GPU-Based Parallel Search of Relevant Variable Sets in Complex Systems

Emilio Vicari[1], Michele Amoretti[1], Laura Sani[1], Monica Mordonini[1], Riccardo Pecori[1,4], Andrea Roli[2], Marco Villani[3], Stefano Cagnoni[1](✉), and Roberto Serra[3]

[1] Dipartimento di Ingegneria ed Architettura, Università di Parma, Parma, Italy
stefano.cagnoni@unipr.it
[2] Dip. di Informatica, Scienza e Ingegneria, Università di Bologna - Sede di Cesena, Cesena, Italy
[3] Dip. Scienze Fisiche, Informatiche e Matematiche, Università di Modena e Reggio Emilia, Modena, Italy
[4] SMARTest Research Centre, Università eCAMPUS, Novedrate, CO, Italy

Abstract. Various methods have been proposed to identify emergent dynamical structures in complex systems. In this paper, we focus on the Dynamical Cluster Index (DCI), a measure based on information theory which allows one to detect relevant sets, i.e. sets of variables that behave in a coherent and coordinated way while loosely interacting with the rest of the system. The method associates a score to each subset of system variables; therefore, for a thorough analysis of the system, it requires an exhaustive enumeration of all possible subsets. For large systems, the curse of dimensionality makes the problem solvable only using metaheuristics. Even within such approaches, however, DCI computation has to be performed for a huge number of times; thus, an efficient implementation becomes a mandatory requirement. Considering that a candidate relevant set's DCI can be computed independently of the others, we propose a GPU-based massively parallel implementation of DCI computation. We describe the algorithm's structure and validate it by assessing the speedup in comparison with a single-thread sequential CPU implementation when analyzing a set of dynamical systems of different sizes.

Keywords: GPU-based parallel programming · Complex systems · Relevant sets

1 Introduction

The behavior of a complex system can be described by identifying emergent dynamical structures within it, i.e., subsets of variables whose members tightly interact with (depend on) one another, as well as hierarchically, by identifying higher-level interactions that occur between such sets.

The study of complex systems is related to the identification of emergent properties of systems whose components are usually well-known and defined in terms of state variables. To describe the organization of complex systems several measures of complexity have been proposed, many of which based on information theory (as, for instance, in [4,6]).

Many different systems can be described effectively in terms of coordinated dynamical behavior of groups of elements; for example, relevant examples in the domain of neuroscience can be found in [8,9].

Tononi et al. [10], and later other authors (Sporns et al. [9], Villani et al. [12]) introduced a method to identify relevant structures in complex systems. Based on a data-set including samples of the system status at different times, one can associate each possible subset of variables with an index T_c. Such an index quantifies how much its behavior deviates from the behavior of a reference (homogeneous) system, in which the variables have, individually, the same distribution as in the data-set, but are homogeneously correlated. Therefore, the higher its T_c, the higher the degree of correlation/interaction between the variables in a subset. The subsets characterized by high T_c values are referred to as Candidate Relevant Sets (CRSs), the properly called Relevant Subsets (RSs) being candidates that do not include (or are not included in) other candidate sets with higher T_c values [12].

For a complete description of the dynamical system, T_c must be computed for each possible set, which becomes unfeasible as the dimension of the system increases. Subsets of variables describing high-dimensional systems can therefore be identified by using a metaheuristic which smartly explores the search space [7]. Even in this case, T_c computation must be repeated hundreds of thousands to millions of times. An efficient implementation of such a function is therefore definitely necessary. Considering that the computation of T_c for each candidate RS is independent of the others, using GPU-based parallel code seems to be the most efficient way of computing the index.

We have developed a set of CUDA C[1] kernels that provide a fine-grained parallel implementation of the main building blocks needed to compute the T_c index, upon which smart and efficient search algorithms can be designed.

The parallel functions were developed to accomplish three different goals in our study:

1. Speeding up an exhaustive sequential search by computing the T_c values of several candidate RSs in parallel;
2. Providing a computationally-efficient objective function for a metaheuristic that searches for the RSs of large dynamical systems for which an exhaustive search is impractical;
3. Making it possible to explore more complex systems and detect possible hierarchical dependencies between RSs.

In the next section, we briefly introduce the basics of the method for which we have developed the CUDA kernels. Then we analyze the computational problem, identifying the algorithm blocks that are most amenable to parallelization, and describe their GPU-based implementation. We conclude our paper by reporting

[1] https://developer.nvidia.com.

the results of the tests in which we compare the performance of our parallel code with respect to a standard single-CPU sequential implementation. Finally, in the last section, we foresee possible future steps in our research that we expect the development of the parallel code to make feasible.

2 Method

In this section we succinctly illustrate the procedure for computing the T_c. The interested reader can find more details in [3,12].

Let the system under exam be modeled by means of a set U of N variables, which assume finite and discrete values. The *cluster index* of a subset S of variables in U, $S \subset U$, as defined by Tononi et al. [10], estimates the ratio between the amount of information integration among the variables in S and the amount of integration between S and U. These quantities depend on Shannon's entropy of both the single elements and the sets of elements in U.

The entropy of an element x_i is defined as:

$$H(x_i) = -\sum_{v \in V_i} p(v) \;\; log\, p(v) \tag{1}$$

where V_i is the set of the possible values of x_i and $p(v)$ the probability of occurrence of symbol v. The entropy of a pair of elements x_i and x_j is defined by means of their joint probabilities:

$$H(x_i, x_j) = -\sum_{v \in V_i} \sum_{w \in V_j} p(v, w) \;\; log\, p(v, w) \tag{2}$$

Equation 2 can be extended to sets of k elements considering the probability of occurrence of vectors of k values. This approach deals with observational data, therefore probabilities are estimated by means of relative frequencies.

The cluster index $C(S)$ of a set S of k elements is defined as the ratio between the integration $I(S)$ of S and the mutual information between S and the rest of the system $U - S$.

The integration of subset S is defined as:

$$I(S) = \sum_{x \in S} H(x) - H(S) \tag{3}$$

$I(S)$ represents the deviation from statistical independence of the k elements in S. The mutual information $M(S; U - S)$ is defined as:

$$M(S; U - S) \equiv H(S) + H(S|U - S) = H(S) + H(U - S) - H(S, U - S) \tag{4}$$

where $H(A|B)$ is the conditional entropy and $H(A, B)$ the joint entropy. Finally, the cluster index $C(S)$ is defined as:

$$C(S) = \frac{I(S)}{M(S; U - S)} \tag{5}$$

Since C is defined as a ratio, it is undefined in all those cases where $M(S; U - S)$ vanishes. In this case, the subset S is statistically independent from the rest of the system and needs to be analyzed separately. As $C(S)$ scales with the size of S, cluster index values of systems of different size need to be normalized. To this aim, a reference system is defined, i.e., the homogeneous system U_h, randomly generated according to the probability distribution of each state of the original system U. Then, for each subsystem size of U_h the average integration $\langle I_h \rangle$ and the average mutual information $\langle M_h \rangle$ are computed. Finally, the cluster index value of S is normalized by means of an appropriate normalization constant:

$$C'(S) = \frac{I(S)}{\langle I_h \rangle} / \frac{M(S; U - S)}{\langle M_h \rangle} \tag{6}$$

Furthermore, to assess the significance of the differences observed in the cluster index values, a statistical index T_c is computed:

$$T_c(S) = \frac{C'(S) - \langle C'_h \rangle}{\sigma(C'_h)} \tag{7}$$

where $\langle C'_h \rangle$ and $\sigma(C'_h)$ are the average and the standard deviation of the population of normalized cluster indices with the same size as S from the homogeneous system.

We emphasize that the indices in 5–7 are defined without any reference to a particular type of system. In their original papers, Edelman and Tononi considered the fluctuations of a neural system around a stationary state. In our approach, this measure is applied to time series of data generated by a dynamical model. In general, these data lack the stationary properties of fluctuations around a fixed point. Moreover, depending upon the case at hand, either transients from arbitrary initial states to a final attractor, or collections of attractor states can be considered, as well as responses to perturbations of attractor states. In all these cases we will use Eq. 5, that will therefore be called the Dynamical Cluster Index (DCI), as it aims at detecting subsets of variables that are relevant to the system's dynamics.

The search for relevant subsets of variables of a dynamical system by means of the DCI requires first the collection of observations of the variables' values at different times. In order to find such sets, in principle, all the possible subsets of system variables should be considered and their DCI computed. In practice, this procedure is feasible only for small-size subsystems in a reasonable amount of time. This paper presents a parallel DCI computation algorithm developed to address this issue.

3 Parallel Algorithm

When large systems are analyzed, the sequential implementation soon reaches unrealistic requirements for computation resources, because the number of possible CRSs increases exponentially with the number of system variables. A possible solution to mitigate this problem consists of a parallel implementation of the main building blocks of the code needed to compute the T_c index.

The GPU is specialized for compute-intensive and highly parallel computation and is capable of addressing highly arithmetically-intense problems that can be expressed as data-parallel computations. The computation of T_c for each CRS is independent of the others, thus a GPU-based parallel code seems to be the most efficient way of computing such an index. That is why we have developed CUDA C code for searching RSs in complex systems.

In order to understand how our code is organized we should consider that the exhaustive computation of the T_c index for all the CRSs of a dynamical system can be divided into the following steps:

1. Computation of the probability distribution function for each system variable;
2. Generation of the homogeneous system;
3. DCI computation for each subset of variables of the homogeneous system;
4. T_c computation for each CRS of the system variables.

From the point of view of the implementation:

– Each sample is stored in a memory area including S adjacent unsigned ints large enough to contain the N_{bit} bits needed to represent the N variables of the system. For example, if we consider a system consisting of N binary variables, then $N_{bit} = N$ and $S = \lceil N_{bit}/\mathsf{sizeof(unsigned\ int)}\rceil$. If M is the number of samples, then the system data can be stored in an array of $M \cdot S$ unsigned integers.
– Each CRS is represented as a bitmask of N_{bit} bits, where the i^{th} bit is set to 1 if the i^{th} variable is contained in the CRS.

3.1 Computation of the Probability Distribution Function

Each variable of the system is examined individually in order to compute its probability distribution function. In case of binary variables, for example, the distribution of the i^{th} variable is defined by the frequency of the values 0 and 1 (f_{i0} and f_{i1}). The frequency information thus obtained will be used for the generation of the homogeneous system as described in Sect. 3.2.

The frequencies of occurrence of the variables are also used to compute the entropy of each variable, necessary for the computation of the DCI as described in Sect. 3.3. If we consider a binary variable, then the entropy is defined by:

$$H_i = -f_{i0} \cdot log_2 f_{i0} - f_{i1} \cdot log_2 f_{i1}$$

3.2 Homogeneous System Generation

The homogeneous system (HS) is generated from N random variables, homogeneously correlated with one another, having the same probability distribution as the corresponding variables of the dynamical system to be studied.

We obtain M samples by assigning to the i^{th} variable, for each sample, a randomly generated value from the previously estimated distribution.

In case of a system described by binary variables, the i^{th} variable of the homogeneous system, for each sample, will be 0 with probability f_{i0} and 1 with probability f_{i1}. In this way, the HS meets the homogeneity requirement while, at the same time, it maintains a relationship with the dynamical system under consideration.

3.3 DCI Computation on the Homogeneous System

All possible CRS sizes (or classes) from 2 to $N-1$ are analyzed in order to compute, for each of them, the mean value and the standard deviation of the DCI. If the considered size is r, then the CRSs to be examined are selected by scanning all possible permutations of an N-bit string containing r bits set to 1 and $N-r$ bits set to 0.

The selected CRSs are grouped into grids of T threads each, where each thread is responsible for computing the DCI of one CRS. We have $T = N_B N_T$, where N_B is the number of blocks per grid and N_T is the number of threads per block. Each CRS of a certain size is coupled with its complementary cluster, whose entropy is necessary for computing the mutual information. In other words, each grid is composed of $T/2$ complementary CRS pairs. By synchronizing the execution of parallel threads in order for the entropy of one CRS to be available at the right time, it is possible to compute the statistics of classes r and $N-r$ at the same time, halving the computation time with respect to the original algorithm.

The outputs of this processing step are the mean value and the standard deviation of the DCI for each CRS class of the homogeneous system, which are necessary for computing the T_c index.

3.4 T_c Computation

In the following, we describe the main modules involved in the computation of the T_c index, as shown in Fig. 1.

DCI Module: The computation of the *DCI* of a CRS consists of three phases:

1. *Creation of the frequency histogram*; the number of occurrences of each value of the CRS is counted; the result is a list of value/number of occurrence pairs;
2. *Entropy computation*; based on the list obtained in the previous phase, the entropy is computed according to Eq. 1;
3. *Computation of the final output*; the threads of the block are synchronized to make the complementary entropy available to each CRS. This enables the computation of the mutual information, which, along with the integration, is used to compute the DCI.

Calculating the frequency histogram is, computationally, the heaviest step. In particular we need:

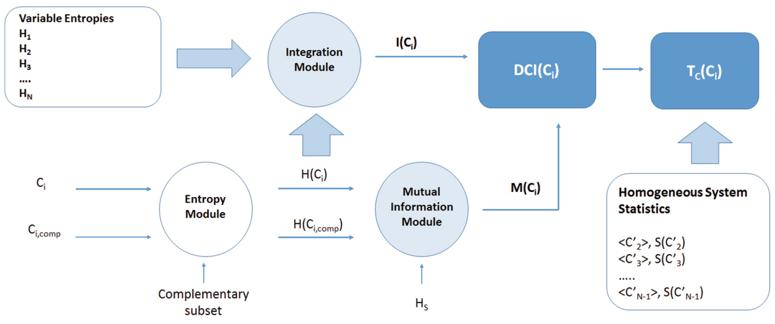

Fig. 1. T_c computation.

– Processing resources to extract the value of the variables in the CRS from each sample of the system;
– Memory to store the frequency histogram of the CRS.

To obtain a good trade-off between performance and memory usage, we generate a hash map, pre-allocated for each thread to be managed by the GPU kernel that computes the histogram (Sect. 3.5).

***T_c* Module:** The module that computes the T_c statistical index is a simple extension of the one which computes the DCI, that takes advantage of the above-mentioned organization into coupled threads. Particularly, in this case, the CRSs of each class, ranging from 2 to N/2, are placed aside their complementary CRSs and are inserted, as for the DCI computation for the homogeneous system, in parallel computation batches, each composed of T threads. Once the DCI has been computed, it is sufficient to normalize it according to the statistics (expected value and standard deviation) of the homogeneous system that were obtained earlier (see Sect. 3.3). As the T_c module simply extends the DCI module, both call the same CUDA kernel to perform their computations; the calls differ only in the input parameters.

Once the T_c indices of all the CRSs of the system are obtained, they are compared to select the CRSs having the highest index values.

3.5 Resource Occupation and Scalability

If N is the number of variables that compose the system, the total number of possible CRSs is 2^{N-1}. Thus, the computational complexity of the problem is $O(2^N)$. Parallelizing the computation allows one to obtain a relevant reduction of the execution time. However, this is still not enough to perform an exhaustive search on systems characterized by a large number of variables.

Different considerations can be made regarding memory occupation. Our implementation is based on a simple fact: it is not possible for a CRS to assume a number of configurations that is higher than the number of available samples M, which is usually much lower than the total number of possible CRS configurations (i.e., $M \ll 2^N$). Thus, for each CRS it is possible to pre-allocate a hash table with maximum size M. For this reason, the device memory that is necessary to contain the hash tables of a grid of threads is directly proportional to three independent variables, namely:

- T: number of threads per grid;
- N_{bit}: number of bits needed to store a sample;
- M: number of samples.

Accordingly, the memory occupation increases linearly with the problem size. A good estimation of the device memory needed is:

$$MEM_{TOT} = M \cdot T \cdot (S + 2) \cdot \mathsf{sizeof(unsigned\ int)} \tag{8}$$

where S is the number of unsigned int that are necessary to store N_{bit} bits. On a device provided with 2 GB of memory, it is possible, for example, to launch 1024 parallel threads and compute the T_c of the same number of CRSs from a system characterized by 1000 binary variables, with $M = 10000$ available samples (in this case, $MEM_{TOT} \simeq 1.4$ GB).

These considerations show that, to analyze large systems, the exponential dependence on the problem size makes an exhaustive search computationally unfeasible. However, an approach based on a metaheuristic would definitely be, as the device memory occupation scales linearly with the problem size.

4 Experimental Results

In this section we illustrate the experimental results we have obtained on four different dynamical systems. The algorithm was evaluated on both artificial and biological systems.

The first case study (referred to as LF) is described by 10 variables and consists of three independent groups, each of which replicates a simple leader-followers dynamic. The model abstracts situations where agents modify their opinion agreeing with (or contrasting the) opinion of other specific agents, called leaders. The system is simply composed of a vector of 10 binary variables $x_1, x_2, ..., x_{10}$ that represent, for example, the positive or negative opinion of 10 agents about a given proposal. The model generates a series of 10 binary vectors (each vector representing an observation of the system) according to the following rules:

- variables are divided into three groups, $G1 = [x_1, ..., x_3]$, $G2 = [x_4, ..., x_6]$ and $G3 = [x_7, ..., x_{10}]$;
- x_1 is a leader; at each step its value is a random value in $\{0, 1\}$;
- the values of the followers x_2 and x_3 are set as a copy of x_1 with probability $1 - p_{noise}$ and randomly with probability p_{noise};
- x_4 and x_7 are "second order" leaders; in each step their values are randomly assigned in $\{0, 1\}$ with probability $1 - p_{copy}$; otherwise x_4 is a copy of x_1 and x_7 is a copy of x_4;

– the values of the followers x_5 and x_6 are set as a copy of x_4 with probability $1 - p_{noise}$ and randomly with probability p_{noise};
– the values of the followers x_8, x_9 and x_{10} are set as a copy of x_7 with probability $1 - p_{noise}$ and randomly with probability p_{noise}.

It is therefore possible to tune the integration among elements in $G1$, $G2$ and $G3$ and the mutual information between $G1$ and $G2$, and between $G2$ and $G3$ by changing p_{noise} and p_{copy} [2,12].

The second and third cases model simplified gene regulatory networks. In particular, the second case study (referred to as AT) models the gene regulatory network shaping the developmental process of Arabidopsis Thaliana; although the whole network is largely unknown, a certain subsystem has been identified as responsible for the floral organ specification. The network is modeled by means of a Boolean network described in [1], having 15 nodes and 10 different attractors (all fixed points): in order to perform an analysis we built a data series containing a number of repetitions of these attractors proportional to the size of their basins of attraction.

The third case (referred to as TH) features 23 Boolean variables, used in [5] to model the regulatory network controlling the T-helper cell differentiation; also in this case we built a data series containing a number of repetitions of the Boolean system attractors proportional to the size of their basins of attraction. We will not discuss about the adequacy of these simplified models, but we will take them for granted and apply our method to test whether it can discover significant MDSs (Mesolevel Dynamical Structures).

The fourth case study is a deterministic simulation of a catalytic chemical system (Catalytic Reaction System - CRS - in the following), characterized by 26 variables, in which there are two distinct reaction pathways: a linear chain and an autocatalytic circle. The reactions happen in an open well-stirred chemostat (CSTR) with a constant influx of feed molecules and a continuous outgoing flux of all the molecular species proportional to their concentration. The dynamics of the system is described adopting a deterministic approach whereby the reaction scheme is translated into a set of ordinary differential equations integrated by means of a Euler method with step-size control. The asymptotic state of this system consists of constant concentrations. In order to apply our analysis, however, one needs to observe the feedbacks in action: thus, we perturbed the concentration of some molecules in order to trigger a response (i.e., a series of changes) in the concentration of (some) other species. The perturbations consisted of temporarily setting to zero the concentration of some species after the system reached its stationary state. To analyze the system response we used a three-level coding where, for each species, the digit ‘0’, ‘1’ and ‘2’ stand respectively for “decreasing concentration”, “no change” and “increasing concentration” (Fig. 2) [11,12].

The four cases present different dynamics and representations: in particular, the first test case consists of a binary time series, whereas the second and third cases are the juxtaposition of the binary states of several different attractors, and the fourth case is the encoding of a continuously perturbed situation into a

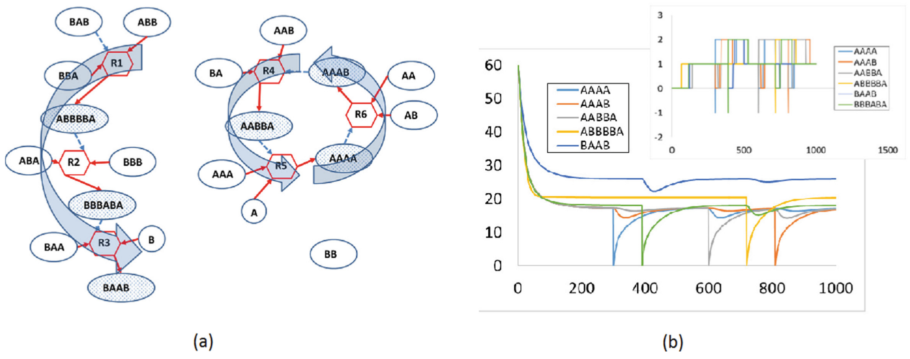

Fig. 2. (a) The reaction scheme of the Catalytic Reaction System (CRS): white ellipses represent the chemicals injected in the incoming flux, meshed ellipses represent the chemicals produced inside the CSTR vessel, hexagons represent the reactions; continuous arrows represent the consumptions/productions and dashed arrows represent the catalytic activities. Chemical BB does not participate in any reaction, and it is used as reference. The six reactions are arranged into two independent groups: a linear chain and an autocatalytic circle. (b) A time series of the six produced chemicals and the corresponding three-level encoding.

three-level representation. The method we implemented on GPU is able to find the correct relevant sets in all situations (some of them being discussed in details in [11,12]): in this paper, however, we focus our interest on the performance of the sequential and parallel algorithms.

The parallel algorithm (PA) has been evaluated in terms of correctness and efficiency (speedup), compared to the sequential algorithm (SA). To this purpose, we have used a Linux server provided with CPU Intel(R) Xeon(R) 2.10 GHz, 64 GB of RAM and a GPU NVIDIA GeForce GTX 1070. We have executed 10 independent runs for each example, using different random seeds when generating the homogeneous system.

Table 1 summarizes the algorithms' performance in relation to the system size (expressed as number of variables) and to the number of samples.

In all these case studies the results are correct: they are equal to the ones obtained by the sequential implementation, but they have been computed in a

Table 1. Performance summary of the sequential (SA) and parallel (PA) algorithms

System	#Variables	#Samples	Time (SA)	Time (PA)	Speedup
LF	10	500	2.15 s	11 ms	195.5
AT	15	5000	861 s	0.23 s	3743.5
TH	23	5000	60 h[a]	27.5 s	7854.5
CRS	26	751	20 d[a]	245 s	7053.1

[a]Estimated

significantly shorter time. Using our parallel implementation we can now exhaustively analyze systems of up to 35 variables in less than 24 h.

The speedup with respect to the sequential CPU implementation is very relevant.

5 Conclusion and Final Remarks

In this paper we have presented a fine-grained parallel implementation of the main building blocks needed to compute the T_c index. In summary, the most relevant choices, aiming at algorithm efficiency, are:

- Subgroup-wise parallelization (as opposed to a possible system data-wise parallelization);
- "Smart" allocation of threads/data (like using a hashmap for each thread, implemented on the graphics device).

These choices produce an algorithm which obtains a large speedup, but they are a little more critical as concerns memory allocation.

In the benchmarks we took into consideration, the algorithm obtained a dramatic speedup with respect to the sequential implementation, allowing us to detect RSs in dynamical systems of much larger size than previously possible.

When large systems are analyzed, the increasing number of CRSs makes it impossible to compute the T_c index for every possible subset, even using massively parallel hardware such as GPUs, so we need to design efficient strategies to quickly identify the most promising subsets, limiting the extension of the search.

Considering multi-GPU implementations, the structure of the parallel algorithm is such that the computation of each T_c index is totally independent of the others, which suggests that the number of T_c computations scales almost perfectly linearly with the number of GPUs.

Smart and efficient search algorithms can be easily designed upon our parallel implementation. For example, in [7], we proposed a metaheuristic based on a genetic algorithm that draws the search towards the basins of attraction of the main local maxima in the search space, along with a local search that improves the results by exploring those regions more finely and extensively. Such a metaheuristic computes the fitness function using the GPU-based implementation of the T_c computation described in this paper. The speedups achieved by our parallel implementation of the metaheuristic made it possible for us to analyze systems consisting of up to 137 variables in a reasonable time. Using an exhaustive approach based on a sequential implementation, the same time would have allowed us to analyze only very simple and rather uninteresting systems.

References

1. Chaos, A., Aldana, M., Espinosa-Soto, C., de León, B.G.P., Arroyo, A.G., Alvarez-Buylla, E.R.: From genes to flower patterns and evolution: dynamic models of gene regulatory networks. J. Plant Growth Regul. **25**, 278–289 (2006)

2. Filisetti, A., Villani, M., Roli, A., Fiorucci, M., Poli, I., Serra, R.: On some properties of information theoretical measures for the study of complex systems in advances in artificial life and evolutionary computation. Commun. Comput. Inf. Sci. **445**, 140–150 (2014)
3. Filisetti, A., Villani, M., Roli, A., Fiorucci, M., Serra, R.: Exploring the organisation of complex systems through the dynamical interactions among their relevant subsets. In: Andrews, P., et al. (ed.) Proceedings of the European Conference on Artificial Life 2015 (ECAL 2015), pp. 286–293. The MIT Press (2015)
4. Gershenson, C., Fernandez, N.: Complexity and measuring emergence, self-organization, and homeostasis at multiple scales. Complexity **18**(2), 29–44 (2012)
5. Mendoza, L., Xenarios, I.: A method for the generation of standardized qualitative dynamical systems of regulatory networks. Theor. Biol. Med. Model. **3**(1), 13 (2006)
6. Prokopenko, M., Boschetti, F., Ryan, A.J.: An information-theoretic primer on complexity, self-organization, and emergence. Complexity **15**(1), 11–28 (2009)
7. Sani, L., et al.: Efficient search of relevant structures in complex systems. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA 2016. LNCS (LNAI), vol. 10037, pp. 35–48. Springer, Cham (2016). doi:10.1007/978-3-319-49130-1_4
8. Shalizi, C.R., Camperi, M.F., Klinkner, K.L.: Discovering functional communities in dynamical networks. In: Airoldi, E., Blei, D.M., Fienberg, S.E., Goldenberg, A., Xing, E.P., Zheng, A.X. (eds.) ICML 2006. LNCS, vol. 4503, pp. 140–157. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73133-7_11
9. Sporns, O., Tononi, G., Edelman, G.: Theoretical neuroanatomy: relating anatomical and functional connectivity in graphs and cortical connection matrices. Cereb. Cortex **10**(2), 127–141 (2000)
10. Tononi, G., McIntosh, A., Russel, D., Edelman, G.: Functional clustering: identifying strongly interactive brain regions in neuroimaging data. Neuroimage **7**, 133–149 (1998)
11. Villani, M., Filisetti, A., Benedettini, S., Roli, A., Lane, D., Serra, R.: The detection of intermediate level emergent structures and patterns. In: Liò, P., Miglino, O., Nicosia, G., Nolfi, S., Pavone, M. (eds.) Proceedings of ECAL2013, The 12th European Conference on Artificial Life. MIT Press (2013)
12. Villani, M., Roli, A., Filisetti, A., Fiorucci, M., Poli, I., Serra, R.: The search for candidate relevant subsets of variables in complex systems. Artif. Life **21**(4), 412–431 (2015)