Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **200X**

doi:10.3934/xx.xx.xx.xx

pp. **X–XX**

EFFICIENT COMPUTATION OF LYAPUNOV FUNCTIONS FOR MORSE DECOMPOSITIONS

ARNAUD GOULLET, SHAUN HARKER, KONSTANTIN MISCHAIKOW

Rutgers University
110 Frelinghusen Road
Piscataway, NJ 08854, USA

WILLIAM D. KALIES, DINESH KASTI

Florida Atlantic University
777 Glades Road
Boca Raton, FL 33431, USA

ABSTRACT. We present an efficient algorithm for constructing piecewise constant Lyapunov functions for dynamics generated by a continuous nonlinear map defined on a compact metric space. We provide a memory efficient data structure for storing nonuniform grids on which the Lyapunov function is defined and give bounds on the complexity of the algorithm for both time and memory. We prove that if the diameters of the grid elements go to zero, then the sequence of piecewise constant Lyapunov functions generated by our algorithm converge to a continuous Lyapunov function for the dynamics generated the nonlinear map. We conclude by applying these techniques to two problems from population biology.

1. **Introduction.** This paper describes a computationally efficient approach to approximating Lyapunov functions for a nonlinear dynamical system generated by a continuous map $f\colon X \to X$ where X is a compact metric space. To emphasize both the potential generality and the computational challenges associated with this work we begin by recalling Conley's fundamental decomposition theorem [6, 21]. Recall that $A \subset X$ is an *attractor* for f if there exists a neighborhood U of A such that $A = \omega(U, f)$, i.e. A is the *omega limit set* of U under f [21]. Let $\mathrm{Att}(X, f)$ denote the set of attractors of f. Given $A \in \mathrm{Att}(X, f)$, its *dual repeller* is defined by

$$A^* := \{x \in X \mid \omega(x, f) \cap A = \emptyset\}\,.$$

2010 *Mathematics Subject Classification.* Primary: 37B25 ; Secondary: 37B30, 37B35, 37M99.

Key words and phrases. Lyapunov function, Morse decomposition, combinatorial dynamics, Conley's Decomposition Theorem, algorithms.

W.D. Kalies is partially supported by NSF grant NFS-DMS-0914995. S. Harker, A. Goullet and K. Mischaikow are partially supported by NSF grants NSF-DMS-0915019, 1125174, 1248071, and contracts from AFOSR and DARPA. .

Recall that $\mathcal{R}(f)$, the *chain recurrent set* of f, satisfies

$$\mathcal{R}(f) = \bigcap_{A \in \mathsf{Att}(X,f)} (A \cup A^*).$$

Conley's decomposition theorem guarantees the existence of a continuous function $V\colon X \to \mathbb{R}$ such that for all $t > 0$ and all $x \in X$, $V(f(x)) \leq V(x)$, and $V(f(x)) = V(x)$ if and only if $x \in \mathcal{R}(f)$. Moreover, $\mathcal{R}(f)$ is the smallest set for which a function with these properties can exist. Therefore, we refer to V as a *finest Lyapunov function* for f, but emphasize that in general there is no unique finest Lyapunov function. Thus the goal of this paper is to provide a computational algorithm which can approximate a finest Lyapunov function.

Conley's original proof [6] was recast as an algorithm in [16] and this algorithm then implemented in a computationally tractable manner in [3]. The algorithm we use in this paper is essentially the same, except that certain key computations are performed differently in order to incorporate the efficiencies needed for large computations. These more efficient methods stem from using the succinct grid data structure (Section 2), a modification of Tarjan's algorithm [11], and the lattice structure of attractors [17].

The starting point of Conley's proof is the realization that for any attractor $A \in \mathsf{Att}(X, f)$, there exists a continuous function

$$V_A\colon X \to [0, 1]$$

that is strictly decreasing for $x \in X \setminus (A \cup A^*)$ and satisfies $V_A(A) = 0$ and $V_A(A^*) = 1$. We refer to V_A as a *Lyapunov function for the attractor repeller pair* (A, A^*). Making use of the fact that $\mathsf{Att}(X, f)$ is at most countable, a finest Lyapunov function can be defined by

$$V(x) := \sum_{A_n \in \mathsf{Att}(X,f)} \beta_n V_{A_n}(x) \tag{1}$$

where $\beta_n > 0$ and $\sum \beta_n$ is bounded. In general, we will use the term *Lyapunov function for* f to mean any function of the form

$$V_{\mathsf{A}}(x) := \sum_{A_n \in \mathsf{A}} \beta_n V_{A_n}(x) \tag{2}$$

where A is a subset of $\mathsf{Att}(X, f)$.

This formula highlights the first computational issue that needs to be addressed. Namely, V_{A} cannot be explicitly represented when A is infinite. To deal with this we restrict our attention to certain finite sets of attractors that, as is shown later, arise naturally in computations.

Recall that $\mathsf{Att}(X, f)$ is a bounded distributive lattice [17]. Let A denote any bounded finite sublattice of $\mathsf{Att}(X, f)$. By [18, Corollary 5.13] the numerical methods described in this paper are, at least conceptually, capable of identifying A.

Define

$$\mathsf{M}_\mathsf{A} = \bigcap_{A \in \mathsf{A}} (A \cup A^*). \tag{3}$$

Observe that $\mathcal{R}(f) \subset \mathsf{M}_\mathsf{A}$. From the perspective of dynamics the most significant difference between a finest Lyapunov function V of the form defined in (1) and a function V_A of the form defined in (2) is that the latter may be constant on a larger set of orbits than V. Furthermore, if one chooses a nested sequence of sublattices that converges to $\mathsf{Att}(X, f)$, then V_A will converge monotonically to V.

This suggests that we approximate a finest Lyapunov function by approximating V_A. However, as is demonstrated in Section 5 it is not necessary to use all attractors in A to define a Lyapunov function for A that is constant on orbits in M_A. More precisely, using the lattice structure of A we prove in Theorem 5.2 that a Lyapunov function can be defined using only the join irreducible attractors.

The next computational issue that needs to be addressed is that of identifying a finite bounded sublattice $\mathsf{A} \subset \mathsf{Att}(X, f)$. This involves approximating the dynamical system f for which we make use of the structure of M_A. To explain this structure requires some standard concepts.

Given $x \in X$, consider a function $\gamma_x \colon \mathbb{Z} \to X$, satisfying $\gamma_x(0) = x$, and $\gamma_x(n+m) = \varphi(n, \gamma_x(m))$ for all $n \in \mathbb{Z}$ and all $m \in \mathbb{Z}^+$. Observe that given our assumptions on f, γ_x need not exist. However, if it does its image, also denoted by γ_x, is called a *complete orbit* through x. A set $S \subset X$ is *invariant* under φ if and only if for each $x \in S$ there exists a complete orbit $\gamma_x \subset S$. The restriction to $t \leq 0$ gives the *backward orbit* γ_x^-. For a point $x \in X$ and for a backward orbit γ_x^- the *orbital alpha-limit set* is defined as

$$\alpha_\mathrm{o}(\gamma_x^-) = \bigcap_{t \leq 0} \mathrm{cl}\left(\gamma_x\big((-\infty, t]\big)\right)$$

where cl denotes closure.

Definition 1.1. Let S be a compact invariant set for f. A finite collection $\mathsf{M} = \{M(p) \subset S \mid p \in \mathsf{P}\}$ of compact, pairwise disjoint, invariant subsets of S, labeled by the poset $(\mathsf{P}, \leq)$, is a *Morse decomposition in* S if for every $x \in S \setminus \left(\bigcup_{p \in \mathsf{P}} M(p)\right)$ and every orbit $\gamma_x \subset S$ there exist $p, q \in \mathsf{P}$ with $q < p$ such that

$$\alpha_\mathrm{o}(\gamma_x^-) \subset M(p) \quad \text{and} \quad \omega(x, f) \subset M(q).$$

Each set $M(p)$ is called a *Morse set*.

As defined by (3), M_A is a Morse decomposition [6], and V_A is constant precisely on each of the Morse sets. As is made clear below, we identify A by identifying a Morse decomposition.

Observe that, up to this point, the discussion applies to any dynamical system defined on a compact metric space X and that we have reduced to problem to one in which we need to approximate a continuous function $f \colon X \to X$. An efficient

universal means of approximating continuous functions on arbitrary compact metric spaces is probably not possible, thus at this point we restrict the breadth of problems being considered. In particular, for the remainder of this paper we assume that X is a rectangular subset of $\mathbb{R}^d$. The justification of this choice is twofold. First, a wide variety of models of dynamics, whether in the form of maps or differential equations, are framed in $\mathbb{R}^d$. In addition, many of the specific techniques employed in this paper have been extended to the setting of compact triangulable manifolds via chart maps, which again employ computations on rectangular subsets of $\mathbb{R}^d$ (e.g. the "Atlas" feature in the `Conley-Morse-Database` software [12]). Thus, from the perspective of applications working in $\mathbb{R}^d$ seems to be a reasonable restriction. Second, our approximation method utilizes the framework for computational global dynamics developed in a series of papers [16, 1, 4, 3, 18] and is based on an adaptive nonuniform discretization of X. For rectangular sets in $\mathbb{R}^d$, the construction of the discretization can be done straightforwardly via bisections resulting in a nonuniform rectangular grid denoted by $\mathcal{X}$.

Having established the goals and the generality of the results of this paper we now turn to an brief description of the contents. Section 2 describes the combinatorial approximation of the dynamics. This takes the form a directed graph $\mathcal{F}$ whose vertices are defined in terms of the rectangular grid $\mathcal{X}$ and whose edges provide bounds on the images of f. Our focus is on describing the data structures since this obviously impacts the size and dimension of problems that can be considered. In particular, we describe the "TreeGrid" data structure that is used to represent $\mathcal{X}$ and compute $\mathcal{F}$ and indicate how this is implemented using succinct binary trees. We also provide a brief discussion on the expected memory savings provided by this approach.

Section 3 describes the use of Tarjan's algorithm [24] to identify a linearly ordered collection of sets of grid elements $\mathcal{M}(p) \subset \mathcal{X}$, $p \in \mathsf{P}$ called *combinatorial Morse sets* with the fundamental property that the maximal invariant sets in $\mathcal{M}(p)$ determine a Morse decomposition for f. We also discuss the fact that for large applications because of the number of edges involved, storing $\mathcal{F}$ in memory is impractical. For this reason we employ a modified algorithm [11] for which the memory requirements are of the same order as the size of $\mathcal{X}$ while preserving the run time of Tarjan's algorithm.

Section 4 provides algorithms that take the combinatorial Morse sets as input and determines pairs of sets of grid elements $\mathcal{A}, \mathcal{A}^* \subset \mathcal{X}$ such that $\mathcal{A}$ approximates an attractor and $\mathcal{A}^*$ approximates the dual repeller.

Section 5 provides the theory and algorithms for the construction of a piecewise constant approximation of a Lyapunov function. We begin by recalling the relationship between the partial order structure of Morse decompositions and the lattice structure of attractors that is used to provide a proof of Theorem 5.2 cited above. We then turn to the construction of an approximate Lyapunov function

based on a combinatorial attractor repeller pair $(\mathcal{A}, \mathcal{A}^*)$. To be more precise about the form of this approximate Lyapunov function, we introduce the following definition.

Definition 1.2. Given a combinatorial attractor repeller pair $(\mathcal{A}, \mathcal{A}^*)$ a function $V_{\mathcal{A}} \colon \mathcal{X} \to [0,1]$ is a *combinatorial Lyapunov function* for f if it satisfies the following conditions:

1. $\mathcal{A} = V_{\mathcal{A}}^{-1}(0)$ and $\mathcal{A}^* = V_{\mathcal{A}}^{-1}(1)$;
2. for every $x \in X$, if $x \in |\xi|$ for some $\xi \in \mathcal{X}$ and $f(x) \in \xi' \in \mathcal{X}$, then $V_{\mathcal{A}}(\xi') \leq V_{\mathcal{A}}(\xi)$ and $V_{\mathcal{A}}(\xi') = V_{\mathcal{A}}(\xi)$ if and only if $\xi \in \mathcal{A} \cup \mathcal{A}^*$.

A key step in making the construction of combinatorial Lyapunov functions practical is the introduction of efficient algorithms for evaluating the distance between points in X and the sets $\mathcal{A}$ and $\mathcal{A}^*$. These algorithms culminate in Proposition 5.7 which provides explicit upper bounds on the memory and time costs for the computation of the combinatorial Lyapunov function based on the dimension of the problem and the approximation associated with the combinatorial Morse decomposition $\{\mathcal{M}(p) \subset \mathcal{X} \mid p \in \mathsf{P}\}$.

It should be noted that combinatorial Lyapunov functions follow directly from the algorithms described in Section 3. Unfortunately, as one refines the grid $\mathcal{X}$ and the approximation of the dynamics $\mathcal{F}$, in general, these combinatorial Lyapunov functions will not converge to Lyapunov functions for f. Thus, from our perspective they should not be viewed as approximate Lyapunov functions for f. However, as we make precise at the end of Section 5 the construction we perform leads to combinatorial Lyapunov functions that converge to a Lyapunov function of f as the approximation of $\mathcal{X}$ and $\mathcal{F}$ are refined.

Section 6 provides results on the implementation of these techniques to two population models. These models are chosen because they exhibit complex recurrent dynamics with multiple basins of attraction.

2. Combinatorial representation of dynamics.

As stated in the introduction, we are interested in computing approximate Lyapunov functions for the discrete time dynamical system obtained by iterating a map $f \colon X \to X$ with X a compact rectangular subset of $\mathbb{R}^d$. Note that f need not be injective nor surjective. The computational method utilizes the framework for computational global dynamics developed in a series of papers [16, 1, 4, 3, 18] as described in the introduction. First, the phase space is discretized using a finite, rectangular subdivision of X into a union of closed subboxes which have potential overlap only on their boundaries, the collection of which is called a *grid* and denoted by $\mathcal{X}$. We use the notation $|\cdot| \colon 2^{\mathcal{X}} \to \mathbb{R}^d$ to denote the realization map for sets of grid elements given by $|\mathcal{S}| = \cup_{\xi \in \mathcal{S}} \xi$. The dynamics is then approximated by a combinatorial multivalued

map $\mathcal{F}\colon \mathcal{X} \rightrightarrows \mathcal{X}$ which maps grid elements to sets of grid elements. A combinatorial map $\mathcal{F}$ can be represented as a directed graph whose vertices are the grid elements in $\mathcal{X}$ and which has an edge from ξ to η if and only if $\eta \in \mathcal{F}(\xi)$.

The concept of an outer approximation as a method for discretizing the phase space and approximating a discrete dynamical system was originally introduced in [23].

Definition 2.1. Let $f\colon X \to X$ be a continuous map. A multivalued mapping $\mathcal{F}\colon \mathcal{X} \rightrightarrows \mathcal{X}$ is an *outer approximation* if $f(\xi) \subset \operatorname{int}|\mathcal{F}(\xi)|$ for all $\xi \in \mathcal{X}$ where int denotes interior.

The fundamental property of an outer approximation $\mathcal{F}$ is that an orbit of the underlying dynamical system f is contained in grid elements corresponding to a walk through the directed graph of $\mathcal{F}$, in other words for every orbit $\{x_n\}$ with $x_{n+1} = f(x_n)$ there exists a sequence of grid elements $\{\xi_n\}$ with $\xi_{n+1} \subset \mathcal{F}(\xi_n)$ such that $x_n \subset \xi_n$ for all n.

In order to store a grid $\mathcal{X}$ and an outer approximation $\mathcal{F}$ in an efficient manner for large computations, appropriate data structures are necessary, and we must also take into account the efficiency of the operations on $\mathcal{X}$ and $\mathcal{F}$ that are required to perform the algorithms. Since one of the main goals of this paper is to explore the overall capability of computing Lyapunov functions, we now describe some of these implementation issues in some detail.

Let **Rect** denote the collection of grid-aligned rectangular prisms (i.e. products of intervals) in $\mathbb{R}^d$. In order to construct a suitable $\mathcal{F}$ and $\mathcal{X}$ given a formula for f on X, we require a data structure to represent $\mathcal{X}$ which provides the following functionality:

Grid Data Structure Interface

1. **Cover:** given $R \in \mathbf{Rect}$, produce a collection of grid elements containing $\{\xi \in \mathcal{X} \mid \xi \cap R \neq \varnothing\}$.
2. **Geometry:** given a grid element $\xi \in \mathcal{X}$, produce $R \in \mathbf{Rect}$ such that $\xi \subset R$.

Note that both of the methods $\mathbf{Cover} : \mathbf{Rect} \rightrightarrows \mathcal{X}$ and $\mathbf{Geometry} : \mathcal{X} \to \mathbf{Rect}$ allow for the possibility of overestimating. This allows for a trivial implementation where **Cover** returns all of $\mathcal{X}$ and **Geometry** returns a bounding box for X. Such a trivial implementation will result in correct, yet trivial results. Tighter enclosures are better. Topological structure is captured by

$$\mathbf{Neighbors}(\xi) := \{\eta \in \mathcal{X} \mid \xi \cap \eta \neq \varnothing\}.$$

Observe that for each $\xi \in \mathcal{X}$, $\mathbf{Neighbors}(\xi) \subset (\mathbf{Cover} \circ \mathbf{Geometry})(\xi)$ The grids we will consider will satisfy the tightness condition

$$\mathbf{Neighbors} = \mathbf{Cover} \circ \mathbf{Geometry}, \tag{4}$$

which precludes trivial implementations.

Assuming we can implement a function $F\colon \mathbf{Rect} \to \mathbf{Rect}$ such that

$$f(R) \subset F(R) \text{ for all } R \in \mathbf{Rect},$$

we can produce a combinatorial map $\mathcal{F}$ which is an outer approximation (Definition 2.1) via the composition

$$\mathcal{F} := \mathbf{Cover} \circ F \circ \mathbf{Geometry}. \tag{5}$$

In practice one can usually implement F via rewriting the formula for f using interval arithmetic. We do note, however, that sometimes this procedure results in very large bounding boxes. In fact we can be somewhat more flexible and introduce an enlarged collection of acceptable geometric shapes $\mathbf{Geo}$ in $\mathbb{R}^d$ such that $\mathbf{Rect} \subset \mathbf{Geo}$. Then we can let $\mathbf{Cover} : \mathbf{Geo} \rightrightarrows \mathcal{X}$, $\mathbf{Geometry} : \mathcal{X} \to \mathbf{Rect}$, and $F\colon \mathbf{Rect} \to \mathbf{Geo}$. This allows the images of $F(R)$ to be tighter around the true image $f(R)$ since they can be chosen from a more refined family of possibilities. For example, we could choose $\mathbf{Geo}$ to be sets which can be constructed from a finite union of $\mathbf{Rect}$ objects. Or we could let $\mathbf{Geo}$ consist of all parallelepipeds. However, in the interests of simplicity, for this paper we will simply take $\mathbf{Geo} = \mathbf{Rect}$.

2.1. **Binary Tree Based Grids.** We implement grids using a binary tree structure. For simplicity we take the phase space X to be a $\mathbf{Rect}$ object itself. We take the root of the binary tree to correspond to the $\mathbf{Rect}$ object X. We choose a *splitting dimension*, and subdivide the **Rect** object corresponding to the root node into two congruent **Rect** objects via a hyperplane bisection. We take these two **Rect** objects to correspond to the two children nodes. The left child will correspond to the **Rect** object with coordinates below the hyperplane, and vice-versa for the right child. We continue this procedure as we please to make a binary tree of any shape. See Figure 1. We impose the constraint that the splitting dimension is a function of the depth alone; in our implementation the subdivision is done round-robin through the dimensions so that the splitting dimension at depth k is k mod d.

The binary tree one constructs via this procedure corresponds to a grid, where the grid elements are taken to be the leaves of this tree. Notice that this binary tree is *full*, meaning that a node is either a leaf or has precisely two children. In a computer implementation one must refer to these leaves/grid elements in some fashion, so we use contiguous integers $(0, 1, 2, \cdots, N-1)$, where N is the total number of grid elements. We label them according to the order the leaves are visited in a preorder traversal (the ordered depth-first-search of the tree starting at

the root). We will call this *leaf indexing*. In the course of devising algorithms, we also need names for the nodes of the tree as well. For this purpose we will again use the order in which the nodes are visited in a preorder traversal, so the nodes are labelled $(0, 1, \cdots, M-1)$. We will call this *node indexing*. Note that a leaf has both a leaf index and a node index, which need not be equal, and for a full binary tree we must have $M = 2N - 1$.

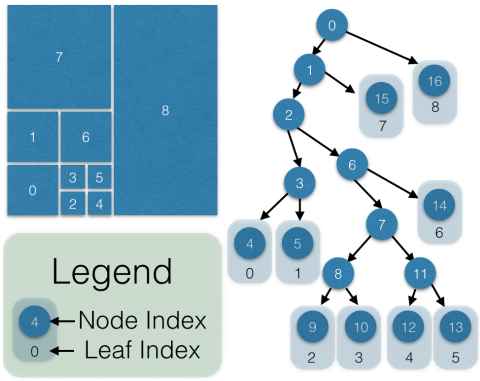

FIGURE 1. This figure illustrates a correspondence between Grid subdivisions and binary trees, as well as the node and leaf indexing scheme we use.

We will assume the binary tree data structure provides the following functionality:

TreeGrid Data Structure Interface

1. **parent**, **left**, **right**. Given the node index for a node, return the node index for its parent, left child, or right child (respectively). If there is no such node (i.e. the root has no parent and leaves have no left or right child) then return M.
2. **GridToTree**. Given a leaf index, return the node index of the corresponding leaf.
3. **TreeToGrid**. Given a node index, give the leaf index of the corresponding node if the node is a leaf. If the node is not a leaf, return N.
4. **bounds**. Return the **Rect** object corresponding to the root node.

Proposition 2.2. *The TreeGrid interface can be used to implement a Grid satisfying the tightness condition (4).*

Proof. We give implementations of **Geometry** and **Cover**. For **Geometry**, we can determine the **Rect** object for a grid element, which will be referred to by its leaf index, by calling **GridToTree**, calling **parent** repeatedly to find the path to root,

and along the way checking if we came from the left or right child by calling **left** and **right** and comparing the node indices. This results in path-to-root information, which tells us a sequence of subdivisions which when applied to **bounds** will result in the correct **Rect** object to return.

Cover can also be implemented efficiently using the binary-tree structure. When presented with a **Rect** object to cover, we begin by seeing if it intersects with the **Rect** corresponding to the root of the tree. If it does intersect, we recurse and ask this question for both of its children. If it does not intersect, we do nothing and continue on with other branches. When we find a leaf node that intersects, we call **TreeToGrid** and place the appropriate leaf index in the list of results for **Cover**.

Provided the numerics used in handling computer representations of **Rect** objects, which involve floating point representations, are performed with care, it is straightforward to see the tightness condition (4) can be achieved. ∎

2.2. **Succinct Binary Tree Grid Implementation.** One way of implementing a data structure respecting the TreeGrid interface is to create a binary tree structure by creating "Node" objects in memory that contain pointers to parent, left child, right child, and an integer containing the node index. We could use a size M array to store pointers to these Nodes. GridToTree and TreeToGrid could be implemented by storing arrays of integers of length N and M, respectively. The space usage of such an implementation would be $6M + N \approx 13N$ words of storage. A *word* of storage is the amount of computer memory used to store a pointer. In this era, it is usually 64 bits. Hence this results in a data structure that requires roughly 104 bytes per grid element.

The pointer-based method is quite reasonable in many situations. Its drawback is its space usage. At the expense of an up-front constant in time usage, it is possible to substantially bring down the space requirements by using methods from the succinct data structures literature, in particular *succinct balanced parentheses lists* and *rank-select structures* [20, 5, 13]. Specifically, it is possible to construct a succinct TreeGrid data structure providing the required methods in constant time and using only $2N + o(N)$ *bits* of space. Ignoring the $o(N)$ overhead, this means that for 64-bit words the succinct methods are roughly $(13N \cdot 64)/2N = 416$ times as space-efficient as a pointer-based method.

We describe how we accomplish this. First, we review the succinct data structures at our disposal. The key features of a succinct balanced parentheses list are that they require only $2n + o(n)$ bits of space to store n pairs of matching parentheses, and they allow the operations of **findopen** and **findclose**, which find opening and closing matching parenthesis, respectively, in constant $O(1)$ time. Meanwhile, a rank-select structure is a succinct data structure that requires $n + o(n)$ bits of space and is capable of answering **rank** and **select** queries about an n bit sequence in $O(1)$ time. A **rank**(i) query asks how many 1 bits occur strictly before some

position $0 \leq i \leq n$ in the sequence. A $\mathbf{select}(i)$ query asks for the maximal j such that $\mathbf{rank}(j) = i$. We also assume the ability to access the underlying bit sequence in constant time.

We utilize these structures in the following way. Given a full binary tree with N leaves, and hence $M := 2N - 1$ nodes, we construct an M bit sequence we call the *leaf sequence* ℓ such that $\ell[k] = 1$ if and only if the node with node index k is a leaf. A leaf sequence completely and succinctly characterizes a full binary tree. Note that the last bit in the leaf sequence is a 1 so that the preorder traversal ends at a leaf. Hence the first $M-1$ bits of the leaf sequence has an equal number of 0's and 1's. In fact, interpreting a 0 (non-leaf) as an opening parenthesis and a 1 (leaf) as a closing parenthesis, the first $M - 1$ bits are a balanced parentheses list. Accordingly, we can build a rank-select query structure and a balanced parentheses query structure on top of this for an additional $o(M)$ space.

Proposition 2.3. *Let T be a full binary tree with M nodes. The queries **findopen**, **findclose** on the M-bit leaf sequence can be used to implement **parent**, **left**, and **right** via the following formulas, where we refer to nodes via their node index:*

$$\textbf{\textit{left}}(x) = \begin{cases} x+1 & \text{if } \ell[x] = 0 \text{ i.e. } x \text{ is not a leaf} \\ m \text{ (undefined)} & \text{otherwise} \end{cases} \tag{6}$$

$$\textbf{\textit{right}}(x) = \begin{cases} \mathbf{findclose}(x) + 1 & \text{if } \ell[x] = 0 \text{ i.e. } x \text{ is not a leaf} \\ m \text{ (undefined)} & \text{otherwise} \end{cases} \tag{7}$$

$$\textbf{\textit{parent}}(x) = \begin{cases} \text{undefined} & \text{if } x = 0 \text{ (i.e. } x \text{ is the root)} \\ x-1 & \text{if } \ell[x-1] = 0 \text{ (i.e. } (x-1) \text{ is not a leaf)} \\ \mathbf{findopen}(x-1) & \text{otherwise} \end{cases} \tag{8}$$

*Moreover, the queries **rank** and **select** on the leaf sequence implement **TreeToGrid** and **GridToTree**, respectively.*

Proof. The key idea is that each pair of matching parentheses in the leaf sequence corresponds to a pairing between a tree node x (the open parenthesis) and the last preordered leaf y on x's left subtree (the closing parenthesis). This is very useful since the next node preordered after the left subtree of x, namely $y + 1$, is the right child of x. This yields the formulas. The final statement follows directly from the definitions. □

Proposition 2.3 implies we can construct the succinct "TreeGrid" data structure requiring only the $2N + o(N)$ bits of space advertised above. We should emphasize that this is by no means the only possible construction to achieve these ends. For example, the pioneering paper of Jacobson [13] describes a method using rank-select on *level-order bitmaps* that is suitable to binary trees such as ours that could

achieve $4N + o(N)$ bits. Our construction (which we presume is not original) is interesting because it is ultra-succinct [14], that is, it beats the $4N+o(N)$ information-theoretic bounds for representing general binary trees by utilizing special structure (namely, fullness).

In the computation of combinatorial Morse sets, we also have occasion to deal with subgrids, and the strategy our implementation uses is to wrap the above data structure with a bit-sequence which "knocks-out" extra leaves from a full binary tree to achieve an arbitrary binary graph. For an arbitrary binary tree with M nodes, this results in at worst $2M+o(M)$ bits for a full tree containing the arbitrary binary tree, and another $M + o(M)$ bits to represent a rank/select structure on the knock-out sequence. This does not qualify as succinct; we would need to achieve $2M + o(M)$ bits worst-case, not $3M + o(M)$ bits in the worst-case. However, in practice the subgrids that arise in computational dynamics do not approach this worst case, because nodes with only one child tend to be rare. Accordingly, we consider the best case, where the tree is "almost full". In this situation one sees our strategy achieves roughly $1.5M + o(M)$ bits. Hence despite not technically being succinct in the worst case, in the best case, which we claim will be the average case in practice, we obtain ultra-succinct performance. We discuss the performance of our implementation, which is based on the Succinct Data Structures Library [10], in Section 6.

3. **Extracting recurrent and gradient-like dynamics.** Since we do not assume that the continuous map $f\colon X \to X$ generating the dynamics is injective or surjective, X need not be invariant. We compute Morse decompositions with respect to the maximal invariant set in X,

$$S = \mathrm{Inv}(X, f) = \{x \mid \text{there exists a complete orbit } \{x_n\}_{n=-\infty}^{\infty} \text{ with } x_0 = x\}.$$

As we described in the previous section, orbits of a dynamical system are rigorously contained in walks through the directed graph generated by an outer approximation of that system. In particular this implies that if an orbit $\{x_n\}$ is chain recurrent, then it is contained in a periodic cycle in the directed graph, i.e. there exists a *periodic* sequence $\{\xi_n\}$ with $\xi_{n+1} \subset \mathcal{F}(\xi_n)$ such that $x_n \subset \mathcal{F}(\xi_n)$ for all n. Therefore, the problem of computing a neighborhood of the chain recurrent set can be reduced to finding the set of cyclic vertices in a directed graph, which we will call the *recurrent set of* $\mathcal{F}$.

Analogous to the chain recurrent set of a dynamical system, the recurrent set of a graph is naturally partitioned by the reachability relation. That is, the equivalence relation $\xi \sim \eta$, if there exists a walk from ξ to η and a walk from η to ξ in $\mathcal{F}$, partitions the recurrent set of $\mathcal{F}$ into its *recurrent components*, which are also known as the strongly connected path components of the digraph. These recurrent components have a partial order induced by the reachability relation in the digraph of $\mathcal{F}$. Corollary 4.2 in [16] implies that once we have computed the recurrent set of $\mathcal{F}$,

the geometric realization of the recurrent components are isolating neighborhoods of a Morse decomposition with the same partial order, i.e. we can rigorously extract a Morse decomposition from an outer approximation $\mathcal{F}\colon \mathcal{X} \rightrightarrows \mathcal{X}$. Due to this connection with a Morse decomposition, we denote the recurrent components by $\mathcal{M}$. We emphasize that the recurrent components do not partition the digraph of $\mathcal{F}$, since elements in the recurrent set must be cyclic through a nontrivial walk in $\mathcal{F}$. In contrast, the strongly connected components do partition the digraph. Each recurrent component is a strongly connected component. However, each noncyclic vertex is also itself a strongly connected component but is not a recurrent vertex.

The classical algorithm for extracting strongly connected components of a directed graph given by its adjacency lists (lists of target vertices given a source vertex), is called *Tarjan's algorithm* [24]. For a graph with V vertices and E edges, Tarjan's algorithm requires $O(E+V)$ time and $O(E+V)$ space to identify the collection of strongly connected components $\mathcal{S}$. One identifies the recurrent components M as the strongly connected components containing at least one edge; i.e. components with more than one vertex as well as the single vertex components with self loops.

Tarjan's algorithm outputs the strongly connected components in a useful order. Recall that any directed acyclic graph admits a *topological sort*, which is a total ordering of the vertices such that $a < b$ only if b cannot reach a. This concept applies to strongly connected components of a directed graph since collapsing these components to single vertices renders the directed graph into an acyclic one, the so-called *condensation graph.* The order in which Tarjan's algorithm produces strongly connected components will correspond to a reverse topological sort of the condensation graph. We will make use of this property later, when we describe algorithms that require iterating through strongly connected components $\mathcal{S}$ or recurrent components $\mathcal{M}$ in the order of a topological sort or reverse topological sort.

We point out that the $O(E)$ space bounds arise from storage of the adjacency lists and could be quite large. Indeed, E could be as large as V^2 in worst case. Thus the space requirements for Tarjan's algorithm could dominate over the space requirements of the "Grid" data structure. Observe that size of the adjacency lists computed via Equation (5) are easily bounded from below by the product of the eigenvalues of Df that are larger than one. In general it will be considerably larger. In other words, storage of the graph can be quite expensive.

There is a solution to this problem. A modified version of Tarjan's algorithm, due to one of the authors and motivated by precisely this application, circumvents the need to store the adjacency lists and provides an $O(V)$ space bound [11]. This algorithm requires that adjacency lists can be retrieved or computed on demand. Moreover, during execution it only asks for each adjacency list precisely once. Consequently it runs in time comparable to Tarjan's algorithm. These properties make it well suited for computing strong components of the directed graph arising from

the combinatorial map $\mathcal{F}$ of Equation (5). This algorithm has been implemented and tested, and is publicly available in `Conley-Morse-Database` [12].

We have also recently become aware of the strong components algorithm of [15], which assumes adjacency lists are available in external memory and attempts to minimize I/Os. While it was not the author's intention, in fact it can be reinterpreted as an algorithm for computing strongly connected components which requires only $O(V)$ space provided one is given the ability to compute adjacency lists on demand. Detailed comparisons of up-front constants between these alternatives will be made in [11].

4. **Attractor-repeller pairs.** In the previous section we observed that graph theoretic techniques provide a naturally efficient way to obtain an approximate Morse decomposition from an outer approximation. To recover an approximate Lyapunov function from this information, we need to describe the relationship between Morse decompositions and attractor-repeller pairs. We assume that the reader is familiar with basic concepts of limit sets, attractors, and repellers in the context of a dynamical system. However, since we are working with semidynamical systems that are not necessarily invertible, we must address some subtleties that do arise; also we must describe these concepts in the context of combinatorial multivalued maps.

Since X need not be invariant, A^* also need not be invariant. The phase space X is only forward invariant in general, and A^* is the maximal forward invariant set in the complement of an attracting neighborhood U of A. For combinatorial multivalued maps $\mathcal{F}$ define the $\boldsymbol{\omega}$*-limit set* and $\boldsymbol{\alpha}$*-limit set* of a set $\mathcal{U} \subset \mathcal{X}$ by

$$\boldsymbol{\omega}(\mathcal{U}) = \bigcap_{k\geq 0} \bigcup_{n\geq k} \mathcal{F}^n(\mathcal{U}) \quad \text{and} \quad \boldsymbol{\alpha}(\mathcal{U}) = \bigcap_{k\leq 0} \bigcup_{n\leq k} \mathcal{F}^n(\mathcal{U}).$$

Definition 4.1. Let $\mathcal{F}\colon \mathcal{X} \rightrightarrows \mathcal{X}$. A set $\mathcal{A} \subset \mathcal{X}$ is an *attractor* for $\mathcal{F}$ if $\mathcal{F}(\mathcal{A}) = \mathcal{A}$. A set $\mathcal{R} \subset \mathcal{X}$ is a *repeller* for $\mathcal{F}$ if $\mathcal{F}^{-1}(\mathcal{R}) = \mathcal{R}$. The *dual repeller* $\mathcal{A}^*$ to an attractor $\mathcal{A}$ is defined by $\mathcal{A}^* = \boldsymbol{\alpha}(\mathcal{X} \setminus \mathcal{A})$. Given an attractor $\mathcal{A}$, the pair $(\mathcal{A}, \mathcal{A}^*)$ is called an *attractor-repeller* pair for $\mathcal{F}$. The set of all attractors is denoted by $\mathsf{Att}(X, \mathcal{F})$.

An attractor-repeller pair is a Morse decomposition with two Morse sets, the attractor and its dual repeller, with the partial order where the repeller is greater than the attractor. Just as in the case of Morse decompositions, [16, Proposition 5.5] implies that if $\mathcal{F}$ is an outer approximation, then the realizations $(|\mathcal{A}|, |\mathcal{A}^*|)$ of an attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$ are attracting and repelling neighborhoods respectively for some attractor-repeller pair (A, A^*) for f.

Recall that in Section 3 we describe an efficient algorithm to obtain the following information from the digraph representing $\mathcal{F}$:

1. the recurrent components $\mathcal{M} = \{M(p) \mid p \in \mathsf{P}\}$,
2. the strongly connected components $\mathcal{S}$, and

3. a topological sort of $\mathcal{S}$.

Given a recurrent component $M \in \mathcal{M}$, define $\Gamma^+(M)$ to consist of all grid elements in $\mathcal{X}$ reachable from M. Since $\mathcal{F}(\Gamma^+(M)) = \Gamma^+(M)$, $\Gamma^+(M) \in \mathsf{Att}(\mathcal{F})$. We show in Section 5 that the information in the attractors $\{\mathcal{A}_p := \Gamma^+(M(p)) \mid p \in \mathsf{P}\}$ are sufficient to produce a combinatorial Lyapunov function. Consequently, we now give an algorithm to compute the combinatorial attractor-repeller pairs of this form using only the information from the connected components and topological sort.

Algorithm 1 Computing Attractor/Dual Repeller Pairs

Global variables:
A list $\mathcal{M}$ of the combinatorial Morse sets, a topologically sorted list $\mathcal{S}$ of strongly connected components, and an array SCC indicating which strongly connected component each grid element is in.

AttractorRepellerPairs
for $M \in \mathcal{M}$ **do**
  $\mathcal{A} \leftarrow$ **ComputeAttractor**(M).
  $\mathcal{A}^* \leftarrow$ **ComputeDualRepeller**$(\mathcal{A})$.
end for

ComputeAttractor
Given: $M \in \mathcal{M}$
Return: Minimal attractor containing M
Create an empty set $\mathcal{U}$
Insert M into $\mathcal{U}$.
$\mathcal{A} \leftarrow$ **ForwardReachable**$(\mathcal{U})$
Return $\mathcal{A}$.

ComputeDualRepeller
Given: Combinatorial attractor $\mathcal{A} \subset \mathcal{S}$
Return: Combinatorial dual repeller of $\mathcal{A}$
Create an empty set $\mathcal{U}$
for $M \in \mathcal{M}$ **do**
  if $M \notin \mathcal{A}$ **then**
    Insert M into $\mathcal{U}$.
  end if
end for
$\mathcal{A}^* \leftarrow$ **BackwardReachable**$(\mathcal{U})$
Return $\mathcal{A}^*$.

Global variables:
A topologically sorted list $\mathcal{S}$ of strongly connected components, and an array SCC indicating which strongly connected component each grid element is in.

ForwardReachable:
Given: $\mathcal{U} \subset \mathcal{S}$
Return: $\{S \in \mathcal{S} \mid \exists T \in \mathcal{U},\ T \text{ reaches } S\}$
$\mathcal{A} \leftarrow \mathcal{U}$.
for $S \in \mathcal{S}$ (in order) **do**
  if $S \in \mathcal{A}$ **then**
    for $u \in S$ **do**
      for $v \in \mathcal{F}(u)$ **do**
        $T \leftarrow SCC[v]$.
        Insert T into $\mathcal{A}$.
      end for
    end for
  end if
end for
Return $\mathcal{A}$

BackwardReachable:
Given: $\mathcal{U} \subset \mathcal{S}$
Return: $\{S \in \mathcal{S} \mid \exists T \in \mathcal{U},\ S \text{ reaches } T\}$.
$\mathcal{A}^* \leftarrow \mathcal{U}$.
for $S \in \mathcal{S}$ (in *reverse* order) **do**
  if $S \notin \mathcal{A}^*$ **then**
    for $u \in S$ **do**
      for $v \in \mathcal{F}(u)$ **do**
        if $SCC[v] \in \mathcal{A}^*$ **then**
          Insert S into $\mathcal{A}^*$.
        end if
      end for
    end for
  end if
end for
Return $\mathcal{A}^*$.

Proposition 4.2. *Given the recurrent components $\mathcal{M} = \{M(p) \mid p \in \mathsf{P}\}$, Algorithm (1) computes all combinatorial attractor-repeller pairs of the form*

$$\left\{(\mathcal{A}_p, \mathcal{A}_p^*) \mid \mathcal{A}_p = \Gamma^+(M(p)) \text{ and } p \in \mathsf{P}\right\}$$

in time $O(\mathrm{card}(\mathsf{P}) \cdot E)$, where E is the number of edges in the directed graph corresponding to the combinatorial map $\mathcal{F}$ of Equation (5).

Proof. The correctness of **ComputeAttractor** requires showing that it computes $\omega(M) = \Gamma^+(M)$ for a recurrent set $M \in \mathcal{M}$. This is the case provided **ForwardReachable** is correct, since it promises to provide all strongly connected components reachable from M. Correctness of **ComputeDualRepeller** requires that we show it computes $\alpha(\mathcal{X} \setminus \mathcal{A})$. It is straightforward to verify that $\alpha(\mathcal{X} \setminus \mathcal{A})$ is the subset of grid elements which can reach some recurrent set M in $\mathcal{X}$ outside of $\mathcal{A}$. These grid elements comprise the strongly connected components which can reach recurrent sets M which are not in $\mathcal{A}$. Hence the algorithm for **ComputeDualRepeller** is correct assuming the correctness of **BackwardReachable**.

Next, we show **ForwardReachable** is correct and executes in $O(E)$ time, which follows from that disjointness of strongly connected components. Indeed we have

that `Adjacencies`(u), which takes $O(\text{card}(\mathcal{F}(u))$ time, is called at most once per vertex $u \in \mathcal{X}$. Moreover, "Insert $SCC[v]$ into $\mathcal{A}$", which takes $O(1)$ time, is executed at most $\text{card}(\mathcal{F}^{-1}(u))$ times per vertex $u \in \mathcal{X}$. This counts each edge of the directed graph at most twice, so that the sum of this time can thus be bounded by a constant factor times the number of edges of the directed graph corresponding to $\mathcal{F}$, which gives a total complexity of $O(E)$ time. Correctness of **ForwardReachable** follows from induction given (1) that $\mathcal{S}$ is processed in topologically sorted order and (2) the observation that if a strongly connected component S belongs to the forward reachable set of $\mathcal{A}$, then either $S \in \mathcal{U}$ or there exists a strongly connected component T which directly reaches S, that is, a vertex of S is in the adjacency list of a vertex in T, and hence T necessarily occurs before S in the topological sort of $\mathcal{S}$. A similar argument shows that **BackwardReachable** is correct and computes in worst case $O(E)$ time.

Since $\text{card}(\mathsf{P}) < E$, it follows that the complexities of **ComputeAttractor** and **ComputeDualRepeller** are dominated by the complexities of their calls to **ForwardReachable** and **BackwardReachable**, and hence are both $O(E)$ as well. Since **AttractorRepellerPairs** calls **ComputeAttractor** and **ComputeRepeller** $\text{card}(\mathsf{P})$ times, its complexity is bounded by $O(\text{card}(\mathsf{P}) \cdot E)$, as claimed. □

5. **Lyapunov functions.** Algorithm (1) takes Morse sets in a Morse decomposition and produces a collection of attractors. However, there exists a deeper structural relationship between Morse sets and attractors that we now describe. We use this relationship to restrict the set of attractors used to define a Lyapunov function for the Morse decomposition. As discussed in the Introduction, this Lyapunov function is defined in terms of Lyapunov functions for attractor repeller pairs. Thus we address the algorithmic issues of constructing a combinatorial Lyapunov function in this limited setting, before extending it to the full Morse decomposition.

5.1. **A Lyapunov function for Morse decompositions.** We continue to study the dynamics generated by a continuous map $f\colon X \to X$. Throughout this section we set $S := \text{Inv}(X, f)$. For the remainder of this subsection we restrict our attention to $f\colon S \to S$. Since we are focussing, for the most part, on the relationship between attractors and Morse decompositions and since $\mathsf{Att}(X, f) = \mathsf{Att}(S, f)$, this is not a serious restriction.

As indicated in the Introduction $\mathsf{Att}(S, f)$ is a bounded distributive lattice. To be more precise given $A, A' \in \mathsf{Att}(S, f)$ the lattice operations are defined by

$$A \vee A' = A \cup A' \quad \text{and} \quad A \wedge A' = \omega(A \cap A', f)$$

and the minimal and maximal elements are $0 = \emptyset$ and $1 = \text{Inv}(X, f)$.

Let $\mathsf{M} = \{M(p)\}$ be a Morse decomposition of S labelled by the poset $(\mathsf{P}, \leq)$.

Given a finite poset $(\mathsf{P}, \leq)$, the set of all *down sets of* P is defined by

$$\mathsf{O}(\mathsf{P}) = \{\gamma \subset \mathsf{P} \mid p \in \gamma \text{ implies } r \in \gamma \text{ for all } r \leq p\}.$$

A direct calculation shows that $\mathsf{O}(\mathsf{P})$ is a finite, distributive lattice under the operations of union and intersection. In [19] it is shown that the map $\mu\colon \mathsf{O}(\mathsf{P}) \to \mathsf{Att}(S, f)$ defined by

$$\mu(\gamma) = \bigcup_{q\in\gamma} W^u(M(q))$$

is a lattice embedding, where $W^u(M)$ is the unstable set of M. The image $\mathsf{A} = \mu(\mathsf{O}(\mathsf{P}))$ is then a finite bounded sublattice of $\mathsf{Att}(S, f)$ which is isomorphic to $\mathsf{O}(\mathsf{P})$. In this way the attractor A is determined uniquely from the Morse decomposition M and the partial order $(\mathsf{P}, \leq)$.

We are interested in Lyapunov functions that are compatible with Morse decompositions which leads to the following definition.

Definition 5.1. Let M be a Morse decomposition for $S = \mathrm{Inv}(X)$, labeled by a poset $(\mathsf{P}, \leq)$, i.e. $\mathsf{M} = \{M(p) \mid p \in \mathsf{P}\}$. A continuous function $V : X \to [0, 1]$ which satisfies

(i) for every $p \in \mathsf{P}$ there is $c_p \in [0, 1]$ such that $M(p) \subset V^{-1}(c_p)$,
(ii) if $q < p$, then $c_q < c_p$, and
(iii) if $x \in X \setminus \bigcup_{p\in\mathsf{P}} M(p)$, then $V(f(x)) < V(x)$,

is called a *Lyapunov function for* $(X, \mathsf{M}, \mathsf{P}, \leq)$.

From the definition of Morse decomposition, one can check that for every attractor $A \in \mathsf{A}$ each Morse set is contained in either A or A^*, and in addition

$$\bigcap_{A\in\mathsf{A}} (A \cup A^*) = \bigcup_{p\in\mathsf{P}} M(p).$$

Following the original ideas of Conley, for any set of weights $\beta_A > 0$ with $\sum_{A\in\mathsf{A}} \beta_A = 1$ and any choice of Lyapunov functions V_A for attractor-repeller pairs (A, A^*) the function

$$V = \sum_{A\in\mathsf{A}} \beta_A V_A \tag{9}$$

is a Lyapunov function for the Morse decomposition M with order $(\mathsf{P}, \leq)$, see Definition 5.1 and also [16, Equation 7]). However, using all attractors in A to define such a Lyapunov function is more than necessary, and to develop a computationally efficient algorithm it is desirable to find a smaller collection of attractors that will generate a Lyapunov function. The optimal such set is ultimately determined by the specific structure of the poset $(\mathsf{P}, \leq)$, but there is a natural subset that works which can be defined for all partial orders and which typically yields a significant reduction in the number of attractors that need to be considered.

The elements in $\mathsf{O}(\mathsf{P})$ can be written as unions of down sets of the form $\downarrow p = \{q \in \mathsf{P} \mid q \leq p\}$, which are called the *join-irreducible* elements of the lattice $\mathsf{O}(\mathsf{P})$. The set of join irreducibles is denoted by $\mathsf{J}^\vee(\mathsf{O}(\mathsf{P}))$. Therefore each attractor in A can be written as a union of attractors of the form $A_p = \nu(\downarrow p)$, i.e. $\mathsf{J}^\vee(\mathsf{A}) = \{A_p \mid p \in \mathsf{P}\}$.

Theorem 5.2. *Let S be a compact, invariant set, let M be a Morse decomposition for S, labeled by a poset $(\mathsf{P}, \leq)$, and let A be the corresponding sublattice of attractors whose join irreducible elements are A_p. For each $p \in \mathsf{P}$ let $\beta_p > 0$ with $\sum_{p\in\mathsf{P}} \beta_p = 1$ and V_p be a Lyapunov function for the attractor-repeller pair (A_p, A_p^*). Then*

$$V = \sum_{p\in\mathsf{P}} \beta_p V_p \tag{10}$$

is a Lyapunov function for $(S, \mathsf{M}, \mathsf{P}, \leq)$.

Proof. From the definition of $A_r = \nu(\downarrow r)$ we have that $M_p \subset A_r$ so that $V_r(M_p) = 0$ if $p \leq r$ and $M_p \subset A_r^*$ so that $V_r(M_p) = 1$ if $p \not\leq r$. Hence $V(M_p) = \sum_{p\not\leq r} \beta_r$. If $q < p$, then $c_q = V(M_q) = \sum_{q\not\leq r} \beta_r < \sum_{p\not\leq r} \beta_r = V(M_p) = c_p$. This establishes conditions (i) and (ii) in Definition 5.1.

Now suppose $x \in S \setminus \cup_{p\in\mathsf{P}} M(p)$. Since M is a Morse decomposition, there exist p, q with $q \leq p$ such that $\omega(x) \subset M(q)$ and $\alpha_{\mathrm{o}}(\gamma_x^-) \subset M(p)$. Then $M(q) \subset A_q, M(p) \subset A_q^*$, and $x \in S \setminus (A_q \cup A_q^*)$, which implies that $V_q(f(x)) < V_q(x)$. Since each function V_p is decreasing, $V(f(x)) < V(x)$ This establishes condition (iii) in Definition 5.1 and completes the proof. ∎

Note that in Theorem 5.2 the partial order $(\mathsf{P}, \leq)$ on the Morse sets is given. Any coarser order obtained by adding more relations is also admissible, but such a coarser order relation results in a loss of dynamical information about the system. In particular, the set of join irreducible elements may grow, thus requiring additional attractors to be included in the definition of V, i.e. in (10).

5.2. **Approximating Lyapunov functions for attractor-repeller pairs.** Let $A \in \mathsf{Att}(X, f)$. We begin by describing the the standard analytical construction of a Lyapunov function for an attractor-repeller pair (A, A^*). The *distance potential* $\mathsf{dp}_A \colon X \to [0, 1]$ defined by

$$\mathsf{dp}_A(x) = \frac{\operatorname{dist}(x, A)}{\operatorname{dist}(x, A) + \operatorname{dist}(x, A^*)} \tag{11}$$

satisfies $\mathsf{dp}_A^{-1}(0) = A$ and $\mathsf{dp}_A^{-1}(1) = A^*$. Moreover, dp_A is Lipschitz since A and A^* are disjoint and compact. We incorporate the dynamics of f through the family of functions

$$v_A^*(x, k) = \max\{\mathsf{dp}_A(y) \mid y \in \gamma_x([k, \infty))\}$$

which maximize the distance potential over forward orbits. The function

$$V_A(x) = \sum_{k=0}^{\infty} 2^{-k-1} v_A^*(x, k) \tag{12}$$

is a Lyapunov function for (A, A^*), as is shown in [16].

Following [16, 3] this function can be approximated by a function on grid elements as follows. Let $\mathcal{A} \in \mathsf{Att}(\mathcal{X}, \mathcal{F})$. With respect to the attractor repeller pair

$(\mathcal{A}, \mathcal{A}^*)$ for $\mathcal{F}$, a *distance potential* $\mathsf{dp}_{\mathcal{A}} \colon \mathcal{X} \to [0,1]$ is defined by

$$\mathsf{dp}_{\mathcal{A}}(\xi) = \frac{\operatorname{dist}(x_\xi, |\mathcal{A}|)}{\operatorname{dist}(x_\xi, |\mathcal{A}|) + \operatorname{dist}(x_\xi, |\mathcal{A}^*|)} \tag{13}$$

where x_ξ is the centroid of ξ. Then the function $\nu^*_{\mathcal{A}} \colon \mathcal{X} \to [0,1]$ defined by

$$\nu^*_{\mathcal{A}}(\xi) = \max_{\eta \in \Gamma_0^+(\xi)} \mathsf{dp}_{\mathcal{A}}(\eta) \tag{14}$$

maximizes the value of $\mathsf{dp}_{\mathcal{A}}$ over the complete forward image of ξ. Finally we consider the function $\mathcal{V}_{\mathcal{A}} \colon \mathcal{X} \to [0,1]$ defined recursively by

$$\mathcal{V}_{\mathcal{A}}(\xi) = \begin{cases} 0 & \text{if } \xi \in \mathcal{A} \\ 1 & \text{if } \xi \in \mathcal{A}^* \\ \frac{1}{2}\nu^*_{\mathcal{A}}(\xi) + \frac{1}{2}\max_{\eta \in \mathcal{F}(\xi)} \mathcal{V}_{\mathcal{A}}(\eta) & \text{otherwise.} \end{cases} \tag{15}$$

Lemma 3.2 and Theorem 3.3 in [3] imply that if $(\mathcal{A}, \mathcal{A}^*)$ is an attractor-repeller pair for $\mathcal{F}$ that well approximates the attractor-repeller pair (A, A^*), then $\mathcal{V}_{\mathcal{A}}$ is a combinatorial Lyapunov function for $(\mathcal{A}, \mathcal{A}^*)$ which well approximates V_A. We make this statement more specific in Section 5.4.

In practice, computing the distance potential $\mathsf{dp}_{\mathcal{A}}$ using distances between sets of grid elements is not computationally efficient. However, we now describe an $O(N \log N)$ time algorithm, where N is the number of grid elements in $\mathcal{X}$, to compute a different combinatorial distance potential $\mathsf{cdp}_{\mathcal{A}} \colon \mathcal{X} \to \mathbb{R}$ given a combinatorial attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$. We show it also approximates the true distance potential (11) arbitrarily closely as we refine the grid. Our algorithm works in the case where the distance metric d is chosen to be Manhattan distance $d(x,y) = \sum_{k=1}^{d} |x_k - y_k|$ in $\mathbb{R}^d$, and the grid implements the "TreeGrid" interface discussed above so that there is an underlying binary tree. We remark that this algorithm is a vast improvement over a naive approach where one calculates via brute force the minimum distance from each grid element of $\mathcal{X}$ to each grid element in the attractor and in the repeller, which would require $O(N^2)$ time.

5.2.1. *Dijkstra's Algorithm.* A key subroutine of our distance potential algorithm will be Dijkstra's algorithm [8] for computing shortest paths on a graph $\mathcal{W}$ with weighted edges. Recall that Dijkstra's algorithm proceeds by seeding an initial vertex set $\mathcal{S}$ with a priority value 0, and placing these vertices in a priority queue, with the highest priority being given to those vertices with the *smallest* priority value. A vertex u with highest priority is processed by marking it as processed, and placing each unprocessed neighbor v of u into the priority queue with a priority value equal to the priority value of u plus the weight of the edge from u to v. When a vertex is inserted into the priority queue multiple times, the effect is the vertex will have the smallest priority with which it has been inserted. This continues until all vertices have been processed. The *Dijkstra distance* of a vertex v to the set S, which we will denote $d_{\mathcal{W}}(v, S)$, is the priority value attached to the vertex v

when it is processed. If the algorithm is implemented with a so-called *Fibonacci heap*, then this algorithm, **Dijkstra**$(S, \mathcal{W})$, requires $O(E + V \log V)$ time and $O(V)$ space [9], and returns the Dijkstra distances $d_{\mathcal{W}}(v, S)$ for all $v \in \mathcal{W}$. Here the $\log V$ cost is associated with the sorting necessary in the priority queue.

Our algorithm for computing a combinatorial distance potential requires a weighted graph $\mathcal{W}$ to which Dijkstra's algorithm is applied. To this end, we utilize the binary tree structure underlying the grid $\mathcal{X}$. In particular, we realize the vertices of $\mathcal{W}$ as the nodes in the binary tree underlying $\mathcal{X}$. We will abuse notation and regard $\mathcal{X}$ as a subset of the vertex set. What is left is to specify the edges of $\mathcal{W}$ and their weights. To this end, first recall that to each such node $v \in \mathcal{W}$ there is an associated rectangular box in $\mathbb{R}^d$, and we will use the notation $|v|$ to denote this geometric realization.

We let (u, v) be an edge in $\mathcal{W}$ if one of two conditions holds:

- Condition (A): u and v have a parent-child or child-parent relationship.
- Condition (B): The rectangles $|u|$ and $|v|$ are congruent and share a codimension-1 face.

In either case, we assign a weight to the edge (u, v) which is the Manhattan distance between the midpoints of $|u|$ and $|v|$.

The motivation for this weighted graph comes from the following observation. Call a path in $\mathbb{R}^d$ that is comprised of a finite sequence of axis-aligned treks a *Manhattan curve*. Paths in the graph $\mathcal{W}$ correspond to Manhattan curves, and the weights on the edges correspond to the Manhattan distances of the treks of which they are comprised. Hence the Dijkstra distance on this graph gives an upper bound for the Manhattan distance between the grid elements. Shortly, we will show that the amount of overestimate can be bounded.

We omit the algorithm for producing the weighted adjacency list of a vertex v in the weighted graph $\mathcal{W}$ and content ourselves with only a few comments. It is straightforward to give the adjacencies due to Condition (A). Condition (B) can be re-expressed it in terms of the path-from-root information, i.e. the sequence of left/right directions which one must follow from the root of the tree to find the vertex. Expressing it this way provides a method to construct an efficient algorithm. Given path-from-root information, we split it into d lists such that the k-th list has the path information at depths $\{i \mid i \bmod d = k\}$. The idea is that the k-th list has the path information corresponding to choices made for splitting dimension k. If for each such list we construct a sequence of digits by a digit 0 for a left move and a 1 for a right move, we obtain d binary sequences. These binary sequences we may then reinterpret as nonnegative integers written in binary notation. Call these the *coordinates* of the tree node, and indeed for a uniformly subdivided grid they would be precisely the natural integer coordinates one would put at a fixed depth. Supplemented with depth, the coordinates provide an alternative characterization of the tree node. Using this terminology, Condition (B) states that u and v are the

same depth and that the coordinates of u and v are identical in all but one dimension, and in the dimension the coordinates differ, they differ by precisely one. With this insight it becomes straightforward, albeit somewhat tedious, to produce an efficient algorithm that relies on tree-moves for computing the weighted adjacency list of a vertex v. We find the coordinates for v, find all of the at most $2d$ coordinates which are neighboring in the sense of Condition (B), and locate them via tree moves **left** and **right** starting at the root by following the directions of the binary digits of the new coordinates in round-robin fashion.

Proposition 5.3. *The Dijkstra distance on $\mathcal{W}$ approximates the Manhattan distance in the following sense. For each pair of grid elements $\xi_1, \xi_2 \in \mathcal{X}$, and for each $x_1 \in \xi_1$, $x_2 \in \xi_2$, we have*

$$|d_{\mathcal{W}}(\xi_1, \xi_2) - d(x_1, x_2)| < 5\,\delta \tag{16}$$

where δ is the maximal diameter of the grid elements, i.e. $\delta := \max\{\mathrm{diam}\,(\xi) \mid \xi \in \mathcal{X}\}$.

Proof. Let x_{ξ_1}, x_{ξ_2} be the centroids of ξ_1, ξ_2, respectively. Observe $d(x_i, x_{\xi_i}) \leq \delta/2$ for $i = 1, 2$. Hence $|d(x_1, x_2) - d(x_{\xi_1}, x_{\xi_2})| \leq \delta$. What remains is to show $|d(x_{\xi_1}, x_{\xi_2}) - d_{\mathcal{W}}(\xi_1, \xi_2)| < 2\delta$. We must have $d_{\mathcal{W}}(\xi_1, \xi_2) \geq d(x_{\xi_1}, x_{\xi_2})$, since the Dijkstra distance in $\mathcal{W}$ between ξ_1 and ξ_2 measures the Manhattan distance of some Manhattan curve between x_{ξ_1} and x_{ξ_2}. We need only bound how much $d_{\mathcal{W}}(\xi_1, \xi_2)$ may be overestimating $d(x_{\xi_1}, x_{\xi_2})$. In order to do this, we give a path in $\mathcal{W}$ which we can show does not overestimate by too much. We construct this path as follows. Let D be the largest number so that every leaf in the binary tree for $\mathcal{X}$ is at least depth D. Let η_1 and η_2 be the unique ancestors at depth D of ξ_1 and ξ_2 in the binary tree for $\mathcal{X}$. Note that we might have $\xi_1 = \eta_1$ or $\xi_2 = \eta_2$ in this construction. Consider the path in $\mathcal{W}$ that starts at ξ_1, climbs through parents until depth D is reached at $\eta_1 \in \mathcal{W}$, walks to adjacent neighbors at depth D, which are defined by Condition (B), in the most straightforward way to reach $\eta_2 \in \mathcal{W}$, and then descends through the appropriate path of children in order to reach ξ_2. Let x_{η_1}, x_{η_2} be the centroids of η_1, η_2, respectively. Observe that $d_{\mathcal{W}}(\eta_1, \eta_2) = d(x_{\eta_1}, x_{\eta_2})$ since the adjacencies given by Condition (B) of $\mathcal{W}$ at depth D provide a uniform grid. From this we may realize the bound $0 \leq d_{\mathcal{W}}(\xi_1, \xi_2) - d(x_{\xi_1}, x_{\xi_2}) \leq d_{\mathcal{W}}(\xi_1, \eta_1) + d(x_{\xi_1}, x_{\eta_1}) + d_{\mathcal{W}}(\xi_2, \eta_2) + d(x_{\xi_2}, x_{\eta_2}) \leq d_{\mathcal{W}}(\xi_1, \eta_1) + d_{\mathcal{W}}(\xi_2, \eta_2) + 2\delta$. The latter inequality holds since the points x_{ξ_i}, x_{η_i} for $i = 1, 2$ are both contained in a rectangle of diameter less than δ. For each $k = 0, 1, 2, \cdots$, the sum of the weights of parent-child edges leading from depth $D + kd$ to depth $D + (k+1)d$ is at most $\delta/2^{1+k}$ since it consists of d treks whose lengths most be less than the diameter of a depth $D + kd$ tree node. Hence a geometric series argument gives us $d_{\mathcal{W}}(\xi_1, \eta_1) < \sum_{k=1}^{\infty} \delta/2^{1+k} = \delta$ and similarly $d_{\mathcal{W}}(\xi_2, \eta_2) < \delta$. This gives us $d_{\mathcal{W}}(\xi_1, \xi_2) - d(x_{\xi_1}, x_{\xi_2}) < 4\delta$, which can be combined with $|d(x_1, x_2) - d(x_{\xi_1}, x_{\xi_2})| \leq \delta$ to give the desired result. □

Corollary 5.4. *Given $\mathcal{S} \subset \mathcal{X}$ and $\xi \notin \mathcal{S}$ we have*

$$|d_{\mathcal{W}}(\xi, \mathcal{S}) - d(x_\xi, |\mathcal{S}|)| < 5 \operatorname{diam}(\mathcal{X})$$

where $d_{\mathcal{W}}(\xi, \mathcal{S}) = \min_{\eta \in \mathcal{S}} d_{\mathcal{W}}(\xi, \eta)$ and $d(x_\xi, |\mathcal{S}|) = \min_{y \in |\mathcal{S}|} d(x_\xi, y)$.

Proof. Proposition 5.3 gives $d_{\mathcal{W}}(\xi, \eta) \leq 5\operatorname{diam}(\mathcal{X}) + d(x_\xi, y)$ for all $\eta \in \mathcal{S}$ and all $y \in |\mathcal{S}|$ which implies $\min_{\eta \in \mathcal{S}} d_{\mathcal{W}}(\xi, \eta) \leq 5\operatorname{diam}(\mathcal{X}) + \min_{y \in |\mathcal{S}|} d(x_\xi, y)$. Hence $d_{\mathcal{W}}(\xi, \mathcal{S}) - d(x_\xi, |\mathcal{S}|) \leq 5\operatorname{diam}(\mathcal{X})$. The inequality $d(x_\xi, |\mathcal{S}|) - d_{\mathcal{W}}(\xi, \mathcal{S}) \leq 5\operatorname{diam}(\mathcal{X})$ follows by the same reasoning. ∎

5.2.2. *Combinatorial Distance Potential.* Given $\mathcal{A} \in \mathsf{Att}(\mathcal{X}, \mathcal{F})$ and the corresponding attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$, we can now define our formula for the combinatorial distance potential $\mathsf{cdp}_{\mathcal{A}} \colon \mathcal{X} \to \mathbb{R}$, for all $\xi \in \mathcal{X}$,

$$\mathsf{cdp}_{\mathcal{A}}(\xi) := \frac{d_{\mathcal{W}}(\xi, \mathcal{A})}{d_{\mathcal{W}}(\xi, \mathcal{A}) + d_{\mathcal{W}}(\xi, \mathcal{A}^*)}. \tag{17}$$

We give the following algorithm to compute $\mathsf{cdp}_{\mathcal{A}}$.

Algorithm 2 Computing Combinatorial Distance Potential

DistancePotential
Given: A combinatorial attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$.
Output: A distance potential $\mathsf{cdp}_{\mathcal{A}}$ and also the Dijkstra distance $d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)$
Realize the weighted graph $\mathcal{W}$ corresponding to the grid $\mathcal{X}$.
$d_{\mathcal{W}}(\cdot, \mathcal{A}) \leftarrow$ **Dijkstra**$(\mathcal{A}, \mathcal{W})$.
$d_{\mathcal{W}}(\cdot, \mathcal{A}^*) \leftarrow$ **Dijkstra**$(\mathcal{A}^*, \mathcal{W})$.
for $\xi \in \mathcal{X}$ **do**
  $\mathsf{cdp}_{\mathcal{A}}(\xi) \leftarrow \frac{d_{\mathcal{W}}(\xi, \mathcal{A})}{d_{\mathcal{W}}(\xi, \mathcal{A}) + d_{\mathcal{W}}(\xi, \mathcal{A}^*)}$
end for
$\alpha \leftarrow \infty$
for $\xi \in \mathcal{A}$ **do**
  $\alpha \leftarrow \min\{\alpha, d_{\mathcal{W}}(\xi, \mathcal{A}^*)\}$
end for
$d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*) \leftarrow \alpha$

Proposition 5.5. *Algorithm **DistancePotential** computes the combinatorial distance potential defined by Equation 17. The time and space complexity are $O(dN + N \log N)$ and $O(N)$, respectively, where $N = \operatorname{card}(\mathcal{X})$.*

Proof. Correctness of the combinatorial distance potential computation follows from the preceding discussion and the correctness of the computation of $d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)$ follows from the observation that $d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*) = \min_{\xi \in \mathcal{A}} d_{\mathcal{W}}(\xi, \mathcal{A}^*)$. Since $\mathcal{W}$ will have approximately $(d+1)N$ edges, the earlier discussion of Dijkstra's algorithm tells us that the time complexity will be $O(dN + N \log N)$. The space complexity of $O(N)$ reflects the cost of storing the result, and storing the priority queues. □

5.2.3. *Combinatorial Distance Potential Star.* Finally we compute a combinatorial distance potential star $\mathsf{c}\nu^*_{\mathcal{A}} \colon \mathcal{X} \to \mathbb{R}$ in the same manner as $\nu^*_{\mathcal{A}}$ is defined in (14), namely

$$\mathsf{c}\nu^*_{\mathcal{A}}(\xi) := \max\{\mathsf{cdp}_{\mathcal{A}}(\eta) \mid \eta \text{ is reachable from } \xi\}. \tag{18}$$

To this end we provide the following algorithm.

Algorithm 3 Computing Distance Potential Star

Global: The topologically sorted list $\mathcal{S}$ of the strongly connected components

DistancePotentialStar
Given: An attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$
Output: The combinatorial distance potential star as defined in Equation (18) and also the Dijkstra distance $d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)$.

$(\mathsf{cdp}_{\mathcal{A}}, d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)) \leftarrow$ **DistancePotential**$(\mathcal{A}, \mathcal{A}^*)$
$\mathsf{c}\nu^*_{\mathcal{A}} \leftarrow \mathsf{cdp}_{\mathcal{A}}$
for $S \in \mathcal{S}$ (in *reverse* order) **do**
  if $S \notin \mathcal{A} \cup \mathcal{A}^*$ **then**
    Let $\xi \in \mathcal{X}$ such that $S = \{\xi\}$.
    for $\eta \in \mathcal{F}(\xi)$ **do**
      $\mathsf{c}\nu^*_{\mathcal{A}}(\xi) \leftarrow \max\{\mathsf{c}\nu^*_{\mathcal{A}}(\xi), \mathsf{c}\nu^*_{\mathcal{A}}(\eta)\}$
    end for
  end if
end for

Proposition 5.6. *The algorithm* ***DistancePotentialStar*** *correctly computes the combinatorial distance potential star defined in Equation (18), with time complexity $O(dN + N \log N + E)$ and space complexity $O(N)$. Here N is the number of grid elements and E is the number of edges in the directed graph corresponding to the combinatorial map $\mathcal{F}$ of Equation (5).*

Proof. The correctness of the algorithm follows from the observation that $\mathsf{c}\nu^*_{\mathcal{P}}$ satisfies the following recursive relation:

$$\mathsf{c}\nu^*_{\mathcal{A}}(\xi) = \max\{\mathsf{cdp}_{\mathcal{A}}(\xi), \max\{\mathsf{c}\nu^*_{\mathcal{A}}(\eta) \mid \eta \in \mathcal{F}(\xi)\}\}.$$

Iterating through $\mathcal{S} \setminus (\mathcal{A} \cup \mathcal{A}^*)$, which consists only of singleton components, in a reverse topological sort ensures that $\mathsf{c}\nu^*_{\mathcal{A}}(\eta)$ is always computed before $\mathsf{c}\nu^*_{\mathcal{A}}(\xi)$ whenever $\eta \in \mathcal{F}(\xi)$, so this recursive definition is properly implemented. Proposition 5.5 gives us a baseline resource usage of $O(dN + N \log N)$ time and $O(N)$ space. The time cost apart from this is proportional to the number of adjacencies examined, and no adjacency $\xi \mapsto \eta$ is ever examined more than once; this results in the extra $O(E)$ term in the time complexity. The extra space complexity is due to storage of the result and can be subsumed into the big-oh notation (in fact, the distance potential can be overwritten). 5.5. □

5.2.4. *Computing Lyapunov functions for attractor-repeller pairs.* Finally we replace $\mathsf{c}\nu^*_{\mathcal{A}}$ in formula (15) for $\mathcal{V}_{\mathcal{A}}$ to obtain the function

$$\mathsf{c}\mathcal{V}_{\mathcal{A}}(\xi) = \begin{cases} 0 & \text{if } \xi \in \mathcal{A} \\ 1 & \text{if } \xi \in \mathcal{A}^* \\ \frac{1}{2}\mathsf{c}\nu^*_{\mathcal{A}}(\xi) + \frac{1}{2}\max_{\eta\in\mathcal{F}(\xi)} \mathsf{c}\mathcal{V}_{\mathcal{A}}(\eta) & \text{otherwise,} \end{cases} \tag{19}$$

which approximates $\mathcal{V}_{\mathcal{A}}$. Lemma 3.2 in [3] implies that $\mathsf{c}\mathcal{V}_{\mathcal{A}}$ is a combinatorial Lyapunov function for $(\mathcal{A}, \mathcal{A}^*)$. The following algorithm computes $\mathsf{c}\mathcal{V}_{\mathcal{A}}$.

Algorithm 4 Computing Combinatorial Lyapunov Function Summand

Global: A topologically sorted list $\mathcal{S}$ of the strongly connected components.
LyapunovSummand
Given: An attractor-repeller pair $(\mathcal{A}, \mathcal{A}^*)$
Output: The combinatorial Lyapunov summand as defined in Equation (19) and also the Dijkstra distance $d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)$.
$(\mathsf{c}\nu^*_{\mathcal{A}}, d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)) \leftarrow$ **DistancePotentialStar**$(\mathcal{A}, \mathcal{A}^*)$
for $\xi \in \mathcal{X}$ **do**
  $\mathsf{c}\mathcal{V}_{\mathcal{A}}(\xi) \leftarrow 0$
end for
for $\xi \in \mathcal{A}^*$ **do**
  $\mathsf{c}\mathcal{V}_{\mathcal{A}}(\xi) \leftarrow 1$
end for
for $S \in \mathcal{S}$ (in *reverse* order) **do**
  if $S \notin \mathcal{A} \cup \mathcal{A}^*$ **then**
    Let $\xi \in \mathcal{X}$ such that $S = \{\xi\}$.
    $\alpha \leftarrow 0$
    for $\eta \in \mathcal{F}(\xi)$ **do**
      $\alpha \leftarrow \max\{\alpha, \mathcal{V}_{\mathcal{A}}(\eta)\}$
    end for
    $\mathsf{c}\mathcal{V}_{\mathcal{A}}(\xi) \leftarrow \frac{1}{2}\left(\mathsf{c}\nu^*_{\mathcal{A}}(\xi) + \alpha\right)$
  end if
end for

Proposition 5.7. *Algorithm* ***LyapunovSummand*** *correctly computes the combinatorial Lyapunov function* $\mathsf{c}\mathcal{V}_{\mathcal{A}}$ *for the attractor-repeller pair* $(\mathcal{A}, \mathcal{A}^*)$ *defined in Equation (19) in* $O(dN + N\log N + E)$ *time and* $O(N)$ *space, where* N *is the number of grid elements and*

$$E := \sum_{\xi\in\mathcal{X}} \operatorname{card}(\mathcal{F}(\xi))$$

is the number of edges in the directed graph corresponding to the combinatorial map $\mathcal{F}$ *of Equation (5).*

Proof. The proof is essentially the same reasoning as the proof of Proposition 5.6; we see that iterating through singleton strongly connected components in reverse topological sort correctly implements the recursive definition of Equation (19). □

5.3. **Computing Lyapunov functions for Morse decompositions.** Our goal is to compute combinatorial Lyapunov functions for $\mathcal{F}$, i.e. piecewise-constant, combinatorial Lyapunov functions that approximate continuous Lyapunov functions for

f. Given an outer approximation $\mathcal{F}$, the recurrent set $\mathcal{R}$ has finitely many recurrent components $\mathcal{M}$. Corollary 4.2 in [16] implies that f is gradient-like on the complement of $|\mathcal{R}|$, and the realizations of the recurrent components are isolating neighborhoods for a Morse decomposition for f. This means that the maximum amount of recurrence/gradient-like information we can obtain from $\mathcal{F}$ is a Morse decomposition M with partial order $(\mathsf{P}, \leq)$.

Theorem 5.2 provides a formula for a Lyapunov function for a Morse decomposition as a weighted average of Lyapunov functions for attractor-repeller pairs generated from the join-irreducible attractors. This leads us to define the following piecewise-constant, combinatorial Lyapunov function for $\mathcal{F}$

$$\mathsf{c}\mathcal{V} = \sum_{p \in \mathsf{P}} \beta_p \, \mathsf{c}\mathcal{V}_p \tag{20}$$

where the weights β_p are defined according to the distance from the attractor $\mathcal{A}_p$ to its dual repeller $\mathcal{A}_p^*$. Specifically,

$$w = \sum_{p \in \mathsf{P}} d_{\mathcal{W}}(\mathcal{A}_p, \mathcal{A}_p^*) \quad \text{and} \quad \beta_p = d_{\mathcal{W}}(\mathcal{A}_p, \mathcal{A}_p^*)/w. \tag{21}$$

In the next section we discuss the reasoning behind this specific choice of weights and describe how well these functions approximate continuous Lyapunov functions for f. The following algorithm computes the final combinatorial Lyapunov function $\mathsf{c}\mathcal{V}$.

Algorithm 5 Computing Combinatorial Lyapunov Function

Global variables:
A list $\mathcal{M}$ of the combinatorial recurrent components.

ComputeLyapunov
$w \leftarrow 0$.
for $\xi \in \mathcal{X}$ **do**
  $\mathsf{c}\mathcal{V}(\xi) \leftarrow 0$.
end for
for $M \in \mathcal{M}$ **do**
  $\mathcal{A} \leftarrow$ **ComputeAttractor**(M).
  $\mathcal{A}^* \leftarrow$ **ComputeDualRepeller**$(\mathcal{A})$.
  $(\mathsf{c}\mathcal{V}_{\mathcal{A}}, d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)) \leftarrow$ **LyapunovSummand**$(\mathcal{A}, \mathcal{A}^*)$.
  $w \leftarrow w + d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*)$
  for $\xi \in \mathcal{X}$ **do**
    $\mathsf{c}\mathcal{V}(\xi) \leftarrow \mathsf{c}\mathcal{V}(\xi) + d_{\mathcal{W}}(\mathcal{A}, \mathcal{A}^*) \cdot \mathsf{c}\mathcal{V}_{\mathcal{A}}(\xi)$
  end for
end for
for $\xi \in \mathcal{X}$ **do**
  $\mathsf{c}\mathcal{V}(\xi) \leftarrow \mathsf{c}\mathcal{V}(\xi)/w$
end for

Theorem 5.8. *Given the recurrent components $\mathcal{M} = \{M(p) \mid p \in \mathsf{P}\}$, the algorithm **ComputeLyapunov** correctly computes the combinatorial Lyapunov function defined in Equation (20), with time complexity*

$$O(\mathsf{card}(\mathsf{P}) \cdot (dN + N \log N + E))$$

and space complexity $O(N)$, where N is the number of grid elements and

$$E := \sum_{\xi \in \mathcal{X}} \mathsf{card}(\mathcal{F}(\xi))$$

is the number of edges in the directed graph corresponding to the combinatorial map $\mathcal{F}$ of Equation (5).

Proof. Both correctness and the complexity estimates follow immediately from Equation (20) and Propositions 4.2 and 5.7. □

5.4. **Convergence of approximate Lyapunov functions.** We want to show that the combinatorial Lyapunov function given by the formula (20) with weights β_p specified in (21) is a good approximation to an appropriate continuous Lyapunov function for f, if the grid size is sufficiently small. In particular, we would like to show that for a sequence of outer approximations $\mathcal{F}_n \colon \mathcal{X}_n \rightrightarrows \mathcal{X}_n$ with $\operatorname{diam}(\mathcal{X}_n) \to 0$ as

$n \to \infty$, the corresponding functions cV_n converge uniformly to a continuous Lyapunov function for f.

First, it should be clear that without any further assumptions about how well $\mathcal{F}_n$ approximates the images of f, nothing can be said. For the purposes of clarity, we consider the *minimal outer approximation* of f on $\mathcal{X}_n$ given by

$$(\mathcal{F}_o)_n(\xi) = \{\eta \in \mathcal{X}_n \mid \eta \cap f(\xi) \neq \varnothing\}.$$

Clearly, this outer approximation is not attainable in practical computations, but the notion of a convergent sequence of outer approximations is developed in [18]. We do not include the details of such convergent sequences here, however the results that we obtain for sequences of minimal outer approximations will also hold for convergent sequences of outer approximations using Proposition 5.4 and Corollary 5.6 in [18].

We begin by considering Lyapunov functions for attractor-repeller pairs. Proposition 5.5 in [16] establishes a one-to-one correspondence between well-separated attractor-repeller pairs for f and attractor-repeller pairs for the minimal outer approximation $\mathcal{F}_o$, and Corollary 5.6 in [18] establishes the same result for convergent sequences of outer approximations. Specifically, for $c > 0$ let

$$\mathsf{Q}_c := \{A \in \mathsf{Att}(X, f) \mid \operatorname{dist}(A, A^*) \geq c\}.$$

Let $\epsilon < \min\{\operatorname{dist}(A, A^*)/2 \mid A \in \mathsf{Q}_c\}$. If $\operatorname{diam}(\mathcal{X})$ is sufficiently small, then for each $A \in \mathsf{Q}_c$ there exists a unique $\mathcal{A} \in \mathsf{Att}(\mathcal{X}, \mathcal{F})$ with the property that

$$A \subset |\mathcal{A}| \subset B_\epsilon(A) \quad \text{and} \quad A^* \subset |\mathcal{A}^*| \subset B_\epsilon(A^*). \tag{22}$$

We denote this correspondence by $\Pi \colon \mathsf{Q}_c \to \mathsf{Att}(\mathcal{X}, \mathcal{F})$. This leads to the following theorem.

Theorem 5.9. *If $\mathcal{A}_n$ is a sequence of attractors arising from a convergent sequence of outer approximations $\mathcal{F}_n \colon \mathcal{X}_n \to \mathcal{X}_n$ with $\operatorname{diam}(\mathcal{X}_n) \to 0$ as $n \to \infty$, and there exists an attractor A for f such that $(|\mathcal{A}_n|, |\mathcal{A}_n^*|) \to (A, A^*)$ in the Hausdorff metric, then $\mathsf{cV}_{\mathcal{A}_n}$ converges uniformly to V_A defined in equation (12). Moreover, given A such a sequence $\mathcal{A}_n$ always exists.*

Proof. Corollary 5.4 states that the combinatorial distance $d_{\mathcal{W}}(\xi, \mathcal{S})$ and the geometric, Manhattan distance $d(x_\xi, |\mathcal{S}|)$ satisfy

$$|d_{\mathcal{W}}(\xi, \mathcal{S}) - d(x_\xi, |\mathcal{S}|)| < 5\operatorname{diam}(\mathcal{X}_n).$$

Lemma 3.7 of [3] establishes that the distance potentials $\mathsf{dp}_{\mathcal{A}_n}$ and $\mathsf{cdp}_{\mathcal{A}_n}$ defined in equations (13) and (17) respectively, as well as the functions $\mathcal{V}_{\mathcal{A}_n}$ and $\mathsf{cV}_{\mathcal{A}_n}$ defined in equations (15) and (19) respectively, satisfy similar estimates so that there exists $C > 0$ such that

$$|\mathsf{dp}_{\mathcal{A}_n}(\xi) - \mathsf{cdp}_{\mathcal{A}_n}(\xi)| \leq C\operatorname{diam}(\mathcal{X}_n) \quad \text{and} \quad |\mathcal{V}_{\mathcal{A}_n}(\xi) - \mathsf{cV}_{\mathcal{A}_n}(\xi)| \leq C\operatorname{diam}(\mathcal{X}_n).$$

Now, [3, Theorem 3.3] establishes that $\mathsf{c}\mathcal{V}_{\mathcal{A}_n}$ converges uniformly to V_A. Furthermore, [16, Theorem 5.5] and Corollary 5.6 in [18, Corollary 5.6] establishes that given (A, A^*) the sequence $(\mathcal{A}_n, \mathcal{A}_n^*) = (\Pi_n(A), \Pi_n(A)^*)$ converges to (A, A^*) in the Hausdorff metric using (22). □

When Π maps join-irreducible attractors onto combinatorial join-irreducible attractors, then we can give a result following from Theorems 5.2 and 5.9 that our algorithms compute a combinatorial Lyapunov function approximating Equation (10). In fact, the weighting of the terms allows us to be able to do somewhat better, and relax this to assuming that Π maps to join-irreducible combinatorial attractors, but is not necessarily onto.

In general, however, Π need not map only to join-irreducible combinatorial attractors. This can happen when there are "spurious" combinatorial Morse sets (i.e. recurrent components for which there is an empty maximal invariant set). In this case we do not give a convergence result. However, we will show that it is possible to give an a-posteriori analysis to establish that this was not the case for the computational results presented in this paper. We sketch some ideas for how to handle the general situation at the end of this section.

Theorem 5.10. *Suppose* $\mathsf{Att}(X, f)$ *is finite and let* M *be the Morse decomposition for* f *ordered by* $(\mathsf{P}, \leq)$ *with corresponding lattice of attractors* $\mathsf{A} = \mathsf{Att}(X, f)$. *Let* $c = \min\{\operatorname{dist}(A, A^*) \mid A \in \mathsf{A}\}$. *If* $\operatorname{diam}(\mathcal{X})$ *is sufficiently small, and* $\mathcal{F}$ *is the minimal outer approximation, then all attractors in* A *can be resolved by the correspondence* Π. *Suppose that for every join-irreducible attractor* $\mathcal{A}$ *for* $\mathcal{F}$ *the attractor* $\omega(|\mathcal{A}|)$ *is join-irreducible in* A. *Further suppose that for every join-irreducible* $A_p \in \mathsf{Att}(X, f)$ *the corresponding attractor* $\Pi(A_p) \in \mathsf{Att}(\mathcal{X}, \mathcal{F})$ *is join-irreducible. Then there is a Lyapunov function* V *for* $(X, \mathsf{M}, \mathsf{P}, \leq)$ *such that* $\|\mathsf{c}\mathcal{V} - V\|_\infty \to 0$ *as* $\operatorname{diam}(\mathcal{X}) \to 0$.

Proof. [16, Proposition 5.5] and [18, Corollary 5.6] imply that for $\operatorname{diam}(\mathcal{X})$ small enough, Π is surjective. By [18, Proposition 4.7] the map from $\mathsf{Att}(\mathcal{X}, \mathcal{F})$ to $\mathsf{Att}(X, f)$ given by $\mathcal{A} \mapsto \omega(|\mathcal{A}|)$ is well-defined. Moreover, $\mathsf{Att}(\mathcal{X}, \mathcal{F})$ has a natural lattice structure and the map $\omega(|\cdot|)$ is a lattice homomorphism by [18, Proposition 4.13]. Since Π is surjective, $\omega(|\cdot|)$ is surjective onto $\mathsf{A} = \mathsf{Att}(X, f)$.

By Theorem 5.9 for each $p \in \mathsf{P}$ the combinatorial Lyapunov function $\mathsf{c}\mathcal{V}_{\Pi(A_p)}$ defined in Equation (19) converges uniformly to the attractor-repeller pair Lyapunov function V_{A_p} defined in Equation (12). Suppose the recurrent components of $\mathcal{F}$ are labeled by the poset R. Recall that the combinatorial Lyapunov function for the recurrent components of $\mathcal{F}$ is given by Equation (20)

$$\mathsf{c}\mathcal{V} = \sum_{r \in \mathsf{R}} \beta_r \, \mathsf{c}\mathcal{V}_{\mathcal{A}_r}$$

where the weights are defined in Equation (21). Thus we define

$$V = \sum_{p \in \mathsf{P}} \gamma_p V_{A_p}$$

with weights defined by

$$z = \sum_{p \in \mathsf{P}} \operatorname{dist}(A_p, A_p^*) \quad \text{and} \quad \gamma_p = \operatorname{dist}(A_p, A_p^*)/z.$$

Theorem 5.2 implies that V is a continuous Lyapunov function for $(X, \mathsf{M}, \mathsf{P}, f)$.

To simplify notation we will abuse notation and write $r = \Pi(p)$ when $\Pi(A_p) = \mathcal{A}_r$. Now we can reorder the sum so that

$$\mathsf{c}\mathcal{V} = \sum_{p \in \mathsf{P}} \beta_{\Pi(p)} \, \mathsf{c}\mathcal{V}_{\mathcal{A}_{\Pi(p)}} + \sum_{r \notin \Pi(\mathsf{P})} \beta_r \, \mathsf{c}\mathcal{V}_{\mathcal{A}_r}.$$

Then

$$\|\mathsf{c}\mathcal{V} - V\|_\infty \leq \sum_{p \in \mathsf{P}} \|\beta_{\Pi(p)} \, \mathsf{c}\mathcal{V}_{\mathcal{A}_{\Pi(p)}} - \gamma_p V_{A_p}\|_\infty + \sum_{r \notin \Pi(\mathsf{P})} \beta_r.$$

Now consider $r \notin \Pi(\mathsf{P})$. By hypothesis, the attractor $A_p = \omega(|\mathcal{A}_r|)$ is join-irreducible. However, $r \neq \Pi(p)$. Since $(A_{\Pi(p)}, A^*_{\Pi(p)})$ is the unique attractor repeller pair that satisfies equation (22), we must have

$$B_\epsilon(A) \cap |\mathcal{A}^*| \neq \varnothing \quad \text{or} \quad |\mathcal{A}| \cap B_\epsilon(A^*) \neq \varnothing,$$

so that $\operatorname{dist}(\mathcal{A}_r, \mathcal{A}_r^*) \to 0$ as $\operatorname{diam}(\mathcal{X}) \to 0$. Since as $\operatorname{diam}(\mathcal{X}) \to 0$ we have $\beta_{\Pi(p)} \to \gamma_p$, $\mathcal{V}_{\mathcal{A}_{\Pi(p)}}$ converges to V_{A_p} uniformly, and $\beta_r \to 0$ for $r \notin \Pi(\mathsf{P})$ by Equation (21), we have $\|\mathsf{c}\mathcal{V} - V\|_\infty \to 0$ as $\operatorname{diam}(\mathcal{X}) \to 0$. □

Now we apply Theorem 5.10 to the situation that occurs in the computations of Section 6. As in the proof of Theorem 5.10, the map $\omega(|\cdot|)$ is a lattice homomorphism from $\mathsf{Att}(\mathcal{X}, \mathcal{F})$ to $\mathsf{Att}(X, f)$. For each recurrent component $\mathcal{R}$ of $\mathcal{F}$ with a nontrivial Conley index, the attractor $\Gamma^+(\mathcal{R})$ maps to a distinct attractor under $\omega(|\cdot|)$. If we assume that $\mathsf{Att}(X, f)$ is finite, and the grid size is small enough so that all join-irreducible attractors can be resolved by $\mathcal{F}$, i.e. Π maps onto $\mathsf{Att}(\mathcal{X}, \mathcal{F})$, then $\omega(|\cdot|)$ is also surjective. If every recurrent component is nontrivial, i.e. has a nonempty maximal invariant set, then $\omega(|\cdot|)$ is an isomorphism and the hypotheses of Theorem 5.10 are satisfied. This situation occurs in both computations in Section 6, because the Conley indices of the realizations of the recurrent components are all nontrivial, see Figures 2 and 4.

Finally we briefly comment on how we might attain the goal of an algorithm that both efficient in the number of attractor-repeller pairs that are used and also satisfy a convergence result based only on the information contained in $\mathcal{F}$ without any a priori information about the underlying system. There are two issues to be resolved: (1) there can be infinitely many attractors for f, and (2) one or more of the join-irreducible attractors for f that are necessary for the computation are not well-approximated by any join-irreducible attractor for $\mathcal{F}$.

The first issue we might resolve by choosing some fixed Morse decomposition, which we may determine at some chosen level of resolution, and at all finer resolutions grouping components together to reflect this chosen Morse decomposition.

The second issue is due to spurious recurrent components, and could be handled by a strategy of grouping components as well. In this case we would group components together so that the distances between combinatorial attractors and their dual repellers of the join irreducible pairs were not too small. Pursuing these two strategies both computationally and theoretically will be the subject of future work.

6. **Implementation and Results.** The above algorithms have been implemented and are freely available in the C++ software package `Conley-Morse-Database` [4] [12]. In this section we discuss the performance of the Succinct Grid and Pointer Grid described in Section 2 and computational results for Lyapunov functions for two different dynamical models. We also report a breakdown of the memory usage among subcomponents of our algorithms and discuss scalability prospects.

6.1. **Performance of Succinct Grid vs Pointer Grid.** The `Conley-Morse-Database` software provides C++ classes `PointerGrid` and `SuccinctGrid` implementing both pointer-based and succinct-tree based grids described in Section 2. `SuccinctGrid` makes heavy use of the software package SDSL (Succinct Data Structures Library) [10] which provides a C++ library of succinct data structures including succinct trees and rank-select structures. `PointerGrid`, on the other hand, is a simple pointer-based scheme implemented according to the description given earlier. We measure that the succinct-tree based grid implementation achieves an efficiency of 4.58 *bits* per grid element, which represents a 180-fold increase in space efficiency compared to the pointer grid method we have implemented, which requires 104 bytes per grid element. This is more than the asymptotically achievable 3 bits per grid element (or 1.5 bits per binary tree node – see the discussion at the end of Section 2.2) due to the $o(N)$ usage of the succinct data structures taking approximately 34% of the space in the example we measured. There is the expected time/space trade-off: SuccinctGrid is the slower alternative by a factor of 3. We refer the reader to Table 2 below.

6.2. **Computational Examples.** We have tested our algorithms on two examples, the first being two-dimensional and the second being three-dimensional. Both examples computed in a few hours using a single core on a Macbook Pro laptop with 8GB of RAM. For each example we present a Conley-Morse graph, a visualization of the combinatorial Morse sets, and a visualization of the computed combinatorial Lyapunov function.

6.2.1. *Two-dimensional overcompensatory Leslie Model.* The first example is of the map $f : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} (\theta_1 x_1 + \theta_2 x_2) e^{-\phi(x_1+x_2)} \\ p x_1 \end{bmatrix}, \tag{23}$$

where we choose parameters $\theta_1 = 20.0$, $\theta_2 = 20.0$, $\phi = 0.1$, and $p = 0.7$. The phase space region was taken to be $X = [0, 74] \times [0, 52]$. This model was first

analyzed in Ugarcovici and Weiss [25]. Later in [1] the multiparameter system was examined computationally via Conley-Morse theory. Here we compute the Conley-Morse graph using the program `SingleCMG` and a combinatorial Lyapunov function using `Lyapunov`. `SingleCMG` and `Lyapunov` are both programs in the `Conley-Morse-Database` project.

Resource requirements are described in Table 2 below. We briefly discuss the results. The Conley-Morse graph in Figure 2 has five nodes, corresponding to five combinatorial Morse sets that were found. The labels on these nodes tell us the *Conley index* information. We will not discuss how these are arrived at here, cf. [1]. We content ourselves with describing intuitively what they indicate. To this end we provide the following table. Note that if an invariant set has a certain Conley index, this does not imply that the invariant set is the set indicated in the table. For example, in the model (23), the invariant set with the Conley index of a stable period-3 orbit is actually a 3-part chaotic attractor.

TABLE 1. Conley Index.

Label	Conley Index
(Trivial, Trivial, Trivial)	A trivial index– Conley index of the empty set
($x-1$, Trivial, Trivial)	Conley index of a stable fixed point
(Trivial, $x-1$, Trivial)	Conley index of a saddle point
(Trivial, Trivial, $x-1$)	Conley index of an unstable fixed point
($x-1$, $x-1$, Trivial)	Conley index of a stable invariant circle
(x^T-1, Trivial, Trivial)	Conley index of a stable period-T orbit
(Trivial, x^T-1, Trivial)	Conley index of an unstable period-T orbit

In the phase space picture we can easily see the Morse sets with the Conley indices of an invariant circle and unstable fixed point. The Morse sets with Conley indices of an unstable period-3 orbit and a stable period-3 orbit are small at this resolution so they are highlighted in blue and purple respectively. The Morse set with the saddle-point Conley index corresponds to the origin in the lower left corner, highlighted in red.

We remark that to compute Conley indices of combinatorial Morse sets on the boundary of phase space requires that we extend the phase space slightly into the negative numbers. If we do not, the boundary will cause trivial Conley indices to appear, which we consider an artifact.

In Figure 3 we see what we should expect to see: the combinatorial Lyapunov function is constant on Morse sets, and takes higher values on the Morse sets which are higher in the Conley-Morse graph. In this figure white is the highest value and black is the lowest value.

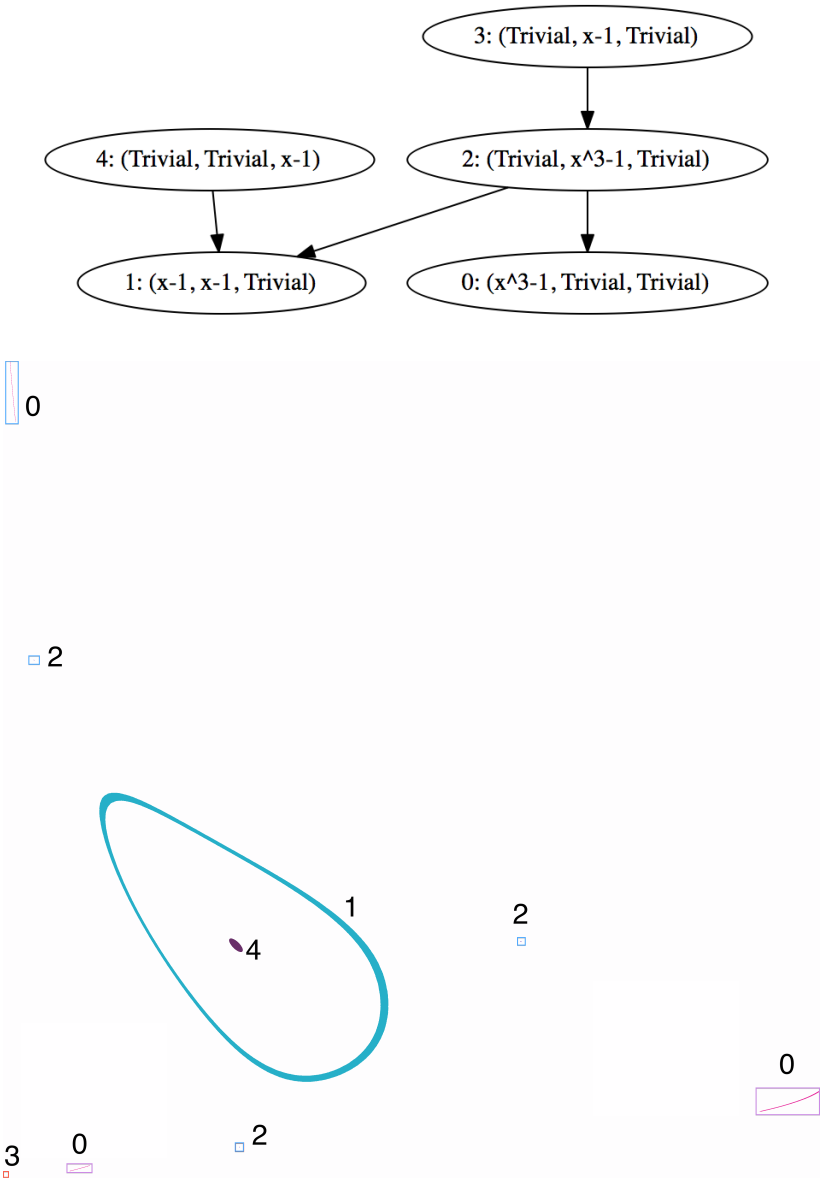

FIGURE 2. Conley-Morse Graph and a depiction of Morse Sets for the overcompensatory Leslie model of Equation (23) as computed by the `SingleCMG` program in the `Conley-Morse-Database` software package.

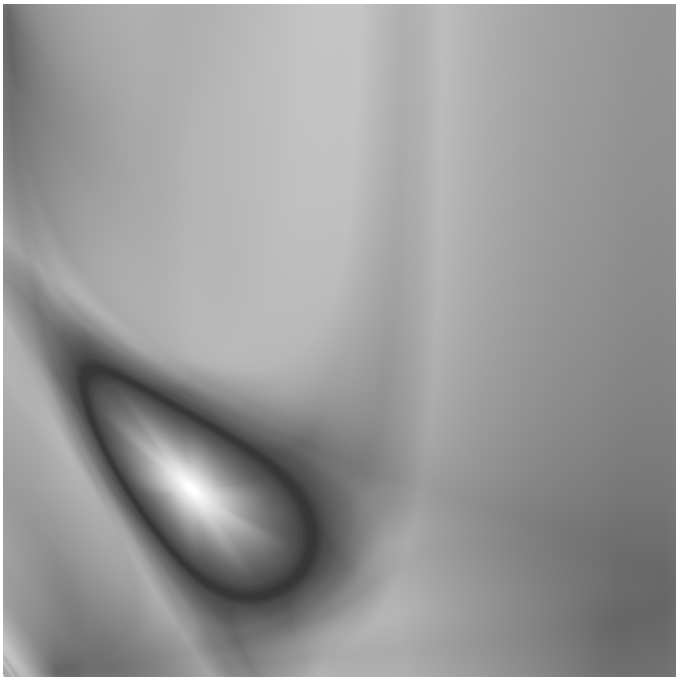

FIGURE 3. Combinatorial Lyapunov function for overcompensatory Leslie model of Equation (23) as computed by the `Lyapunov` program in the `Conley-Morse-Database` software package. The greyscale intensity represents the value of the computed combinatorial Lyapunov function with white corresponding to the highest vale and black the lowest value.

6.2.2. *Three-dimensional Leslie/Gower Competition Model.* Our second example is a three-dimensional model considered by Cushing et. al. [7]. The map is given by

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \mapsto \begin{bmatrix} Ax_2e^{-(Bx_2+Cx_3)} \\ Dx_1 \\ Ex_3e^{-(Fx_1+Gx_3)} \end{bmatrix}, \tag{24}$$

with parameters $A = 5.0$, $B = 0.1$, $C = 0.11$, $D = 0.8$, $E = 5.0$, $F = 0.12$, $G = 0.1$. The phase space region was taken to be $X = [0, 20]^3$. We present our results for this model in Figure 5 and Figure 4. Supplementary materials are available for visualizing the 3D combinatorial Lyapunov function [22].

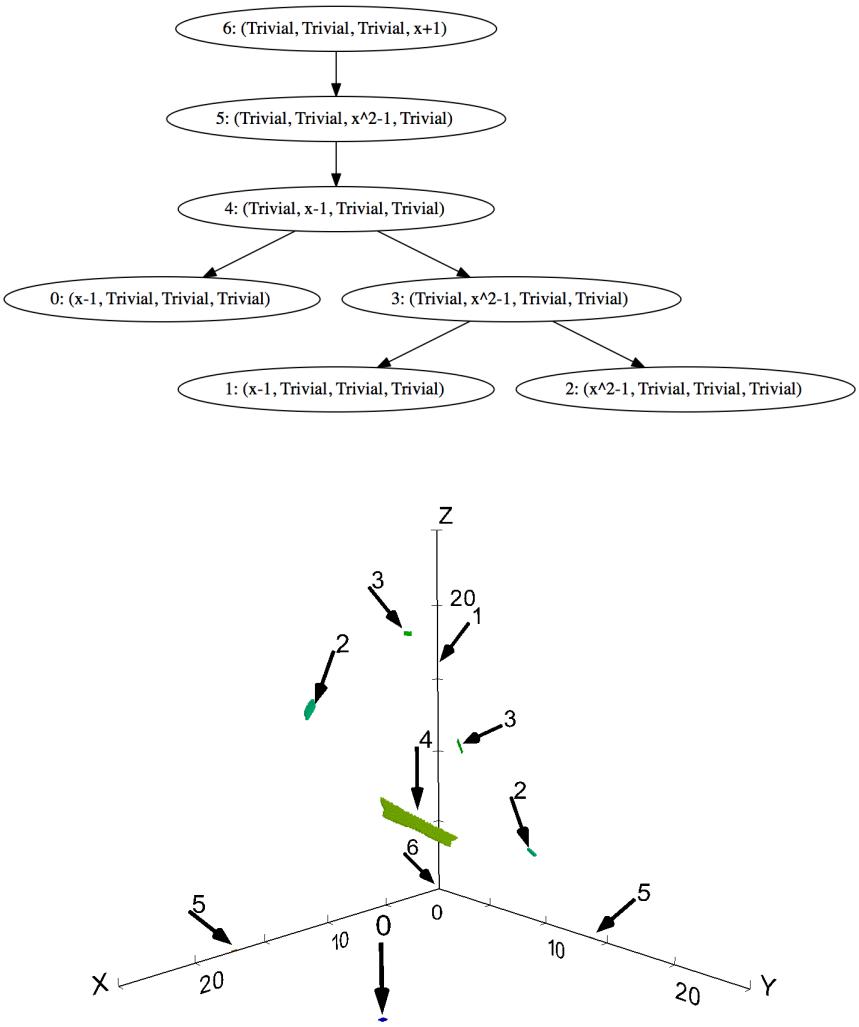

FIGURE 4. Conley-Morse Graph and a depiction of Morse Sets for the Leslie/Gower Model of Equation (24) as computed by the `SingleCMG` program in the `Conley-Morse-Database` software package.

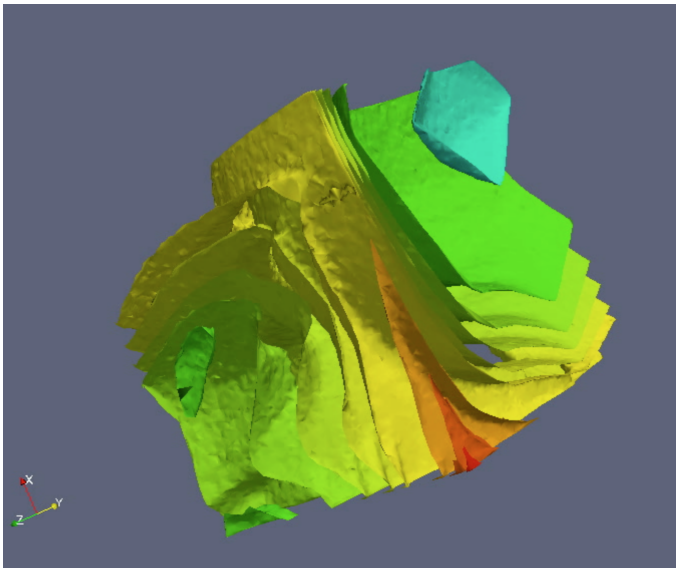

FIGURE 5. Lyapunov function contours for Leslie/Gower Model of Equation (24) as computed by the `Lyapunov` program in the `Conley-Morse-Database` software package.

6.3. **Scalability.** Our space requirements have been reduced with the succinct grid data structure in Section 2.2 and a space efficient technique for computing the strongly connected components, see [11]. We present these resource usage statistics for the computation on the overcompensatory Leslie model of Equation (23) in Table 2.

From this table we observe that had we stored adjacency lists and used pointer-based grids we would have required $3748 + 1398 = 5146$ MB. Dispensing with adjacency list requirements, by itself, allows us to bring this down to 3748 MB. In combination with the succinct grid data structure our space requirements are further brought down to 1704 MB. Thus, we have demonstrated a factor 3 of scaling in this paper.

However, we note that we have reduced memory requirements of objects that require random access patterns, and thus by necessity must be in internal memory. Other memory requirements remain, but it appears that many, if not all of them, can be relegated to external memory. See [26] for a survey of external memory algorithms and data structures. For example, the Dijkstra algorithm's priority queue can be implemented in external memory [2]. This suggests we may have opened the door to orders-of-magnitude scaling via such methods. We leave such an investigation for future work.

References.

TABLE 2. Resource usage for Lyapunov calculation of overcompensatory Leslie Model.

Resource	PointerGrid	SuccinctGrid
Time (`SingleCMG`)	597s	1564s
Time (`Lyapunov`)	5157s	17539s
Grid memory	2054.862424 MB	11.316949 MB
SCC memory	538.803185 MB	same
Lyapunov function memory	316.132656 MB	same
Potential function memory	158.066328 MB	same
Attractor/Dual Repeller memory	4.939572 MB	same
Dijkstra memory	674.415049 MB	same
Total	3748 MB	1704 MB

Grid Elements	Graph Size (hypothetical)
19.758292 million	1397.839640 MB

[1] Zin Arai et al. "A database schema for the analysis of global dynamics of multiparameter systems". In: *SIAM Journal on Applied Dynamical Systems* 8.3 (2009), pp. 757–789.

[2] Lars Arge. "The buffer tree: A technique for designing batched external data structures". In: *Algorithmica* 37.1 (2003), pp. 1–24.

[3] H. Ban and W.D. Kalies. "A Computational Approach to Conley's Decomposition Theorem". In: *Journal of Computational Nonlinear Dynamics* 1 (2006), pp. 312–319.

[4] Justin Bush et al. "Combinatorial-topological framework for the analysis of global dynamics". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22.4 (2012), p. 047508.

[5] Francisco Claude and Gonzalo Navarro. "Practical rank/select queries over arbitrary sequences". In: *String Processing and Information Retrieval*. Springer. 2009, pp. 176–187.

[6] Charles Conley. *Isolated invariant sets and the Morse index*. Vol. 38. CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, R.I., 1978, pp. iii+89. ISBN: 0-8218-1688-8.

[7] JM Cushing et al. "Some Discrete Competition Models and the Competitive Exclusion Principle". In: *Journal of difference Equations and Applications* 10.13-15 (2004), pp. 1139–1151.

[8] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[9] Michael L Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *Journal of the ACM (JACM)* 34.3 (1987), pp. 596–615.

[10] Simon Gog et al. "From Theory to Practice: Plug and Play with Succinct Data Structures". In: *13th International Symposium on Experimental Algorithms, (SEA 2014)*. To appear. 2014.

[11] Shaun Harker. *Space efficient variants of Tarjan's Algorithm for strongly connected components*. 2014. In Preparation.

[12] Shaun Harker, Arnaud Goullet, et al. *Conley-Morse-Database Software Package*. **C**omputational **Hom**ology **P**roject (**CHomP**). 2014. URL: http://chomp.rutgers.edu/Software.html.

[13] Guy Jacobson. "Space-efficient static trees and graphs". In: *Foundations of Computer Science, 1989., 30th Annual Symposium on*. IEEE. 1989, pp. 549–554.

[14] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. "Ultra-succinct representation of ordered trees with applications". In: *Journal of Computer and System Sciences* 78.2 (2012), pp. 619–631.

[15] Bin Jiang. "I/O-and CPU-optimal recognition of strongly connected components". In: *Information Processing Letters* 45.3 (1993), pp. 111–115.

[16] W. D. Kalies, K. Mischaikow, and R. C. A. M. VanderVorst. "An algorithmic approach to chain recurrence". In: *Found. Comput. Math.* 5.4 (2005), pp. 409–449. ISSN: 1615-3375. DOI: 10.1007/s10208-004-0163-9. URL: http://dx.doi.org/10.1007/s10208-004-0163-9.

[17] W. D. Kalies, K. Mischaikow, and R. C. A. M. VanderVorst. "Lattice Structures for Attractors I". In: Journal of Computational Dynamics (2014). accepted.

[18] W. D. Kalies, K. Mischaikow, and R. C. A. M. VanderVorst. "Lattice Structures for Attractors II". In preparation. 2014.

[19] W. D. Kalies, K. Mischaikow, and R. C. A. M. VanderVorst. "Lattice Structures for Attractors III". In preparation. 2014.

[20] J Ian Munro and Venkatesh Raman. "Succinct representation of balanced parentheses and static trees". In: *SIAM Journal on Computing* 31.3 (2001), pp. 762–776.

[21] R. Clark Robinson. *An introduction to dynamical systems—continuous and discrete*. Second. Vol. 19. Pure and Applied Undergraduate Texts. American Mathematical Society, Providence, RI, 2012, pp. xx+733. ISBN: 978-0-8218-9135-3.

[22] *Supplementary Materials*. http://chomp.rutgers.edu/Archives/Lyapunov/SupplementalMaterials.html. 2014.

[23] Andrzej Szymczak. "A combinatorial procedure for finding isolating neighbourhoods and index pairs". In: *Proc. Roy. Soc. Edinburgh Sect. A* 127.5 (1997),

pp. 1075–1088. ISSN: 0308-2105. DOI: 10.1017/S0308210500026901. URL: http://dx.doi.org/10.1017/S0308210500026901.

[24] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[25] Ilie Ugarcovici and Howard Weiss. "Chaotic dynamics of a nonlinear density dependent population model". In: *Nonlinearity* 17.5 (2004), p. 1689.

[26] Jeffrey Scott Vitter. "Algorithms and data structures for external memory". In: *Foundations and Trends® in Theoretical Computer Science* 2.4 (2008), pp. 305–474.

E-mail address: arnaud.goullet@gmail.com
E-mail address: sharker@math.rutgers.edu
E-mail address: dkasti@my.fau.edu
E-mail address: wkalies@fau.edu
E-mail address: mischaik@math.rutgers.edu