groups have been chosen, involving medium-sized graphs ($N < 300$) and large graphs ($N > 300$). The previous set is compared with results from other networks involving computational tasks. Here previous results for both gene and neural networks are also shown for comparison (data from [5]). The reason for using biological networks as a reference system is twofold. First, the chosen systems are known to perform computational tasks (or can be described by means of an equivalent computational circuit). The second is that it has been conjectured that both natural and artificial networks might share some commonalities relating the mechanisms that shape their evolution [12]. Com-

mon features might reflect common functional traits, but also (as shown below) common rules of graph evolution with no special functional meaning.

In order to explore the question of how relevant the overall network structure is in conditioning the frequency of given subgraphs, we should consider the global structure of the network. The first approximation is to consider the degree of heterogeneity as provided by the distribution of links. Software systems have a well-defined scale-free indegree distribution

$$P_i(k) = \frac{\gamma_i - 1}{k_0^{1-\gamma_i}} (k + k_0)^{-\gamma_i}, \quad (5)$$

with $\gamma_i \sim 2$. A mean value $\langle \gamma_i \rangle = 2.09 \pm 0.05$ has been obtained by averaging over all the systems studied here. The distribution of scaling exponents is strongly peaked around $\gamma_i = 2$. The out-degree distribution $P_o(k)$ is much steeper and seems better described by a broad scale distribution, not far from the exponential limit. This is actually the opposite situation considered in [24] but is not difficult to show that it is essentially symmetric in the theoretical treatment.

For the regime considered here, it was shown that $\langle G \rangle$ follows a scaling law

$$\langle G \rangle \sim N^\alpha. \quad (6)$$

Specifically, for a given pair $\{n, g\}$ and a given scaling exponent, we have

$$\langle G \rangle \sim N^{n-g+s-\gamma_i+1}, \quad (7)$$

where s is the maximum in-degree for our case. This scaling is actually valid for $2 < \gamma_i < s+1$.

Four examples of the observed scaling laws are shown in Fig. 4 for different software motifs. Using Eq. (3) the expected number of times a given motif appears would scale as $\langle G \rangle \sim N^{s+1-\gamma_i}$ and using the scaling exponent $\gamma_i \approx 2.09 \pm 0.06$ a scaling law $\langle G \rangle \sim N^\alpha$ would be predicted for uncorrelated, sparse graphs. Here we use our set of systems¹ whose size will be indicated as n_i ($i=1, \dots, 83$) (number of nodes of each graph). For convenience, we order the systems by increasing size (i.e., $n_i < n_{i+1}$). If $G(n_i)$ is the number of times a given subgraph appears in the i th system (of size n_i), we should expect a scaling relation $G(n_i) \sim n_i^\alpha$. In order to reduce the noise level we will use the cumulative distribution

$$G_{cum}(N) = \sum_{n_i \leq N} G(n_i). \quad (8)$$

The cumulative distribution should scale as $G \sim N^{\alpha_c}$ with $\alpha_c \approx \alpha + 1$. As shown in Fig. 4, the predicted scaling is recovered from real data, thus indicating that the average trends are consistent with the expectation from random scale-free networks. It confirms the validity of the prediction of Itzkovitz *et al.*[24] and its agreement with a set of real networks. This agreement is an interesting result, particularly if

¹Although a total of 83 systems have been used, the presence of a specific subgraph is size dependent. Not all systems exhibit all subgraphs: for small software maps some subgraphs are absent.

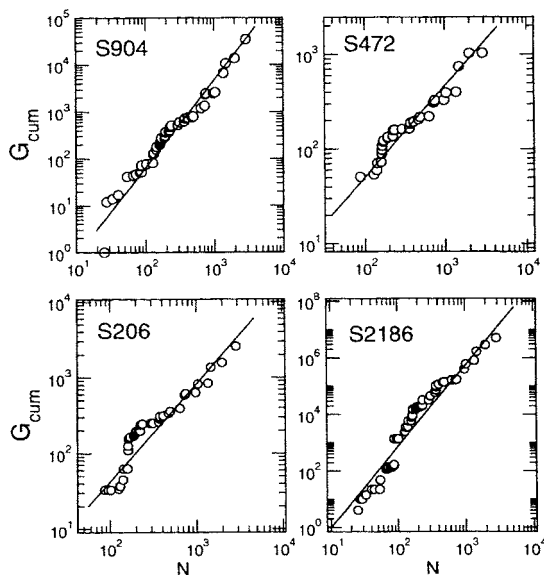


FIG. 4. Scaling in the number of appearances of a given motif against network size. Here four common motifs (each indicated) have been considered over the sample set [24]. Here we have S904 with $n=4$, $g=4$, and $s=2$; S472, with $n=4$, $g=5$, and $s=2$; S206, with $n=4$, $g=5$, and $s=2$; S2186, with $n=4$, $g=4$, and $s=3$. The predicted exponents (using the average scaling exponent for the in-degree distribution $\gamma_i \approx 2.1$) would be $\alpha(S904)=0.9$, $\alpha(S472)=\alpha(S206)=0.1$, and $\alpha(S2186)=2.1$, respectively. Using the cumulative number of graphs, G_{cum} (see text), we obtain $\alpha_c(S904)=1.86 \pm 0.16$, $\alpha_c(S472)=0.97 \pm 0.07$, $\alpha_c(S206)=1.18 \pm 0.17$, and $\alpha_c(S2186)=3.12 \pm 0.11$, in good agreement with the predicted values. The fit was made using least squares on a log-log scale.

we remember that this is a *designed* system to perform given functions. The fact that we obtain the scaling law expected for the random, scale-free graph reveals that the observed scaling in motif abundances are a consequence of top-down constraints derived from graph evolution.

IV. DUPLICATION-BASED EVOLUTION

The topology of software architecture emerges from designed evolution. On top of the process, there must be a basic building plan towards a final function or set of functions. The engineer foresees the outcome of its work. But there are a number of strong constraints no less important and operating through the software building process. On the one hand, modular structures are shaped through parallel paths of evolution. Different blocks will be involved in more specific subfunctions. On the other hand, increased complexity leads to conflicts between different subparts. This is reflected, for example, at the topological level: small software maps tend to display tree structure, whereas larger systems typically display much more complex patterns [16]. The common overall structure detected in software graphs in terms of the degree distribution (and other average properties) suggests that the final topological patterns might be strongly constrained.

We conjecture that the abundance of subgraphs in software networks relates to universal mechanisms of network

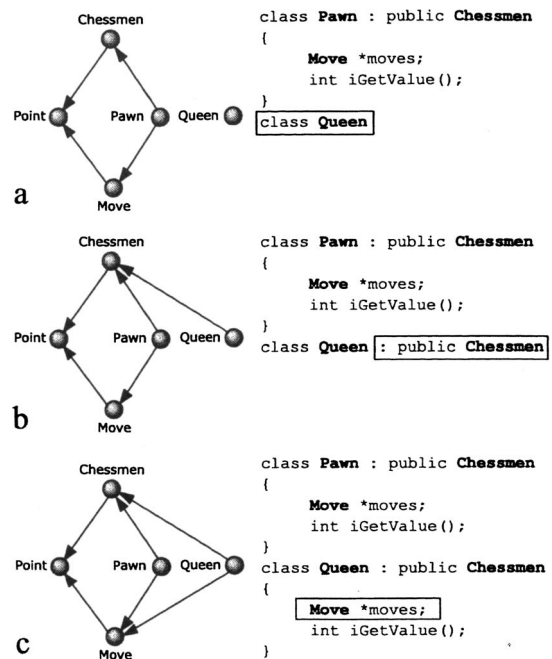


FIG. 5. From (a) to (c), an illustration of the duplication mechanism in software map evolution. Time flows from top to bottom. Here, a new module *queen* is introduced by cloning the links of the similar module *pawn*. Every stage displays the evolving C++ program (right) and its corresponding software network (left) reconstructed by the method described in Sec. III. New text is enclosed in a box. Note how duplication of links in the software map is parallel to duplication of code in the C++ program.

growth underlying their evolution. Real software maps tend to display motif generalizations or subgraphs having an structure comprising many replicas of the four motifs observed here. These structures are highly redundant. This suggests a very simple duplication-based mechanism of subgraph generation. New modules depend upon other modules in order to provide useful functionalities. And it seems reasonable to assume that similar modules will share a large number of module dependences. In a related software engineering study [26], structural similarities in C++ software at the module (class) level have been analyzed. They have found quantitative evidence of structural duplications. However, they did not provide any model explaining the origin of duplications.

Figure 5 shows a detailed example illustrating how top-down duplication works in software development. Imagine we want to add a new software module representing the queen, in the previous chess-playing program [see Fig. 2(a)]. First, we will add a new module declaration, which is conveniently named *queen* [see Fig. 5(a)]. Because a queen is a type of chessman, it seems reasonable to make this module depend upon the same modules referenced by similar modules, which in this case is the pawn. By using the pawn module and its neighborhood as a template, we add an inheritance relationship from the queen to the chessman [see Fig. 5(b)]. Duplication is completed with the addition of a collaboration relationship from queen to *move* [see Fig. 5(c)]. Comparison between final network [see Fig. 5(c)] with the initial network (see Fig. 2) reveals a new biparallel subgraph and twice the number of bifan subgraphs.

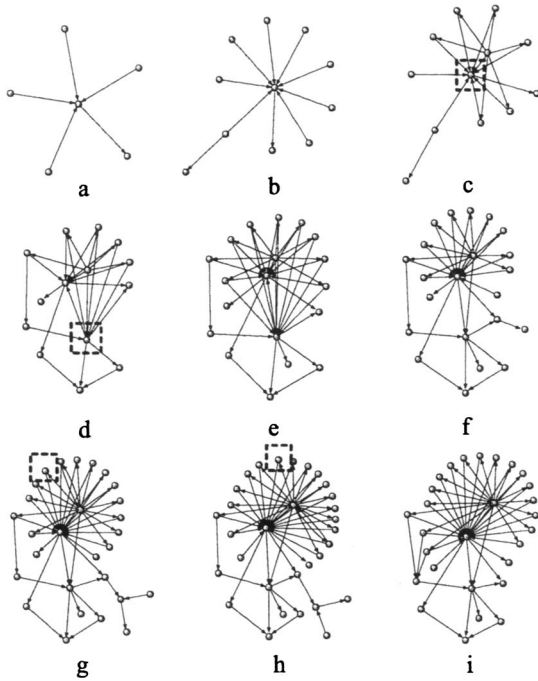


FIG. 6. A real instance of software network growth from a well-defined subsystem of Prorally [16] showing duplication. Evolution goes from top to bottom and left to right. Only the largest connected component is displayed here. The figure shows how the target hub in (c) has been duplicated in (d) (both nodes highlighted with a dotted box). Many duplicated nodes involve less connected targets [see (g) and (h)].

An example of this process taking place in real software development is shown in Fig. 6. Here a given subsystem inside a growing software graph is displayed at different development stages. Duplication of nodes seems to be at work, as well as further removal of many links associated with a given hub. From *c* to *d* a duplication of the hub involving many incoming links has taken place (together with some further node addition). From *d* through *i*, it is evident how a large number of new classes were added by copying the pattern of single nodes connected to two central hubs. There is extensive rewiring in some stages, such as in *f*, where the lower hub losses a large fraction of in-links. Moreover, there is also the addition of new connections between existing nodes (see *h* \rightarrow *i*). The whole sequence spans 1 year of development. The main observation from this example (which is a typical one) is that node duplication plus rewiring, particularly link removal, is widespread. This is also the case in the evolution of cellular networks [6].

Examples like the previous one suggest that duplication-divergence growth is the cause of the observed subgraph abundances in software maps. This hypothesis can be tested by comparing the distribution of subgraphs in real networks with those obtained with a stochastic model of network growth based on asymmetric duplication-divergence rules, previously described in [6]. First, an initial random (or backbone) network of $m_0 < N$ nodes is created. This random graph is generated by the addition of nodes with degree $k_0 = 2$, every link pointing to a random target node [27]. This backbone possesses a treelike structure (as occurs with software

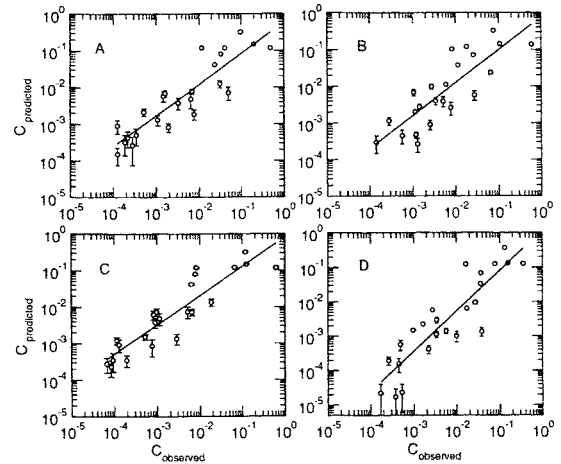


FIG. 7. Comparison of observed and predicted (from a duplication model) 4-motif concentrations for (a) Blender, (b) Filezilla, (c) GTK, and (d) Exult (here concentrations are rescaled by $\times 10^{-3}$). The exponents for the least-squares fit are (a) $\xi = 0.94 \pm 0.12$, (b) $\xi = 0.92 \pm 0.13$, (c) $\xi = 0.96 \pm 0.11$, and (d) $\xi = 1.14 \pm 0.12$, respectively.

maps at the beginning of their evolution). Starting from this backbone, we apply the following rules at each iteration of the model.

(i) *Duplication*. A randomly chosen target node v is cloned, and the new node w attaches to all the neighbors of the target node.

(ii) *Divergence*. For each pair of original and redundant links remove one of them with probability δ .

(iii) *Cross linking*. In addition, the target and new node are linked ($w \rightarrow v$) with probability β . This rule is important in order to generate triads or 3-subgraphs.

In spite of the simple set of rules implicit in the duplication model, the frequencies of subgraphs obtained from our *in silico* system are remarkably close to those seen in their real counterparts. In Fig. 7, we have compared the concentration of 4-subgraphs expressed in various software networks and the concentration of 4-subgraphs predicted with the duplication-based evolution model. These plots were obtained with the following method. We generate 400 graphs, 100 for each of four different software networks: Blender, Filezilla, GTK, and Exult [28]. Each synthetic graph has the same number of links L and number of nodes, N , as measured in the corresponding software map and no further constraints are imposed. The parameter space is sampled uniformly. Once the synthetic networks are obtained, we perform a 4-subgraph census by counting the number of appearances of each 4-subgraph Ω_i in the model and in the synthetic network. Notice that we do not test for statistical significance (as in the motif analysis). Instead, our comparison test is based solely on raw subgraph counts. In order to compare the two systems, the raw number of subgraphs of size 4 is computed and the concentration C of subgraphs evaluated. Here, the concentration is simply the number of appearances of the 4-subgraph over the total number of 4-subgraphs found.

In Fig. 7, each point represents the pair ($C_{observed}, C_{predicted}$) of observed and predicted concentrations for given 4-subgraph Ω_i . Specifically, we display the set of

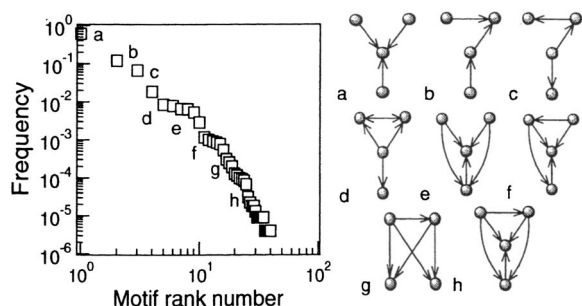


FIG. 8. Frequency-rank distribution of network subgraphs in a software network (\square). Here the most frequent subgraph has rank 1, the second has rank 2, etc. The frequency $P(r)$ of a subgraph with rank r decays rapidly with subgraph rank. An interesting feature is that most common subgraphs are sparser than less common ones, which are more dense.

pairs and the power law fit $C_{pred} \sim C_{obs}^\xi$. Despite fluctuations, the simple duplication model presented here predicts reasonably well the concentration of common software network motifs: the value of the exponent ξ is reasonably close to 1 in all cases. This is remarkable, given the oversimplification considered here and given the limited constraints imposed to the selected model graphs to be compared with the real ones. The error bars grow as less common subgraphs are used. If we restrict ourselves to $C > 10^{-3}$, the exponent ξ becomes much closer to 1. Specifically, we obtain now (a) $\xi = 0.96 \pm 0.11$, (b) $\xi = 0.97 \pm 0.10$, (c) $\xi = 0.98 \pm 0.12$, and (d) $\xi = 1.06 \pm 0.18$, respectively. Consistently with previous work [8], less common subgraphs are typically more dense (have more links). In Fig. 8 an example of this correlation is shown for Exult. Using a frequency-rank plot of 4-subgraphs, we can see that subgraphs with high frequencies have few links whereas higher ranks (small frequencies) are associated with dense subgraphs.

V. DISCUSSION

In this paper we have analyzed the statistical patterns of network motifs in a large set of software diagrams. Software maps have been previously shown to be scale free and display small-world behavior [16,18,21] but no previous analyses focused on the small-scale architecture. The main goal of our study is to explore the relevance of graph evolution in relation with true functionality. Our study actually suggests that dynamical rules, with little relation to underlying functional constraints, largely determine the frequency of motifs in software graphs.

By using recent theoretical and numerical methods to measure and characterize network motifs, we have found the following.

- (i) A number of network motifs are obtained, the most common being shared with other (natural) systems involving computational traits, such as genetic and neural networks.
- (ii) The number of appearances of a given network motif scales as $\langle G \rangle \sim N^{m-g+s-\gamma_r+1}$, in agreement with previous calculations for random graphs with scale-free degree distributions. This result is supported by previous observations of the uncorrelated character of software maps.

(iii) Evidence from software evolution suggests that duplication and rewiring, as occurs with some cellular networks, might play a key role in shaping software maps. Using a previous model of network growth by duplication and diversification, it has been shown that it fits rather well the frequencies of the appearance of network motifs.

Previous studies have proposed the idea that network motifs seem to define the minimal, meaningful building blocks of network complexity. Perhaps not surprisingly we often find them as the basic structures associated with specific functional traits, from computation to pattern formation. The former is exemplified by feed-forward loops, a three-element motif found in genetic regulatory systems [22,29]. The latter is actually a particularly relevant example. However, since the statistical distribution of network motifs involves dealing with large numbers of different subgraphs, the question of how motifs *in general* might reflect functional traits requires the formulation of appropriate null models of graph evolution. Such models must ignore any functional trait in order to test the possibility that the global properties of network structure (such as graph heterogeneity) might strongly influence what we should expect.

The model chosen here has been a duplication-rewiring one [6]. These models have been shown to generate heterogeneous graphs with many properties close to relevant biological systems such as protein-protein interaction maps. Network heterogeneity is largely due to effective preferential attachment. Additionally, the rules of duplication strongly bias the types of motifs to be formed towards some special subsets. The final consequence is that the patterns of network motifs generated by the duplication model might be able to explain (in statistical terms) the observed abundances of motifs, with no further requirement of functional constraints. The fact that biological systems, also involved in performing computations, have common motifs might support this view. Although sharing common motifs seems to call for common functionalities, it is important to remember that biological structures are largely generated through tinkering [10,11]. Protein interaction networks grow by gene duplication, and neural networks also experience increases of cell numbers together with wide synaptic changes. Perhaps the common traits are a by-product of the common tinkered evolution based on extensive reuse and copy of available structures.

One final comment concerns with the common subgraphs also shared by digital circuits. They are not obtained, strictly speaking, through a process of duplication and rewiring. Although the way complex circuits are built does include some amount of reuse,² considerations involving low cost in links are of fundamental importance. In spite of such constraints, it has been shown that electronic circuits have small-world structure and are also highly heterogeneous [30]. Previous work seems to indicate that optimal design towards efficient

²As circuit complexity increases (both in terms of number of components and computational tasks) it becomes more difficult to design from scratch choosing sets of small gates and building optimal, low-cost circuits. Predefined gates involving well-known (and sometimes complex) input-output functions are widely used and assembled together. In that sense, some amount of re-use is at work.

communication at low cost can generate scale-free, heterogeneous architectures [31,32]. Such result suggests again that network heterogeneity might pervade motif abundances [33].

ACKNOWLEDGMENTS

We thank Maggie Fitzgerald, Frankie Dunn, and Eddie Scrap for useful input. We also thank Shalev Itzkovitz for a

careful reading and comments on an earlier version of the manuscript. The analysis of network motifs has been done using available free software from Uri Alon's Lab (see <http://www.weizmann.ac.il/mcb/UriAlon/index.html>). This work has been supported by Grant No. FIS2004-05422 and by the EU within the 6th Framework Programme under Contract No. 001907, "Dynamically Evolving, Large Scale Information Systems" (DELIS).

-
- [1] S. N. Dorogovtsev and J. F. F. Mendes, *Evolution of Networks: From Biological Nets to the Internet and WWW* (Oxford University Press, New York, 2003).
- [2] R. Albert and A. L. Barabási, *Rev. Mod. Phys.* **74**, 47 (2002).
- [3] M. E. J. Newman, *SIAM Rev.* **45**, 167 (2003).
- [4] S. Bornholdt and G. Schuster, *Handbook of Graphs and Networks*, edited by (Wiley-VCH, Berlin, 2002).
- [5] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, *Science* **298**, 824 (2002).
- [6] R. V. Solé, R. Pastor-Satorras, E. D. Smith, and T. Kepler, *Adv. Complex Syst.* **5**, 43 (2002); A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani, *Complexus* **1**, 38 (2003); R. Pastor-Satorras, E. D. Smith, and R. V. Solé, *J. Theor. Biol.* **222**, 199 (2003); J. Kim, P. L. Krapivsky, B. Kahng, and S. Redner, *Phys. Rev. E* **66**, 055101 (2002); K.-I. Goh, B. Kahng, and D. Kim, e-print q-bio.MN/0312009, v2; W. Banzhaf and P. Dwight Kuo, *J. Biol. Phys. Chem.* **4**, 85 (2004).
- [7] R. V. Solé and P. Fernandez (unpublished). See also R. Guimera, M. Sales-Pardo, and L. A. N. Amaral, *Phys. Rev. E* **70**, 025101 (2004).
- [8] A. Vázquez *et al.* *Proc. Natl. Acad. Sci. U.S.A.* **101**, 1794 (2004).
- [9] B. Hayes, *Am. Sci.* **89**, 204 (2001).
- [10] F. Jacob, *Science* **196**, 1161 (1976).
- [11] D. Duboule and A. S. Wilkins, *Trends Genet.* **14**, 54 (1998).
- [12] R. V. Solé, R. Ferrer, J. M. Montoya, and S. Valverde, *Complexity* **8**, 20 (2002).
- [13] U. Alon, *Science* **301**, 1866 (2003).
- [14] A. V. Aho, *Science* **303**, 27 (2004).
- [15] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools* (Addison-Wesley Longman, Boston, 1986).
- [16] S. Valverde, R. Ferrer-Cancho, and R. V. Solé, *Europhys. Lett.* **60**, 512 (2002).
- [17] C. F. Kemerer and S. Slaughter, *IEEE Trans. Software Eng.* **25**, 493 (1999).
- [18] S. Valverde and R. V. Solé (unpublished).
- [19] B. Stroustrup, *The C++ Programming Language* (Addison-Wesley, Reading, MA, 1986).
- [20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software* (Addison-Wesley, New York, 1994).
- [21] C. R. Myers, *Phys. Rev. E* **68**, 046116 (2003).
- [22] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, *Nat. Genet.* **31**, 64 (2002).
- [23] J. Lakos, *Large Scale C++ Software Design* (Addison-Wesley, New York, 1996).
- [24] S. Itzkovitz, R. Milo, N. Kashtan, G. Ziv, and U. Alon, *Phys. Rev. E* **68**, 026127 (2003).
- [25] All software maps analyzed here (and others studied by other authors) have been shown to be sparse. Correlations have been also analyzed in R. V. Solé and S. Valverde, in *Complex Networks*, edited by E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, *Lecture Notes in Physics* (Springer, Berlin, 2004), pp. 169–190. Using statistical measures derived from information theory, it was shown that software maps are considerably uncorrelated.
- [26] F. Fioravanti, G. Migliarese, and P. Nesi, in *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, IEEE, May 12–19, Toronto, 2001, edited by Hausi A. Müller (IEEE, New York, 2001).
- [27] D. S. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. J. Newman, and S. H. Strogatz, *Phys. Rev. E* **64**, 041902 (2001).
- [28] The source code is available at the following web sites: <http://www.blender.org> (Blender), <http://filezilla.sourceforge.net> (Filezilla), <http://www.gtk.org> (GTK), and <http://exult.sourceforge.net> (Exult).
- [29] S. Mangan and U. Alon, *Proc. Natl. Acad. Sci. U.S.A.* **100**, 11980 (2003).
- [30] R. Ferrer, C. Janssen, and R. V. Solé, *Phys. Rev. E* **64**, 046119 (2001).
- [31] R. Ferrer and R. V. Solé, in *Statistical Physics of Complex Networks*, edited by R. Pastor-Satorras, M. Rubi, and A. Diaz-Guilera, *Lecture Notes in Physics* (Springer, Berlin, 2003), pp. 114–125.
- [32] R. V. Solé and S. Valverde, in *Complex Networks*, edited by E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, *Lecture Notes in Physics* (Springer, Berlin, 2004), pp. 169–190.
- [33] H. B. Fraser, A. E. Hirsch, L. M. Steinmetz, C. Scharfe, and M. W. Feldman, *Science* **296**, 750 (2002).