SoftwareX 14 (2021) 100693

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ReSS: A tool for discovering relevant sets in complex systems

Laura Sani [a], Michele Amoretti [a], Stefano Cagnoni [a], Monica Mordonini [a], Riccardo Pecori [b,*]

[a] Department of Engineering and Architecture, University of Parma, Parma (PR), Italy
[b] Department of Engineering, University of Sannio, Benevento (BN), Italy

ARTICLE INFO

Article history:
Received 21 December 2020
Received in revised form 13 March 2021
Accepted 1 April 2021

Keywords:
Complex systems analysis
Relevance index
Relevant sets
Information theory

ABSTRACT

A complex system can be composed of inherent dynamical structures, i.e., relevant subsets of variables interacting tightly with one another and loosely with other subsets. In the literature, some effective methods to identify such relevant sets rely on the so-called Relevance Indexes (RIs), measuring subset relevance based on information theory principles. In this paper, we present ReSS, a collection of CUDA-based programs computing two of such RIs, either through an exhaustive search or a niching metaheuristic when the system dimension is too large. ReSS also includes a script that iteratively activates the search and identifies hierarchical relationships among the relevant subsets. The main purpose of ReSS is to establish a common and easy-to-use general RI-based platform for the analysis of complex systems and other possible applications.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00103
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	C++, CUDA, Python
Compilation requirements, operating environments	64-bit Linux o.s., CUDA > 9.0, Python > 3.7
If available Link to developer documentation/manual	https://github.com/ri-unipr/ress/releases
Support email for questions	michele.amoretti@unipr.it

1. Motivation and significance

The identification of functional structures in dynamical complex systems composed of many interacting parts is a major challenge in science. These structures reflect the dynamics of small-scale processes with peculiar characteristics and can play a critical role in determining the behavior of the whole system.

To detect such structures, several measures, many based on information theory [1], have been proposed. Tononi [2] and others [3,4] introduced a method to identify these relevant structures, considering the system status at different times and associating each possible subset of system variables with an index. This can be referred to as Relevance Index (RI), since its largest values identify the system's (most) Relevant Sets (RSs).

To completely describe the system, one needs to exhaustively compute the RI for all possible variable subsets, which becomes unfeasible as the system dimensions increase. Therefore, metaheuristics are needed to identify subsets of variables describing high-dimensional systems. Even so, the computation of hundreds of thousands to millions of RIs may be required. Since the computation of the RI for each candidate RS (CRS) is independent of the others, an efficient implementation of the index computation can be based on the use of GPUs.

To meet these requirements, we developed ReSS (Relevant Set Search), a package for computing the RIs efficiently. It includes CUDA-based C++ modules and Python scripts that compute the RIs, both exhaustively and through a niching metaheuristic, namely K-Means PSO (KMPSO) [5].

The specific modules composing ReSS fulfill the following goals:

* Corresponding author.
E-mail address: rpecori@unisannio.it (Riccardo Pecori).

https://doi.org/10.1016/j.softx.2021.100693

1. optimizing the exhaustive search by computing the RI of several CRSs in parallel;
2. providing a computationally-efficient objective function for KMPSO, to detect the RSs whenever an exhaustive approach is too computationally demanding;
3. detecting possible hierarchical dependencies between RSs.

The third goal is achieved through a script calling the basic algorithms iteratively [6]. In each iteration, the highest-index RS found in the previous one is treated as a new single variable. This way, a succession of system representations of decreasing dimension is generated. The iterations stop when the RI falls below a pre-set threshold.

ReSS was successfully employed to analyze the hierarchies of RSs in random boolean networks, chemical reactions, and real-world human communities [6], as well as to identify relevant subsets in biological networks [7], reveal communities and emotions in online social networks [8], and detect critical states in the Ising model [9].

2. Relevance indexes

ReSS computes two indexes: the normalized version of the Dynamic Cluster Index (T_c), and the Z-Index (zI).

The Dynamic Cluster Index C of a subset S of size d of a dynamic system U, is defined as the ratio between the integration of S:

$$I(S) = \sum_{x \in S} H(x) - H(S) \tag{1}$$

and the mutual information between S and the rest of the system $U - S$:

$$C(S) = \frac{I(S)}{MI(S; U-S)}, \tag{2}$$

where $MI(S; U-S) \equiv H(S) + H(S|U-S) = H(S) + H(U-S) - H(S, U-S)$ and H denotes entropy.

$I(S)$ measures a global correlation between the variables in the same set, while $MI(S; U-S)$ measures how much information is shared between S and the rest of the system.

In simpler words, large index values identify variable subsets that describe independent 'synchronized' subsystems. These may correspond to functional subsets of a physical system, sets of elements that are contemporarily active in different phases of a chemical reaction, etc.

In data science terms, the task that the considered Relevance Index allows one to perform on a dataset, represented by a relation whose rows are its instances (tuples) and columns its attributes, is to cluster data columnwise based on the correlation between attributes, instead of clustering instances based on their similarity (distance) as happens in traditional clustering.

Since $C(S)$ scales with the size of S, the indexes of differently-sized systems need to be normalized to be comparable. To this aim, a system whose variables are equally correlated, termed homogeneous system U_h, is taken as a reference. For each subset size d, the average integration $\langle I_h^d \rangle$ and the average mutual information $\langle MI_h^d \rangle$ are used to normalize the cluster index value as follows:

$$C'(S) = \frac{I(S)}{\langle I_h^d \rangle} \Big/ \frac{MI(S; U-S)}{\langle MI_h^d \rangle} \tag{3}$$

where d is the dimension of set S.

Indeed, to measure the relevance of each set S, its T_c (normalized cluster index) value is used, defined as follows:

$$T_c(S) = \frac{C'(S) - \langle {C'_h}^d \rangle}{\sigma({C'_h}^d)}, \tag{4}$$

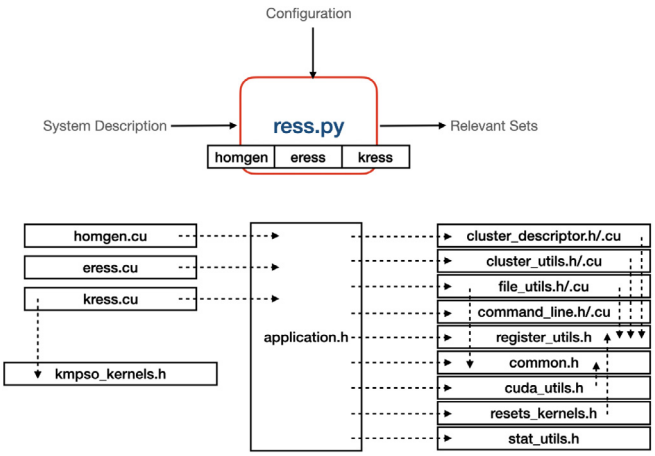

Fig. 1. ReSS architecture.

where $\langle {C'_h}^d \rangle$ and $\sigma({C'_h}^d)$ are the average and the standard deviation, respectively, of the normalized cluster indexes of all the sets from the homogeneous system having the same size d as S.

In practice, the T_c measures how much the dynamics of S diverge from those of the subsets of U_h having the same size.

If the integration is only considered, another index, termed Z-Index, can be computed to assess such a divergence. It is defined as a sort of z-score, as follows:

$$zI(S) = \frac{2nI(S) - \langle 2nI(S_h^d) \rangle}{\sigma(2nI(S_h^d))}, \tag{5}$$

where n is the number of configurations of the whole system given as input, S is a subset of U, $\langle S_h^d \rangle$ and $\sigma(S_h^d)$ are the average and the standard deviation of the integration of all the sets from the homogeneous system that have the same size d as S, respectively.

It can be demonstrated that $2nI \approx \chi^2_g$ under the hypothesis of independent random variables and with n large enough. Therefore, the terms in Eq. (5) can be computed just by knowing the degrees of freedom g of the Chi-squared distribution [10]. Indeed, under the hypothesis of independent variables:

$$\langle 2nI(S_h) \rangle = g \tag{6}$$

$$\sigma(2nI(S_h)) = \sqrt{2g} \tag{7}$$

Therefore, using the Z-Index avoids the need for defining and simulating a homogeneous system.

3. Software description

3.1. Software architecture

As per Fig. 1, ReSS is provided as a Python script (ress.py) that activates three executables (`homgen`, `eress`, and `kress`), whose main functions are defined in the files homgen.cu, eress.cu and kress.cu, respectively. `homgen` computes the statistics of the homogeneous system from the input sequence of configurations of the system to be analyzed. These statistics are needed only to compute the T_c index. `eress` implements the exhaustive RSs search, while `kress` implements the search using KMPSO [11]. Finally, by selecting the proper option, ress.py can also perform the hierarchical grouping.

The system description is expected to be provided as a text file with the following structure (for N variables):

- a header including a sequence of N space-separated variable names following the %% prefix;

- N binary patterns where the bits that encode the corresponding variable (in the order specified in the header) are set to 1;
- one row only displaying the %% string used as a separator;
- a list of rows, each representing a system configuration, where the value of each variable is binary-coded.

For example, the following representation corresponds to a system with 4 variables (B, BA, BAA, and BAAB), each encoded by two bits:

```
%% B BA BAA BAAB
11000000      % the first 2 bits represent B
00110000
00001100
00000011
%%
01010101
10011100
10011110
01010101
01010101
10011100
10011110
...
```

The instructions for running `homgen`, `eress`, `kress`, and ress.py are given in the README file provided with the source code.[1]

The output of both `kress` and `eress` is a text file having the following format:

- a header displaying N variable names, space-separated and followed by a label specifying the chosen RI;
- a list of system subsets (N bits, where 1s correspond to their constituting variables) along with their RI value.

For example, the following output regards a system with 4 variables, analyzed using the *zI*:

```
B BA BAA BAAB zI
0 0 1 1 229.3   %the zI of {BAA,BAAB} is 229.3
1 0 1 0 205.2
0 1 1 1 120.4
...
```

3.2. CUDA-based implementation of the search

In our implementation:

- Each sample is stored in a memory area including K adjacent unsigned integers, containing the N_{bit} bits needed to represent the N variables of the system. If n is the number of samples, then the system data can be stored in an array of $n \cdot K$ unsigned integers.
- Each CRS is represented as a bit mask of N_{bit} bits, where the i^{th} bit is set to 1 if the i^{th} variable is contained in the CRS.

The exhaustive computation of the selected index for all the CRSs is split into the following steps:

1. Computation of the probability distribution function (PDF) for each system variable;
2. Generation of U_h and index computation for its variable subsets (only for T_c);
3. index computation for each CRS.

[1] https://github.com/ri-unipr/ress.

In the first step, each variable is examined individually to compute its PDF. The distribution of the i^{th} variable is estimated as the frequency of the values of its bits. This information is used for the generation of U_h, if needed, and to compute the entropy H of the variable.

In the second step, if N is the dimension of the system X, U_h is generated from N independent random variables, having the same PDF as the corresponding variables of X. We obtain n samples by assigning to the i^{th} variable, for each sample, a randomly generated value from the previously estimated distribution.

Thus, U_h meets the homogeneity requirement while maintaining a relationship with X.

As shown in Fig. 2, for each possible CRS size, from 2 to $N-1$, we compute the mean and the standard deviation of C. If the considered size is d, then the CRSs to be examined are selected by scanning all possible permutations of an N-bit string containing d bits set to 1 and $N-d$ bits set to 0. Each selected CRS is assigned to a grid of T threads, responsible for computing C, with $T = N_B N_T$, where N_B is the number of blocks per grid and N_T the number of threads per block. Each CRS S is paired with its complementary subset $U - S$, whose entropy is necessary for computing the MI; thus, each grid is composed of $T/2$ complementary CRS pairs: one CRS of size d and another of size $N - d$. Therefore, if the threads are properly synchronized, it is possible to concurrently compute the statistics (mean and standard deviation) of the CRSs of both sizes, further halving the computation time.

The third step consists of three sub-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*: the entropy is computed using the frequency histogram;
3. *Computation of the final output*: the threads of the block are synchronized to make the complementary entropy available to each CRS. This also enables the computation of the MI, when computing the T_c.

Calculating the frequency histogram is, computationally, the hardest step. To obtain a good trade-off between performance and memory usage, we generated a hash map, pre-allocated for each thread to be managed by the GPU kernel computing the histogram.

The module computing the T_c or the *zI* is a simple extension of the one that computes the corresponding unnormalized index.

3.3. K-means PSO metaheuristic

An exhaustive search is not feasible for high-dimensional systems, because of the curse of dimensionality. This is why we suggest that the proposed software compute the index exhaustively for systems of dimension up to about 20, while for larger ones one should rely on `kress`. In `kress`, K-means PSO preserves diversity within the swarm, exploring many peaks simultaneously, by alternating classical PSO [12] and K-means [13] clustering steps.

PSO is a continuous optimization method. In this package, it deals with a discrete domain problem, in which each CRS is represented as an N-bit binary string $\mathbf{P_i}$, whose bits are set to 1 if the corresponding variable is included in the CRS, or to 0 otherwise.

Since a particle is actually an N-dimensional vector $\mathbf{p_i} \in \mathbb{R}^N$, `kress` uses the sign of each component to transform its real value into the corresponding bit of $\mathbf{P_i}$ as follows:

$$P_{i,r} = \begin{cases} 1 & \text{if } p_r \geq 0 \\ 0 & \text{if } p_r < 0 \end{cases} \tag{8}$$

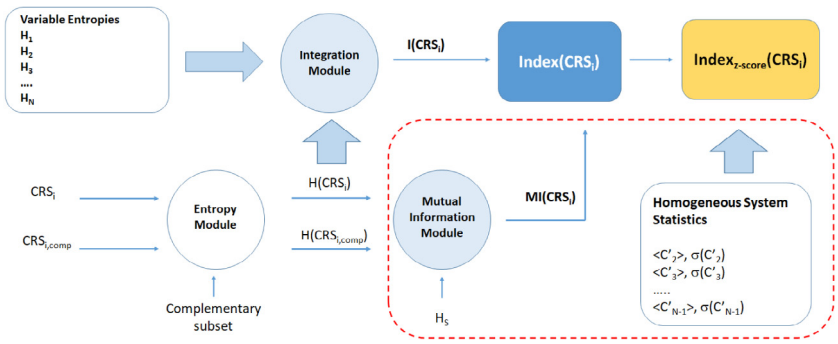

Fig. 2. Computation of the selected index, with the sub-modules for the T_c in dashed red.

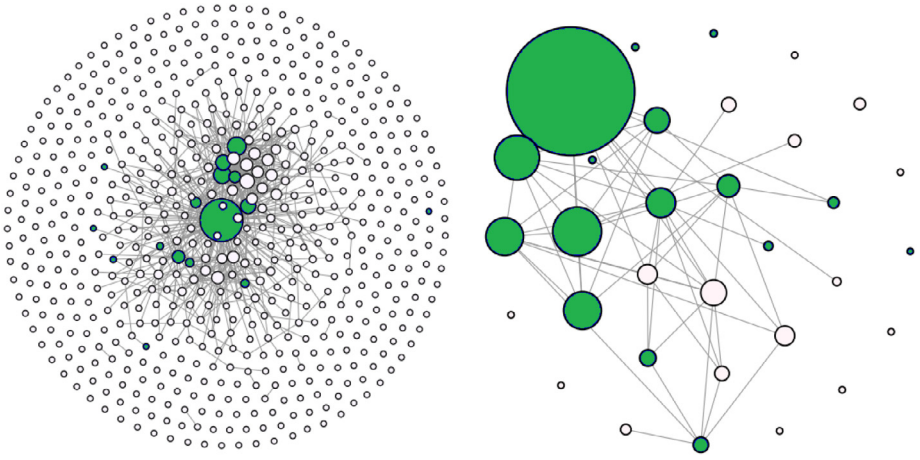

Fig. 3. The 612 users (left) of the Facebook group with the detected sentiment-based community (green nodes) and the 32 users (right) posting at least 15 times in the considered period. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The fitness $F(\mathbf{p_i})$ of a particle $\mathbf{p_i}$ is equal to the RI value of the CRS that $\mathbf{p_i}$ represents. The fitness function F is to be maximized. Finally, `kress` outputs the list of the best N_s sets found during the whole search process.

4. Illustrative examples

We provide two examples of usage of ReSS: the identification of relevant sets in an online social network, and the detection of a hierarchical structure in a chemical reaction.

4.1. Unveiling social relationships

This example entails a Facebook group composed of patients affected by Hidradenitis Suppurativa. The dataset was obtained using the information about the group members who published at least 15 posts between 2010 and 2016. This allowed us to reduce the number of considered users to 32 and to study the most active members. Each sample of the system represents a specific month within the period of interest. In particular, we focused on a *sentiment-based analysis*, i.e., the value corresponding to a certain user can assume 4 different categorical values, corresponding to the prevailing mood of her/his posts (no posts, positive, negative, neutral).

Fig. 3 represents, on the left, all the 612 users of the Facebook group. Each node of a graph is a user and each edge is a friendship relationship between two users, while the size of each node is proportional to the node degree. The green nodes represent the highest-T_c RSs. The right side of the figure focuses on the 32 users posting at least 15 times in the considered period.

The highlighted set of users is the most representative in terms of T_c, showing that the application of ReSS to the Facebook group identified an interesting community, characterized by the presence of many friendship connections within its members. This result confirms the effectiveness of such a support group and the psychological influence that its manager (the largest circle) exerts on the members more closely connected to her.

4.2. Hierarchical grouping

In this subsection, we describe an application of the third algorithm implemented in ReSS, i.e., the search of the RSs based on a hierarchical grouping of the variables according to the iterative strategy mentioned in Section 1.

The example concerns a Catalytic Reaction System (CatRS), involving enzymatic condensations and self-maintaining behaviors such as Reflexive Autocatalytic Food-generated (RAF). The considered entities are molecular species represented by letters. Their state, in the input file, is represented by an encoding of their concentration trend (increasing, decreasing, steady). In Fig. 4, they form a CatRS of seven reactions, divided into two RAFs: a linear chain (RAF1) and a second one (RAF2), where two reciprocally catalyzing reactions are the roots of another linear chain.

Fig. 5 shows that, after five iterations, RAF1 has already been identified as a single RS, whereas, within the more complex RAF2, the other detected RSs highlight strict relations among the reagents and the catalyzer of the same reaction. The available time series is too short to permit an immediate detection of the whole RAF2 group with sufficient significance ($T_c \geq 3$). However, it is actually possible to detect the whole set by going on iterating the method after the groups that are found start having significance below the chosen threshold ($T_c = 3$).

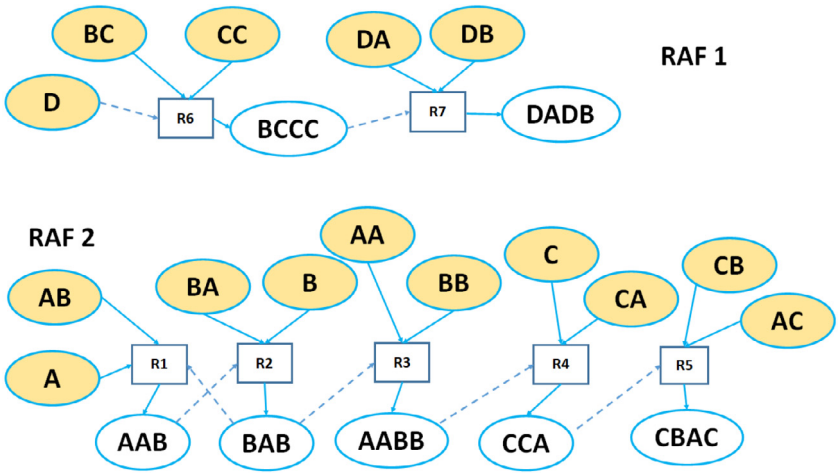

Fig. 4. The considered CatRS. Ellipses represent chemical species (injected in orange, created in white), rectangles represent reactions. Dashed lines indicate the catalytic role of a species. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

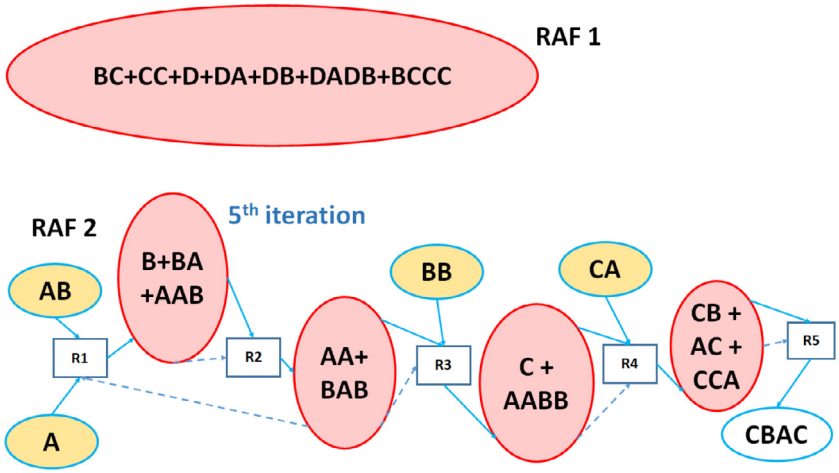

Fig. 5. The five RSs (red) found after the first five iterations of the hierarchical grouping. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Impact

The purpose of this article is to publish ReSS, an open source package to analyze complex systems through the computation of the RIs. ReSS has already been tested in different research scenarios, enabling both a speed-up of the index computation and the search of relationships among the system variables.

Regarding the speed-up, we obtained manageable computation times by using `kress` rather than an exhaustive approach. Statistics of 10 independent runs are summarized in Table 1.[2] They relate to the 26-variable CatRS described in Section 4,[3] a 28-variable stochastic artificial system reproducing a Leaders & Followers (LF) behavior, and a 56-variable system (Green Network Community, GNC) whose status represents the participation of 56 partners in several project meetings. Notice that, for such a size, an exhaustive search is unfeasible on a standard computer, even using GPU parallelization. Also the computation time required by `kress` does not only depend on the size of the system, but also on the number of status samples in the dataset, and on the statistical distribution of the index values over the CRSs. In fact, finding RSs in a real-world system as the CatRS is harder than in larger artificial LF systems, since more and more complex relationships exist among the variables.

Tests were run on a Linux server equipped with two Intel Xeon Silver 4210 2.2 GHz CPUs, 64 GB of RAM, and a GeForce RTX 2080 8GB GDDR6 256-bit GPU by NVIDIA.

As regards applications, beyond its use for the native goals, as described in the examples (see [6–8,14] for more details), the application of ReSS to pattern recognition tasks also seems to be promising. In particular, in some preliminary tests, ReSS aided the detection of malicious users in social networks [15], unsupervised feature extraction applied to character recognition and classification of DNA sequences [16], and pattern clustering [17], obtaining interesting results and hints for further exploration of these mostly unexplored additional capabilities.

ReSS can be further extended to compute other RIs.

6. Conclusions

We have presented ReSS, a package that discovers RSs in complex systems. Presently, no other tools for computing the considered RIs are freely available. The distribution of ReSS, besides allowing more researchers to experiment with RIs, could

[2] The algorithm configuration is the same as in the corresponding scripts included in the package.

[3] Only 21 variables are relevant, since 5 variables have the same value in all instances.

Table 1
Running times, not including the homogeneous system computation that is equal for all cases, for `kress` (K) on three different systems (with N variables and Nd samples) and comparison with `eress` (E) where possible.

System	N	Nd	Time [s] ± std (E)		Time [s] ± std (K)		Avg. speedup	
			T_c	zI	T_c	zI	T_c	zI
CatRS	21	751	0.57 ± 0	0.567 ± 0.004	3.438 ± 0.03	3.88 ± 0.04	0.165	0.146
LF	28	150	83.173 ± 0.842	83.672 ± 0.218	3.575 ± 0.031	3.902 ± 0.020	23.265	21.443
GNC	56	124	n.a.	n.a.	16.401 ± 0.39	21.517 ± 0.26	n.a.	n.a.

foster the comparison of the approach based on such indexes with other methods aiming at identifying the core subsets of complex systems as well as new research in pattern recognition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Marco Villani, Roberto Serra, and Andrea Roli for the theoretical background and the basic sequential code of the exhaustive search module (`eress`). We also thank Emilio Vicari for the CUDA code of `eress` and Gianluigi Silvestri for the CUDA code of k-means PSO (`kress`). Part of this research has been carried out at the High Performance Computing (HPC) facility of the University of Parma.

References

[1] Gershenson C, Fernandez N. Complexity and information: Measuring emergence, self-organization, and homeostasis at multiple scales. Complexity 2012;18(2):29–44.
[2] Tononi G, McIntosh A, Russel D, Edelman G. Functional clustering: Identifying strongly interactive brain regions in neuroimaging data. Neuroimage 1998;7(2):133–49.
[3] Sporns O, Tononi G, Edelman G. Theoretical neuroanatomy: Relating anatomical and functional connectivity in graphs and cortical connection matrices. Cerebral Cortex 2000;10(2):127–41.
[4] 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 2015;21(4):412–31.
[5] Passaro A, Starita A. Particle swarm optimization for multimodal functions: A clustering approach. J Artif Evol Appl 2008;2008:15.
[6] Villani M, Sani L, Pecori R, Amoretti M, Roli A, Mordonini M, Serra R, Cagnoni S. An iterative information-theoretic approach to the detection of structures in complex systems. Complexity 2018;2018:15.
[7] Villani M, Sani L, Amoretti M, Vicari E, Pecori R, Mordonini M, Cagnoni S, Serra R. A relevance index method to infer global properties of biological networks. In: Pelillo M, Poli I, Roli A, Serra R, Slanzi D, Villani M, editors. Artificial life and evolutionary computation. Cham: Springer International Publishing; 2018, p. 129–41.
[8] Sani L, Lombardo G, Pecori R, Fornacciari P, Mordonini M, Cagnoni S. Social relevance index for studying communities in a facebook group of patients. In: Sim K, Kaufmann P, editors. Applications of evolutionary computation. Cham: Springer International Publishing; 2018, p. 125–40.
[9] Roli A, Villani M, Caprari R, Serra R. Identifying critical states through the relevance index. Entropy 2017;19(2):73.
[10] Papoulis A, Pillai SU. Probability, random variables, and stochastic processes. Boston: McGraw-Hill; 2015.
[11] Silvestri G, Sani L, Amoretti M, Pecori R, Vicari E, Mordonini M, Cagnoni S. Searching relevant variable subsets in complex systems using K-means PSO. In: Pelillo M, Poli I, Roli A, Serra R, Slanzi D, Villani M, editors. Artificial life and evolutionary computation. Cham: Springer International Publishing; 2018, p. 308–21.
[12] Poli R, Kennedy J, Blackwell T. Particle swarm optimization. Swarm Intell 2007;1(1):33–57.
[13] MacQueen JB. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th berkeley symposium on mathematical statistics and probability* 1967, p. 281–97.
[14] Sani L, Amoretti M, Vicari E, Mordonini M, Pecori R, Roli A, Villani M, Cagnoni S, Serra R. Efficient search of relevant structures in complex systems. In: Adorni G, Cagnoni S, Gori M, Maratea M, editors. AI*IA 2016 advances in artificial intelligence. Cham: Springer International Publishing; 2016, p. 35–48.
[15] Sani L, Pecori R, Fornacciari P, Mordonini M, Tomaiuolo M, Cagnoni S. A relevance index-based method for improved detection of malicious users in social networks. In: Cicirelli F, Guerrieri A, Pizzuti C, Socievole A, Spezzano G, Vinci A, editors. Artificial life and evolutionary computation. Cham: Springer International Publishing; 2020, p. 78–89.
[16] Sani L, Pecori R, Mordonini M, Cagnoni S. From complex system analysis to pattern recognition: Experimental assessment of an unsupervised feature extraction method based on the relevance index metrics. Computation 2019;7(3):39.
[17] Sani L, D'Addese G, Pecori R, Mordonini M, Villani M, Cagnoni S. An integration-based approach to pattern clustering and classification. In: Ghidini C, Magnini B, Passerini A, Traverso P, editors. AI*IA 2018 – advances in artificial intelligence. Cham: Springer International Publishing; 2018, p. 362–74.