Visualizing the Template of a Chaotic Attractor

Maya Olszewski[1], Jeff Meder[1], Emmanuel Kieffer[2], Raphaël Bleuse[1], Martin Rosalie[2], Grégoire Danoy[1](✉), and Pascal Bouvry[1,2]

[1] FSTC/CSC-ILIAS, University of Luxembourg, 6, Avenue de la Fonte, 4364 Esch-sur-Alzette, Luxembourg
{maya.olszewski.001,jeff.meder.001}@student.uni.lu, {raphael.bleuse,gregoire.danoy,pascal.bouvry}@uni.lu

[2] SnT, University of Luxembourg, 6, Avenue de la Fonte, 4364 Esch-sur-Alzette, Luxembourg
{emmanuel.kieffer,martin.rosalie}@uni.lu

Abstract. Chaotic attractors are solutions of deterministic processes, of which the topology can be described by templates. We consider templates of chaotic attractors bounded by a genus–1 torus described by a linking matrix. This article introduces a novel and unique tool to validate a linking matrix, to optimize the compactness of the corresponding template and to draw this template. The article provides a detailed description of the different validation steps and the extraction of an order of crossings from the linking matrix leading to a template of minimal height. Finally, the drawing process of the template corresponding to the matrix is saved in a Scalable Vector Graphics (SVG) file.

Keywords: Chaotic attractor · Template
Linking matrix · Optimization · Visualization

1 Introduction

Resulting of theoretical studies on chaos attractors, applications including chaotic dynamics can be found in a multitude of domains. Their range goes from computer science [23], through classical sciences with physical networks [14], biology and genetics [27] and chemistry with chaotic dynamics in chemical reactions [8], all the way to electronics and chaos in electronic devices [13] and even environmental studies on population evolution [5].

Birman and Williams [6] introduce templates as knot-holder to describe the topological structure of chaotic attractors. The notion of linking matrices to describe chaotic attractors with integers has been first introduced by Mindlin *et al.* in 1990 [18]. The matrix contains the number of torsions and permutations occurring along the flow of an attractor. The template is a ribbon graph combined with a layering graph. In 1998, Gilmore wrote an extensive survey on the research on chaotic dynamical systems over the past decade [11], in which one can see various drawings of templates. In his paper, he provides the summary of the topological analysis from dynamical system to template.

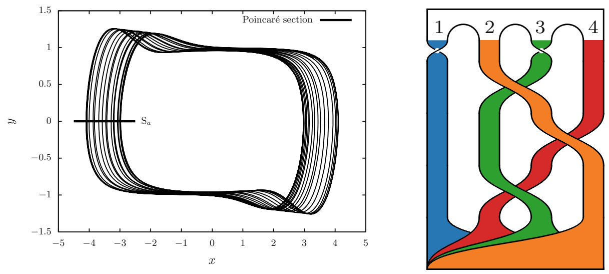
The subject of chaotic dynamics studies are promising and on-going. But it clearly misses matrices validation and drawing tools. The research community would benefit from an efficient application that verifies the validity of matrices and draws their corresponding template. The novel tool presented in this paper is publicly available online at https://gitlab.uni.lu/pcog/cate, and aims to fill this gap.

This paper is structured as follows. In Sect. 2 we give an introduction to the problem. Section 3 provides a state-of-the-art analysis in the field of chaotic attractors, focusing on their validation and visualization. In Sect. 4, we first outline our approach to determine the validity of a linking matrix. Secondly, we describe the procedure to get the minimal height of a template and its visualization. In Sect. 5, we present the experimental work and the results in order to validate our proposed approach. Finally, we conclude and outline some directions for future work in Sect. 6.

2 Problem Description

A chaotic attractor is a solution of a dynamic deterministic process that is very sensitive to its initial conditions. The solution will converge to the same global shape (the attractor), independently of the starting position in the basin of attraction. Malasoma [16] proposed a simple differential equations system

$$\begin{cases} \dot{x} = y \\ \dot{y} = z \\ \dot{z} = -\alpha z + xy^2 - x, \end{cases} \tag{1}$$



(a) Chaotic attractor (b) Template

Fig. 1. A representation of a template of a chaotic attractor solution to the Malasoma system (1) for $\alpha = 2.027$. (a) Chaotic attractor with the Poincaré section (see [25] for the definition of this section named S_a). (b) Template of the chaotic attractor from the Poincaré section.

with chaotic dynamics as solutions when $\alpha \in [2.027; 2.08]$. A detailed analysis of the topological properties of the attractors that can be produced by this system has been proposed in [24,25]. For instance, Fig. 1 summarizes some steps of the topological characterization (Poincaré section and template) of a chaotic attractor when $\alpha = 2.027$. In this article, we are considering only attractors bounded by genus–1 torus such as Rössler attractors [26] or Malasoma attractors [16] (Fig. 1a); it does not work for more complex attractors such as Lorenz attractors [15] bounded by a genus–3 torus.

A *template* is a compact branched two-manifold with boundary and smooth expansive semiflow built locally from two types of charts: joining and splitting [10]. It is a figure that represents the topological structure of a chaotic attractor. Since the 1990s there have been two different ways to represent templates with linking matrices that are still used today, as one can see in the recent paper of Gilmore and Rosalie [12], where algorithms are given to switch from one representation to the other. Hereinafter, the representation first given by Melvin and Tufillaro [17] is considered. This representation only requires a linking matrix, and gives a standard representation at the end, where at the bottom of the template the strips are ordered from the back-most on the left to the front-most on the right. This is the representation used for the template shown in Fig. 1. We also use the orientation convention defined by Tufillaro *et al.* [17,18] (Fig. 2).

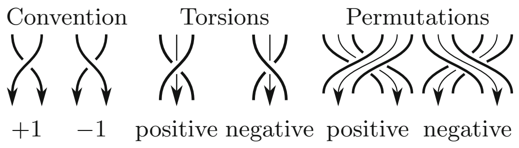


Fig. 2. Convention of representing oriented crossings. The permutation between two branches is positive if the crossing generated is equal to +1, otherwise it is negative. We use the same convention for torsions.

A linking matrix is a matrix that details the number and the direction of crossings in a template. As illustrated in Fig. 2, a torsion is a twist of a branch with itself and a permutation is an exchange of position of two branches. Furthermore, the torsions and permutations can be either positive or negative as defined by the orientation convention shown in Fig. 2. The linking matrix M corresponding to Fig. 1 is given by (2).

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & 0 \end{bmatrix} \tag{2}$$

The diagonal elements in the linking matrix correspond to the torsions. As an example, consider matrix M. The element $M_{1,1} = 1$ represents the number of torsions of branch one of the template from Fig. 1. This branch performs exactly one single positive torsion as indicated by the matrix M. The non-diagonal elements correspond to the number of permutations between the different branches. As an example, $M_{2,4} = -1$ means that branches two and four perform a negative permutation which is depicted by the crossing of the orange and red branch in Fig. 1. It is sufficient to consider the part of the matrix above the diagonal, as it is symmetric.

The linking matrix M is unique but the corresponding template can be drawn in various ways. Some representations can be longer than others. This is why our goal is to find the most concise template. This means that we aim to maximize the number of permutations per level of the template. There might be however several templates with minimum size. In this work we only consider the first template of minimum size generated by the algorithm.

An important remark is that not every matrix corresponds to a valid template of a chaotic attractor. As a chaotic attractor is a solution of a deterministic process and the linking matrix represents it, such a matrix needs to fulfill certain criteria. We will describe the tool we created to verify the validity of a linking matrix, to solve the underlying scheduling problem to find the order of the permutations and to determine the most concise representation of a template. Finally, the tool also renders the solution found.

3 Related Work

The visualization of a template has been addressed in Chap. 5 Sect. 5 of [28] and, according to our best knowledge, the validation of a linking matrix has never been addressed. Usually, this has been done manually by each author. The only comparable project we found is a Mathematica code written by Tufillaro *et al.* [28], which draws templates. Extensive details are available in the Chap. 5 of [28]. It has been used recently in papers written by Barrio *et al.* [2–4]. This implementation, however, only works on older versions of Mathematica. Furthermore, one has to specify as input an explicit order of crossings, which means that it does not find them automatically from a linking matrix, unlike the algorithm presented in this paper. This Mathematica code does not provide a validity verification either, it is purely a tool for drawing "clean" templates.

To the best of our knowledge, such a tool has never been proposed and could be beneficial for the scientific community, as it is not always easy to see whether a matrix is valid or not. Indeed there have been publications with invalid matrices that our tool would have marked as such [18]. Some other papers have presented quite unattractive drawings of templates (eg. Fig. 4 of [1]) and we feel that our tool would provide researchers with an easy and rapid way to solve this problem. Moreover, it can also be used by the community as a tool for building a linking matrix from the linking number numerically obtained during the topological characterization method for attractors bounded by a genus–1 torus (see [11,21] for details).

4 Linking Matrix and Template of a Chaotic Attractor

In this section, we are going to discuss the approach we developed in order to check the validity of a given linking matrix, to find a corresponding template of minimal height as well as to visualize it. Firstly in Sect. 4.1, we will describe the different validation steps which we are applying on a matrix and justify their necessity. Secondly, Sect. 4.2 explains the tree construction we use in order to minimize the height of the resulting template and the methods we apply for the visualization of the template.

Algorithm 1. Drawing of the template of a linking matrix.

```
1: verify correct matrix input form
2: verify continuitiy constraints of matrix
3: verify determinism constraints of matrix
4: if passed all verification steps then:
5:     construct tree
6:     find shortest path in tree
7:     draw template
```

4.1 Validation of a Linking Matrix

A linking matrix is a topological representation of a chaotic attractor, hence it needs to satisfy certain constraints linked to the attractor. Essentially, a template consists of strips that are stretched, twisted, folded and glued at the bottom over and over again after a clockwise rotation. We remind that we are only considering templates of attractors bounded by a genus–1 torus.

In order to visualize this, one can imagine having a sheet of paper split into several strips. The behavior of those strips is given by the elements of the matrix. If one can deform the paper in such a way that the paper respects the constraints given by the matrix without having to tear it apart, then the matrix corresponds to a valid template. If tears are unavoidable, no valid template exists. If there is a tearing mechanism in the attractor, we are out of the scope because this means that the attractor is at least bounded by a genus–2 torus.

Validation Steps. The steps below evaluate whether or not a linking matrix is valid, i.e., if it corresponds to a chaotic attractor.

First of all, we need to verify that a matrix is of the right form. A valid linking matrix, by definition, has a certain construction. It is square, symmetric and has integers as values [17].

The next three validation steps are constraints on the continuity of the template. Going back to the sheet of paper example, these constraints guarantee that no tears occur. The first of these constraints is linked to the diagonal elements of the matrix. These elements have to respect the condition which dictates that they have to differ by exactly one from their diagonal neighbors. Violating this constraint would result in a discontinuous template. Similar to the diagonal constraint, a linking matrix needs to satisfy the condition which states that an arbitrary value in the matrix cannot differ from the values of all of its neighbors by more than one. Finally, the last continuity constraint is based on the order of the elements on the bottom of the template. From a linking matrix, one needs to be able to obtain a valid order for the template. The order is an array which defines the position of the branches at the bottom of the template after performing the crossings. We obtain this order from the matrix by applying a simple algorithm described in [17]. A valid ordering array contains all branch indexes exactly once. An index being present twice would mean that two branches would end up at the same end position, which is impossible without a tear and therefore would result in an invalid template.

The last two verification steps are linked to the determinism of a chaotic attractor. As stated earlier, chaotic attractors are solutions of dynamic deterministic systems, meaning that from any starting point there is a unique image and no choice is possible. As the template is a topological representation of a chaotic attractor, it also needs to respect its intrinsic properties like determinism. The first of those two verifications consists in checking whether the linking matrix has 2×2 sub-matrices located on its diagonal that are not valid. Up to addition of a global torsion (see [24] for details) there are two 2×2 matrices that are not valid, namely B and C:

$$\left\{ B = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}, \; C = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}, \; C + 1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \ldots \right\}. \tag{3}$$

The set (3) corresponds to matrices that are associated to discontinuous templates. If the matrix has such a sub-matrix on its diagonal, this means that it presents a choice opportunity at some point and violates the determinism condition. Therefore, it is not valid.

Finally, in the second step, which we call planarity check, we verify the order of the end positions of the template. The idea is to take the final positions of the branches at the end of the template, and connect them with arcs in a certain way. Start with 1, and connect it to 2 over the list. Then connect 2 to 3 below the list, 3 to 4 over, and so on. If the arcs cannot be drawn without intersecting, then the matrix is invalid. This is illustrated by Fig. 3, where the left part of the figure corresponds to this verification of the matrix (2), and has no intersections. The right side on the other hand corresponding to matrix N (4) does not pass the test.

$$N = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix} \tag{4}$$

If this planarity condition was not verified and there was an intersection, the system would have a choice when arriving at this intersection, which would violate the determinism assumption. Therefore, a matrix that does not satisfy this condition cannot correspond to a valid template.

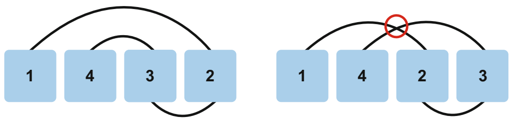


Fig. 3. Planarity check of matrices (2) (left) and (4) (right).

Order of the Validation Steps. The order of the different validation steps is defined in the way described above, we start checking the most general constraints, and then check the most specific ones (Algorithm 1). For example, if a matrix is not square matrix, there is no need to verify specific constraints like the diagonal constraint, as the matrix is not even a linking matrix by definition. The same idea applies to the other criteria.

In literature, there have been publications with invalid matrices that our procedure would have labeled as such. One example would be the first 4×4 linking matrix in [18], which gives the matrix with the following diagonal elements: 6, 5, 5 and 4. This matrix would not have passed the validation step which dictates that all elements on the diagonal of a matrix have to differ by one from their diagonal neighbors.

$$K = \begin{bmatrix} 3 & 2 & 2 & 3 \\ 2 & 2 & 2 & 3 \\ 2 & 2 & 3 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \tag{5}$$

For the matrix K (5) the ordering validation step fails because the ordering at the end is given by the array [2, 2, 3, 3], meaning that both strips one and four are on position two and strips two and three are on position three. As this is a problem for continuity, this matrix would not pass the order test. This illustrates that a tool to validate a matrix would facilitate the analysis of linking matrices, as it is not always easy to see whether a matrix is valid or not. A complete example of the validation process can be found in the appendices of the extended version [20].

4.2 Visualization of a Template

Tree Construction. After having verified the validity of a linking matrix, the next step is to generate a visualization of a template with minimal height from a given linking matrix. In order to determine the minimal height of a template, one has to optimize the scheduling of all the crossings between the different branches. For this purpose, we developed an approach where we take as input a valid linking matrix and make use of its permutations to generate a tree graph using a breadth first approach, meaning that we build it level by level.

To do this, we follow Algorithm 2. We derive the initial order from the matrix which represents the root of the tree as a first step. Furthermore, we also retrieve the list of performable permutations between the branches. Beginning at the

Algorithm 2. Tree construction

```
1: if $validMatrix(matrix)$ then
2:     $init = Node(permutationList, order, father = None)$ ;
3:     $finalOrder = getFinalOrder(matrix)$
4:     $queue = [init]$ ;
5:     while $queue \neq \varnothing$ do
6:         $node = queue[0]$ ;
7:         $queue = queue[1 :]$ ;
8:         $toExecute = permutationList \cap allNeighborCombinations(node.order)$ ;
9:         if $toExecute = \varnothing$ and $node.order = finalOrder$ then
10:            $setLeaf(node)$ ;
11:            $break$ ;
12:        for $p$ in $toExecute$ do
13:            $newNode =$
14:            $Node(updatedPermutationList(p), updatedOrder(p), father = node)$ ;
15:            $queue.append(newNode)$ ;
```

root, we simulate the permutations and generate additional nodes which are annotated with an updated order and then added to the tree. For each node created, the list of permutations yet to be performed will differ. Eventually, a node representing a leaf with an empty permutation list and a valid final order will be generated. At this point, the computation of the tree is stopped. By traversing the tree from the root to that leaf, we get the sequence of permutations to execute in order to obtain a template of minimal height. To illustrate this procedure, consider the following 4 × 4 matrix A (6).

$$A = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix} \tag{6}$$

From this matrix, we get an initial order where the branches are numbered beginning from 1 to 4. To retrieve the set of permutations to perform, we have to consider the non-diagonal elements of the matrix. For example, the branch with the label 1, has to perform a negative permutation with the branches 2, 3 and 4. There is also a positive permutation between branch 3 and 4. So, we obtain the following list of permutations which needs to be executed $[(1, 2), (1, 3), (1, 4), (3, 4)]$.

To find the permutations which can be performed at this stage, we need to consider our initial order from which we can derive which branches are direct neighbors. For instance, we obtain the following list of neighbor pairs $[(1, 2), (2, 3), (3, 4)]$. By taking the intersection of the neighbor list and the set of permutations to perform, we obtain a set of permutation which are possible to process during the initial stage. By doing so, we can permute branch 1 and 2 or 3 and 4. However, we could also perform both permutations in parallel as performing one of them does not prohibit the other one. As illustrated on top of

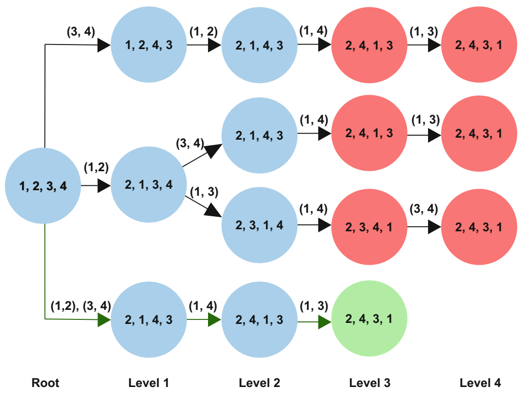


Fig. 4. Final and complete tree for matrix A from (6) including the root and the child nodes generated per level. Each node represents the updated order after each permutation described by the incoming edge. (Color figure online)

Fig. 4, we see the root labeled with the initial order of the branches. After the first set of permutations have been performed, different child nodes are created at level 1. The corresponding order of each child node is obtained by switching the positions of the permuted branches in the initial order of the root.

From the new order of each child node, we try to find a new permutation to perform by defining the neighbor pairs. We then recompute the possible permutations for this iteration. Each iteration will add one or more children to tree and this process is repeated until all permutations have been performed or no new permutation can be computed. However, a node which can no longer perform a permutation while there are still some permutations in the set left to be executed, is not considered valid.

Figure 4 also shows the final tree after all permutations have been performed. The green arrows leading to the green colored leaf denote the shortest path where the labels show the order of execution of the permutations to get to the final order of the template. This will result in a template of shortest possible height. There are also three other possible solutions but they will not reduce the height of the template to a minimum as they perform one additional permutation. However, we stop the computation of building the tree after encountering the first valid leaf, so the red nodes will never be computed. The breadth-first construction of the tree guarantees that the first found solution is the shortest one.

Drawing of the Template. Finally, after verification of the linking matrix and after having found the shortest path in the tree corresponding to the most concise order of crossings, we can now draw the template. To draw the templates as scalable vector graphics, we used python's `swgwrite` module [19].

In order to draw both torsions and permutations, we use a cubic Bézier curve as shape. To illustrate how we use it, consider two points (x_1, y_1) and (x_2, y_2)

and suppose we want to draw this Bézier curve between them, in the same shape as those used in the permutations and torsions. The starting point is given by (x_1, y_1) and we will give the rest of the points relative to this starting point. The relative end point is then given by $(x_2 - x_1, y_2 - y_1)$ and the two relative control points by $(x_1, (y_2 - y_1)/2)$ and $(x_2 - x_1, (y_2 - y_1)/2)$. So the control points are always halfway in height between the two points and straight above respectively below them.

To draw a torsion we first draw one Bézier curve, then add a small white circle in the middle of this curve to *erase* this part. Finally we draw the other Bézier curve. This procedure is illustrated in Fig. 5(a–c). Permutations are drawn in a similar way. The sign of the permutation defines which of the two branches is drawn first, then when the other one is drawn it covers it up as it comes on top of the other one (Fig. 5(d–e)).

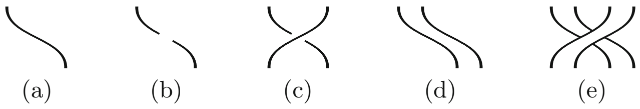


Fig. 5. An illustration of a positive torsion (a–c) and a positive permutation (d–e) drawing process.

We start by considering the torsions of the matrix and draw all of them. Then we move on to the permutations. They are given by the sequence of edges forming the shortest path of the tree generated by the input matrix. We then draw the rest of the template by levels. At each level, every strip can do one of three actions: do a straight transition, permute left or permute right. The shortest path tells us which two strips should permute. Given this information, it is easy to calculate the coordinates at the next level of each strip and apply the correct transition (Fig. 6).

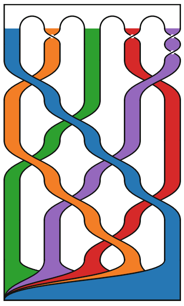


Fig. 6. Template of one linking matrix with five branches and eight permutations.

5 Performance Evaluation

An elementary matrix is a unique linking matrix describing a chaotic mechanism without additional torsions or symmetry properties [22]. Given an input size, Rosalie describes in this article a method to generate all possible elementary linking matrices of such size. We used this method to obtain the 14, 38 and 116 possible elementary matrices with resp. five, six and seven branches (resp. 5×5, 6×6 and 7×7 linking matrices). Figure 7 depicts for each matrix size the distribution of the elementary matrices with respect to the number of permutations to process.

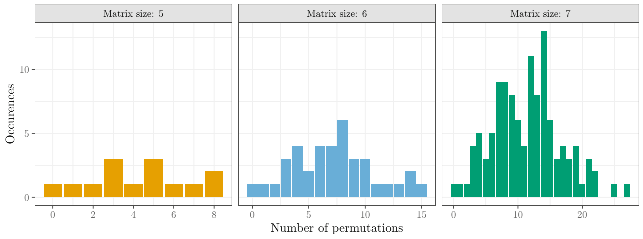


Fig. 7. Distribution of the number of elementary matrices with respect to the number of permutations to process. There are 14 (resp. 38 and 116) matrices of size 5 (resp. 6 and 7).

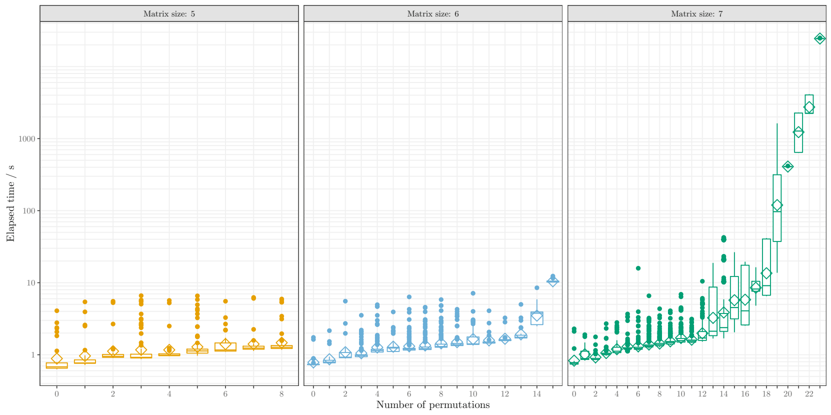


Fig. 8. Elapsed time depending on the number of permutations for the matrices depending on their size. The diamond represents the average value.

The experiments were conducted on a server with an Intel Xeon X7560 processor with a clock speed of 2.27 GHz, and 1024 GB of RAM. Even though this server is not the fastest available, it is the only one fulfilling the memory requirements (instances required between 25 MB and 400 GB of memory). For the sake of comparability, all instances have been run on the same machine. For the complete description of the cluster environment, please refer to https://hpc.uni.lu/systems/chaos/. We computed the templates of all the elementary matrices described above. We ran the experiments with version v0.0.1 of the code. For each input matrix, we measured 30 times the time elapsed to get the template. The 7×7 matrix with 27 permutations ran out of memory and crashed: we removed it from the graphs. Figures 8 and 9 depict the elapsed computation time with respect to the number of permutations to process. As expected, we observe a drastic rise that characterizes a combinatorial explosion in the number of permutations.

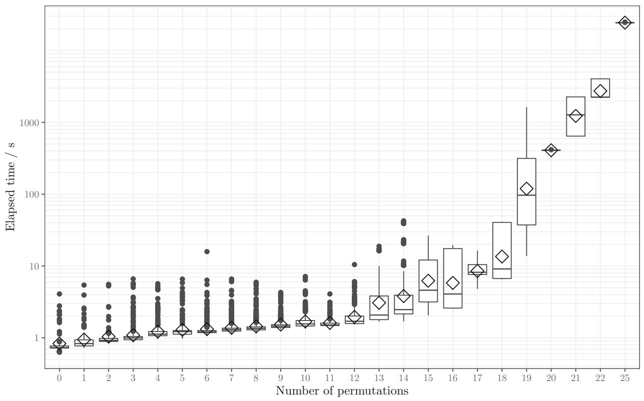


Fig. 9. Elapsed time depending on the number of permutations for the 167 matrices. The diamond represents the average value.

6 Conclusion

In this paper, we presented a tool which verifies whether a linking matrix corresponds to a topologically valid template. Moreover, our approach computes and draws a template of minimal height corresponding to this linking matrix. This is especially interesting for linking matrices with a higher number of crossings. We believe that this tool could benefit the research community as it eases the process of verifying the validity of a linking matrix, and quickly draws one of its matching templates.

A possible extension of our work could be to represent the generated templates as a 3D model in an automated way. One representation of a 3D template was given by Cross and Gilmore, where they include the torsions as a part of the global modification [9]. Another visualization was given by Boulant *et al.*

(Fig. 6 of [7]). Such a 3D visualization would allow to be even closer visually to the nature of a chaotic attractor, and thus could provide more intuitive insights.

Acknowledgments. The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [29] (see https://hpc.uni.lu). This work is partially funded by the joint research programme UL/SnT-ILNAS on Digital Trust for Smart ICT.

References

1. Anastassiou, S., Bountis, T., Petalas, Y.G.: On the topology of the Lü attractor and related systems. J. Phys. A: Math. Theor. **41**(48), 485101 (2008). https://doi.org/10.1088/1751-8113/41/48/485101
2. Barrio, R., Blesa, F., Serrano, S.: Qualitative analysis of the rössler equations: bifurcations of limit cycles and chaotic attractors. Phys. D: Nonlinear Phenom. **238**(13), 1087–1100 (2009). https://doi.org/10.1016/j.physd.2009.03.010
3. Barrio, R., Blesa, F., Serrano, S.: Topological changes in periodicity hubs of dissipative systems. Phys. Rev. Lett. **108**(21), 214102 (2012). https://doi.org/10.1103/physrevlett.108.214102
4. Barrio, R., Dena, A., Tucker, W.: A database of rigorous and high-precision periodic orbits of the Lorenz model. Comput. Phys. Commun. **194**, 76–83 (2015). https://doi.org/10.1016/j.cpc.2015.04.007
5. Benincà, E., Ballantine, B., Ellner, S.P., Huisman, J.: Species fluctuations sustained by a cyclic succession at the edge of chaos. Proc. Nat. Acad. Sci. **112**(20), 6389–6394 (2015). https://doi.org/10.1073/pnas.1421968112
6. Birman, J.S., Williams, R.F.: Knotted periodic orbits in dynamical systems–I: Lorenz's equation. Topology **22**(1), 47–82 (1983). https://doi.org/10.1016/0040-9383(83)90045-9
7. Boulant, G., Lefranc, M., Bielawski, S., Derozier, D.: A nonhorseshoe template in a chaotic laser model. Int. J. Bifurcat. Chaos **08**(05), 965–975 (1998). https://doi.org/10.1142/s0218127498000772
8. Budroni, M.A., Calabrese, I., Miele, Y., Rustici, M., Marchettini, N., Rossi, F.: Control of chemical chaos through medium viscosity in a batch ferroin-catalysed Belousov-Zhabotinsky reaction. Phys. Chem. Chem. Phys. **19**(48), 32235–32241 (2017). https://doi.org/10.1039/c7cp06601e
9. Cross, D.J., Gilmore, R.: Dressed return maps distinguish chaotic mechanisms. Phys. Rev. E **87**(1), 012919 (2013). https://doi.org/10.1103/physreve.87.012919
10. Ghrist, R.W., Holmes, P.J., Sullivan, M.C.: Knots and Links in Three-Dimensional Flows. Springer, Berlin (1997). https://doi.org/10.1007/bfb0093387
11. Gilmore, R.: Topological analysis of chaotic dynamical systems. Rev. Mod. Phys. **70**(4), 1455–1529 (1998). https://doi.org/10.1103/revmodphys.70.1455
12. Gilmore, R., Rosalie, M.: Algorithms for concatenating templates. Chaos: Interdisc J. Nonlinear Sci. **26**(3), 033102 (2016). https://doi.org/10.1063/1.4942799
13. Kumar, S., Strachan, J.P., Williams, R.S.: Chaotic dynamics in nanoscale NbO_2 Mott memristors for analogue computing. Nature **548**(7667), 318–321 (2017). https://doi.org/10.1038/nature23307
14. Larger, L., Penkovsky, B., Maistrenko, Y.: Laser chimeras as a paradigm for multistable patterns in complex systems. Nat. Commun. **6**(1), 7752 (2015). https://doi.org/10.1038/ncomms8752

15. Lorenz, E.N.: Deterministic nonperiodic flow. J. Atmos. Sci. **20**(2), 130–141 (1963). https://doi.org/10.1175/1520-0469(1963)020⟨0130:dnf⟩2.0.co;2
16. Malasoma, J.M.: What is the simplest dissipative chaotic jerk equation which is parity invariant? Phys. Lett. A **264**(5), 383–389 (2000). https://doi.org/10.1016/s0375-9601(99)00819-1
17. Melvin, P., Tufillaro, N.B.: Templates and framed braids. Phys. Rev. A **44**, R3419–R3422 (1991). https://doi.org/10.1103/PhysRevA.44.R3419
18. Mindlin, G.B., Hou, X.J., Solari, H.G., Gilmore, R., Tufillaro, N.B.: Classification of strange attractors by integers. Phys. Rev. Lett. **64**(20), 2350–2353 (1990). https://doi.org/10.1103/physrevlett.64.2350
19. Moitzi, M.: svgwrite (Python Library) (2018). https://pypi.org/project/svgwrite/. Accessed 26 May 2018
20. Olszewski, M., et al.: Visualizing the template of a chaotic attractor. arXiv preprint arXiv:1807.11853 (2018)
21. Rosalie, M.: Templates and subtemplates of Rössler attractors from a bifurcation diagram. J. Phys. A: Math. Theor. **49**(31), 315101 (2016). https://doi.org/10.1088/1751-8113/49/31/315101
22. Rosalie, M.: Chaotic mechanism description by an elementary mixer for the template of an attractor. arXiv preprint arXiv:1703.02768 (2017)
23. Rosalie, M., Danoy, G., Chaumette, S., Bouvry, P.: Chaos-enhanced mobility models for multilevel swarms of UAVs. Swarm Evol. Comput. **41**, 36–48 (2018). https://doi.org/10.1016/j.swevo.2018.01.002
24. Rosalie, M., Letellier, C.: Systematic template extraction from chaotic attractors: I. genus-one attractors with an inversion symmetry. J. Phys. A: Math. Theor. **46**(37), 375101 (2013). https://doi.org/10.1088/1751-8113/46/37/375101
25. Rosalie, M., Letellier, C.: Systematic template extraction from chaotic attractors: II. genus-one attractors with multiple unimodal folding mechanisms. J. Phys. A: Math. Theor. **48**(23), 235101 (2015). https://doi.org/10.1088/1751-8113/48/23/235101
26. Rössler, O.: An equation for continuous chaos. Phys. Lett. A **57**(5), 397–398 (1976). https://doi.org/10.1016/0375-9601(76)90101-8
27. Suzuki, Y., Lu, M., Ben-Jacob, E., Onuchic, J.N.: Periodic, quasi-periodic and chaotic dynamics in simple gene elements with time delays. Sci. Rep. **6**(1), 21037 (2016). https://doi.org/10.1038/srep21037
28. Tufillaro, N.B., Abbott, T., Reilly, J.: An Experimental Approach to Nonlinear Dynamics and Chaos. Addison-Wesley, Redwood City (1992)
29. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic HPC cluster: the UL experience. In: 2014 International Conference on High Performance Computing & Simulation (HPCS). IEEE (2014). https://doi.org/10.1109/hpcsim.2014.6903792