Abstract. We describe an automatic chaos verification scheme, based on set-oriented numerical methods, which is especially well suited to the study of area- and volume preserving diffeomorphisms. The novel feature of the scheme is an iterative algorithm for approximating connecting orbits between collections of hyperbolic fixed and periodic points with greater and greater accuracy. The algorithm is geometric rather than graph theoretic in nature and, in contrast to existing methods, does not require the computation of chain recurrent sets. We provide several example computations in dimensions two and three.

Key words. set-oriented numerics, conservative dynamics, automatic chaos verification, connecting orbits

AMS subject classifications. 65G20, 65-04, 65P10, 65P20, 37B30, 37B10, 37C05, 37C29, 37C50

DOI. 10.1137/090776330

1. Introduction. The main goal of this paper is to develop set-oriented algorithms which can be combined with existing tools from computational topology in order to automatically, efficiently, and rigorously verify the existence of chaotic subsystems of conservative dynamical systems. Unlike the dissipative case, we cannot exploit phase space contraction in the design of efficient algorithms. Instead, we fix a priori a particular geometric mechanism called a tangle and develop algorithms which converge iteratively to chaotic subsystems associated with the tangle. We also provide an interface between our algorithms and tools from the discrete Conley index theory. These tools can be used a posteriori to validate the results of our computations. We illustrate the effectiveness of our algorithms in several example computations for area- and volume preserving diffeomorphisms.

Our computations are built on the so-called set-oriented numerical methods, which we discuss formally in sections 2.1 and 2.2. Loosely speaking, set-oriented discretization of a dynamical system consists of discretizing the domain of the system by a finite grid of cubes, and discretizing the dynamics by associating with each cube in the domain a collection of grid cubes which cover its image. With set-oriented methods, one studies the evolution of sets under combinatorial dynamics, rather than following iterates of individual points.

Set-oriented methods have been applied to a wide variety of computational problems in dynamical systems, including orbit design and optimization in celestial mechanics [13, 14, 3, 4], optimal control theory [21], bifurcations giving rise to connecting orbits [15], computation of...

*Received by the editors November 6, 2009; accepted for publication (in revised form) by T. Sauer July 26, 2010; published electronically October 14, 2010. This research was supported by NSF grant DMS 0354567, and by an RTG Fellowship of the University of Texas Department of Mathematics Program in Applied and Computational Analysis.

http://www.siam.org/journals/siads/9-4/77633.html

†Department of Mathematics, Hill Center for the Mathematical Sciences, Rutgers University, 110 Frelinghuysen Rd., Piscataway, NJ 08854 (jjames@math.utexas.edu).
invariant measures on attractors [23], computation of global dynamics over a wide range of parameters in ecological models [1], and automatic chaos verification [9, 7]. Expositions of set-oriented methods are found, for example, in [34, 24, 21].

A typical set-oriented computation consists of two parts. In the first part, qualitative algorithms are used to locate some interesting combinatorial dynamics. The second part is validation, where one attempts to rigorously establish that some properties of the combinatorial dynamics persist in the original system. The validation step is necessary, as the combinatorial discretization of a dynamical system can introduce spurious orbits.

When working with dissipative systems, it is natural to exploit the Conley decomposition theorem [22] during the qualitative portion of the computation. This far-reaching generalization of Morse decompositions states that the phase space of a dissipative dynamical system is decomposed into chain recurrent components and gradient-like sets. Designing set-oriented algorithms based on the Conley decomposition theorem leads to a comprehensive theory of set-oriented computations for dissipative discrete dynamical systems. For details, see [26, 25].

Another powerful qualitative tool in dynamical systems theory is Smale’s tangle theorem [39], which states that the transverse crossing of the stable and unstable manifolds of a hyperbolic fixed point gives rise to a chaotic subsystem called a topological horseshoe factor (defined in section 2.3). In [5], this tangle theorem is weakened to require only topological crossing of the stable and unstable manifolds, and extended to collections of hyperbolic fixed and periodic points. Since crossings of the stable and unstable manifolds of fixed and periodic points give rise to heteroclinic and homoclinic orbits, tangles are often detected and localized in set-oriented computations by searching for combinatorial connecting orbits between fixed and periodic points.

The following set-oriented meta-algorithm sketches the proof of the existence of a topological horseshoe factor in a dissipative dynamical system \((X, f)\), following the outline above. The meta-algorithm is in the spirit of [9, 7].

Algorithm 1.1 (chaos proof meta-algorithm; dissipative case).

function chaosSearch \((X, f)\)

Part 1: (qualitative computation)

\[ \text{Let } (X, F) = \text{discretization of } (X, f) \]

\[ \text{do (until resolution of } X \text{ is fine enough)} \]

\[ \text{Compute } S = \text{chain recurrent part of } X \]

\[ \text{Subdivide } S \]

\[ \text{Let } (X, F) = \text{discretization of } (S, f) \]

end

- Choose a collection of combinatorial fixed or periodic orbits in \(X\).
- Compute combinatorial connecting orbits.

Part 2: (verification)

- Grow an index pair for the special orbits and the connections.
- Compute discrete Conley index of the index pair.

Return the index pair and the Conley index.

End algorithm.
Remark 1.1.
1. Set-oriented discretization is discussed in sections 2.1 and 2.2. We note that the algorithm which computes chain recurrence is combinatorial, as the cubes in $X$ covering the chain recurrent components of $f|_X$ correspond to the strongly connected component of a certain graph (see section 2.1). By subdividing and recomputing the discretization, the iterative step of the algorithm produces a decreasing sequence of finer and finer approximations to the chain recurrent set of the underlying continuous map.
2. Once the chain recurrent set has been localized at high resolution, the combinatorial connecting orbits are typically computed using graph theoretic shortest paths algorithms such as Dijkstra’s algorithm [2, 7]. (A reference on graph algorithms that we found useful is [33].)
3. The verification stage is discussed in section 2.3. There, we recall the formal definitions of an index pair and the discrete Conley index. Studying the homomorphism induced by $f$ on the relative homology of the index pair leads to the proof that $f$ has a chaotic subsystem. (See the worked examples in [24, Chapter 10.7].)
4. In terms of the present work the essential point concerning Algorithm 1.1 is that, since the chain recurrent components of a dissipative dynamical system have measure zero, the iterative procedure in Algorithm 1.1 produces a major reduction in the size of the region of phase space which needs to be searched. Expensive topological and graph theoretic computations need be carried out only on this greatly reduced region.

Conservative systems admit invariant sets of large measure, so that the iterative loop in Algorithm 1.1 is of little or no help. (See Example A.1 in the appendix.) We make the following modification to the meta-algorithm: instead of computing global chain recurrent sets in the qualitative step, we choose a priori some hyperbolic fixed or periodic orbits and apply an iterative geometric algorithm which localizes combinatorial connecting orbits directly. Here follows a sketch of our scheme.

Algorithm 1.2 (chaos proof meta-algorithm; conservative case).

Fix $p$; some hyperbolic fixed or periodic points.

function conservativeChaosSearch ($p, X, f$)  

Part 1: (qualitative computation)  

Let $(X, F) = \text{discretization of } (X, f)$  
do (until resolution of $X$ is high enough)  

Compute $S = \text{reliable combinatorial connecting orbits for } p$  
Subdivide $S$  
Let $(X, F) = \text{discretization of } (S, f)$  

end  

Part 2: (verification)  

:  

return  

Remark 1.2 (results of this work). Our main result is Algorithm 3.5. The algorithm computes $S$ in the iterative loop of Algorithm 1.2. In section 4 we demonstrate, through a series of example computations, that Algorithm 3.5 facilitates successful and efficient completion of Algorithm 1.2.
Remark 1.3.

1. The automation of Part 2 of Algorithm 1.2 is the same as in the dissipative case, and is treated elsewhere [7]. Then in section 4 we focus on the performance of Part 1, i.e., on the execution of Algorithm 3.5.

2. In Examples A.2 and A.3 in the appendix, we look at difficulties which arise if we naively apply graph theoretic shortest path algorithms to computing $S$ in the iterative loop. The material in the appendix motivates the need for Algorithm 3.5.

3. Algorithm 3.5 is not strictly graph theoretic. Rather, the algorithm is based on insights provided by the qualitative theory of dynamical systems, namely, Smale’s tangle theorem [39] and the $\lambda$-lemma (see [35]).

4. By focusing on zero dimensional objects such as tangles, we are able to rapidly cull the phase space of a conservative system in much the same way as is done for dissipative systems in Algorithm 1.1.

5. Algorithm 3.5 is in the spirit of the qualitative theory of Hamiltonian systems developed in [20, 19, 18, 17, 38]. In fact, the present work is best viewed as an attempt to blend the qualitative work on Hamiltonian systems just mentioned with the computational theory of [26, 25] in order to bring conservative systems into the automated chaos verification paradigm of [8] and [9].

6. Strictly speaking, Algorithm 3.5 does not require $f$ to be conservative. In fact, even though a satisfactory theory of automatic chaos verification for dissipative systems already exists, in high enough dimension the cost of computing chain recurrent sets is prohibitive. (Here we are thinking of the Hausdorff dimension of the recurrent set as well as the dimension of the ambient space.) When working with a chain recurrent set of high dimension, a priori focus on some interesting hyperbolic set $p$ and application of a scheme like Algorithm 1.2 could still produce valuable results.

The remainder of the paper is organized as follows. Sections 2.1 and 2.2 review the formal definitions of set-oriented discretization and combinatorial dynamics, as well as the notation and fundamental algorithms of set-oriented numerics. Section 2.3 reviews some notions from the discrete Conley index theory, which we refer to throughout the present work.

In section 3 we develop Algorithm 3.5. Section 4 presents example computations which are meant to demonstrate the utility of the algorithm. Each example highlights some important feature of our methods.

In section 5 we discuss the software implementation and performance of our algorithms. Implementation is discussed in some detail, giving context to the performance results. We also list specific files in which to find the source code of the implementations. Finally, in the appendix we give several motivating numerical examples which highlight some difficulties that arise when applying set-oriented computations to conservative systems.

2. Background and notation. In this section we review material from [26, 9, 5, 24, 22, 25, 40, 32, 12, 42, 43], which we use throughout the remainder of the paper.

2.1. Combinatorial representation of a dynamical system. Let $X \subset \mathbb{R}^n$ and let $f : X \to \mathbb{R}^n$ be a continuous map. We call the pair $(X, f)$ a discrete time topological dynamical system. In order to study a dynamical system using the computer, it is necessary to discretize both $X$ and $f$, meaning that we must choose some finite representation of each. In the
remainder of this section we discuss first the combinatorial discretization of the phase space and then the combinatorial discretization of the dynamics.

A full cube $Q \subset \mathbb{R}^n$ is a product of intervals $Q = [a_1, b_1] \times \cdots \times [a_n, b_n]$, where $a_i < b_i$ for all $1 \leq i \leq n$. A full cube $Q \subset \mathbb{R}^n$ is a uniform cube if $b_i - a_i = r > 0$ for some fixed $r \in \mathbb{R}$ and all $1 \leq i \leq n$. We call $r$ the resolution of the uniform cube.

Let $\mathbb{Z}^n$ be the set of vectors with integer coordinates, and fix both an $\alpha \in \mathbb{R}^n$ and $0 < r \in \mathbb{R}$. The uniform cubical grid with resolution $r$ and origin $\alpha$, denoted $G_{r, \alpha}$, is defined to be the set

$$G_{r, \alpha} = r \cdot \mathbb{Z}^n + \alpha.$$ 

The $r$ and $\alpha$ subscripts are suppressed for simplicity whenever there is no danger of confusion. The elements of $G$ are called grid points.

If $q \in \mathbb{Z}^n$, we say that $q = (q_1, \ldots, q_n)$ are the grid coordinates of the grid point $v = r \cdot q + \alpha \in G_{r, \alpha}$. The unit cube based at $q$ in $\mathbb{Z}^n$ is the cube $[q_1, q_1 + 1] \times \cdots \times [q_n, q_n + 1]$, which we denote $B_q$. The grid cube with base $v = rq + \alpha$ in $G$ is defined to be

$$Q_v = r \cdot B_q + \alpha.$$ 

Note that $Q_v$ is a full uniform cube in $\mathbb{R}^n$ with resolution $r$. To say simply that $Q$ is a grid cube of $G$ means that $Q$ is a grid cube with base $v$ for some unspecified $v \in G$.

Consider a finite collection $A = \{Q_1, \ldots, Q_K\}$ of grid cubes of $G$. The space $\mathcal{X} = \bigcup_{n=1}^{K} Q_n$ is called a uniform full cubical complex in $\mathbb{R}^n$ subordinate to the grid $G_{r, \alpha}$. A cubical complex is a topological space under the subspace topology inherited from $\mathbb{R}^n$. In the remainder of this paper a cubical complex is always a full uniform cubical complex subordinate to some grid $G$.

We often ignore the distinction between the set $A = \{Q_1, \ldots, Q_K\}$ of grid cubes and the space $\mathcal{X} = \bigcup_{n=1}^{K} Q_n$. We abuse the notation and write $Q \in \mathcal{X}$ instead of $Q \in A$, while $B \subset \mathcal{X}$ might mean that $B$ is a point set in $\mathbb{R}^n$ or that $B$ is a subcollection of the cubes in $A$. The meaning will be clear depending on the context.

We come to the definition describing the combinatorial discretization of compact subsets of $\mathbb{R}^n$.

**Definition 2.1 (cubical cover).** Let $X$ be a compact subset of $\mathbb{R}^n$. An outer cubical cover of $X$ is a cubical complex $\mathcal{X}$ containing $X$ in its interior, i.e., having

$$X \subset \mathcal{X}^\circ.$$ 

Throughout this paper, a cubical cover always means an outer cubical cover. Suppose that $\mathcal{X}$ is a cubical cover of a compact set $X \subset \mathbb{R}^n$ such that for each $Q \in \mathcal{X}$, $Q \cap X \neq \emptyset$. Then we say that $\mathcal{X}$ is a minimal cubical cover of $X$.

Now we turn to the discretization of the dynamics generated by $f : X \to \mathbb{R}^n$. Given any $Q \in \mathcal{X}$, an outer cubical covering of $f(Q)$ by cubes of $\mathcal{X}$ is called a combinatorial image of $Q$. More precisely, a combinatorial image for $Q$ is any collection $\{Q_j\}$ of cubes in $\mathcal{X}$ having

$$f(Q) \subset \left( \bigcup Q_j \right)^\circ.$$ 

We refer to this as the outer enclosure property for combinatorial images. The definition is due to [43]. The term symbolic image is used by [34]. A schematic of this construction is given in Figure 1.
The combinatorial dynamics are codified as follows.

**Definition 2.2 (combinatorial enclosure of \( f \)).** For each cube \( Q \in \mathcal{X} \) fix an associated combinatorial image. The association of each \( Q \in \mathcal{X} \) with its combinatorial image defines a combinatorial outer enclosure of \( f \) on \( \mathcal{X} \). The enclosure is a function, which we denote by \( F : \mathcal{X} \rightarrow \text{Pow}(\mathcal{X}) \) (here \( \text{Pow} \) is the collection of all full cubical subsets of \( \mathcal{X} \)). If \( Q \in \mathcal{X} \), then the combinatorial image associated with \( Q \) under \( F \) is denoted by \( F(Q) \). In order to highlight the fact that this is a multivalued function, the notation

\[
F : \mathcal{X} \Rightarrow \mathcal{X}
\]

is used. \( F \) is also referred to as a combinatorial multivalued map. In this paper, combinatorial enclosure always means an outer enclosure.

**Definition 2.3.** The pair \((\mathcal{X}, F)\) is called a combinatorial dynamical system. This is the notion of discretization that we use throughout the paper.

With this notion in place, one can combinatorialize many of the familiar objects of the qualitative theory of dynamical systems. The combinatorial inverse \( F^{-1} \) of \( F \), acting on a cube \( Q_i \in \mathcal{X} \), is defined by

\[
F^{-1}(Q_i) = \{ Q_j \in \mathcal{X} : Q_i \in F(Q_j) \}.
\]

Note that the combinatorial inverse \( F^{-1} \) inherits the outer enclosure property of \( F \), in the sense that

\[
f^{-1}(Q) \subset (F^{-1}(Q))^\circ.
\]

(Here \( f^{-1}(Q) \) is just the usual preimage of the set \( Q \), as we do not require that \( f \) be invertible.)

Let \( A = \bigcup_{i \in I} Q_i \) be a cubical subset of \( \mathcal{X} \). Define the combinatorial image of \( A \) by \( F(A) = \bigcup_{i \in I} F(Q_i) \). Similarly, define \( F^n(A) \) to be the set defined inductively by \( F^0(A) = A \) and \( F^n(A) = F(F^{n-1}(A)) \) for all integers \( n \geq 1 \). Define \( F^{-n}(A) = [F^{-1}]^n(A) \).

The dynamical graph associated with \((\mathcal{X}, F)\), denoted \( \mathcal{G}_{\mathcal{X}, F} \), is the directed graph wherein we associate a node with each \( Q \in \mathcal{X} \) and insert a directed arrow from node \( Q \) to node \( Q' \) if and only if \( Q' \in F(Q) \). We suppress the subscripts when the combinatorial dynamical system

---

**Figure 1.** A cube \( Q \) and its image under \( f \). A combinatorial image of \( Q \) is shown in red. Note that the combinatorial image is a set of grid cubes which contains \( f(Q) \) in its interior.
is understood. The dynamical graph $G^{-1}$ associated with $F^{-1}$ is obtained by reversing the direction of the arrows of $G$.

For $Q \in \mathcal{X}$, a **combinatorial trajectory through $Q$ under $F$** is a function $T_Q : \mathbb{Z} \to \mathcal{X}$ so that $T_Q(0) = Q$ and $T_Q(n + 1) \in F(T_Q(n))$ for all $n \in \mathbb{Z}$. Consider a finite set of consecutive integers $I = \{0, \ldots, n\}$ starting at 0. A **combinatorial orbit segment** $T_Q|_I$ is the restriction of an orbit $T_Q$ to the set of integers $I$. In terms of the dynamical graph, a combinatorial orbit segment $T_Q|_I$ is a graph theoretic path through $Q$.

The following definition is important when discussing the relationship (if any) between the combinatorial dynamics and the underlying dynamics generated by the map $f$.

**Definition 2.4 (false positive).** Let $Q \in \mathcal{X}$, and let $T_Q$ be a combinatorial trajectory through $Q$ under $F$. $T_Q$ is called a **false positive** if there exist $n \in \mathbb{N}$ and $Q' \in \mathcal{X}$ so that $Q' = T_Q(n)$ but $f^n(x) \notin Q'$ for all $x \in Q$.

On the level of the combinatorial graph representation, a false positive occurs when there exists a path in the graph from node $Q$ to node $Q'$ but there is no point in $Q$ with an iterate in $Q'$. This will occur frequently in practice, as a combinatorial image of a cube is required to be an outer enclosure of its pointwise image.

Let $N \subset \mathcal{X}$. The **maximal combinatorial invariant subset of $N$ under $F$** is the set

$$\text{Inv}(N, F) = \{Q \in N : \exists T_Q : \mathbb{Z} \to N \subset \mathcal{X}\}.$$  

Suppose that $S = \text{Inv}(N, F)$ for $N \subset \mathcal{X}$ and $S \neq \emptyset$. The sets

\[
W^s_N(S) = \{Q \in N : \text{there is a } T_Q : \mathbb{Z} \to N \text{ so that } T_Q(n) \in S \text{ for some } n \geq 0\}
\]

and

\[
W^u_N(S) = \{Q \in N : \text{there is a } T_Q : \mathbb{Z} \to N \text{ so that } T_Q(n) \in S \text{ for some } n \leq 0\}
\]

are called the **combinatorial stable** and **unstable sets of $S$**, relative to $N$.

Let $A \subset \mathcal{X}$. Then

$$C_k = \bigcup_{i=0}^{k} F^k(A),$$

with $k \in \mathbb{Z}$, is called a $k$-**set orbit** of $A$. If $k > 0$, then $C_k$ is called a **forward orbit**, while if $k < 0$, $C_k$ is called a **backward or inverse orbit**. $C_k$ is also referred to as a **globalization** of $A$.

We note that images of cubical subsets inherit the outer enclosure property of individual cubes, as

\[
f(A) = f \left( \bigcup_{i \in I} Q_i \right) = \bigcup_{i \in I} f(Q) \subset \bigcup_{i \in I} [F(Q)]^\circ
\]

\[
= \left[ \bigcup_{i \in I} F(Q) \right]^\circ = [F(A)]^\circ.
\]

**Remark 2.5.**

1. It is perfectly acceptable to have $F(Q) = \emptyset$ for some $Q \in \mathcal{X}$. This happens, for example, whenever $f(Q) \cap \mathcal{X} = \emptyset$. On the graph level, $Q$ corresponds to a node with no outgoing arrow.
2. Our numerical implementation of the discretization operator, which assigns a cubical image \( F(Q) \) for each \( Q \in \mathcal{X} \), is explained in detail in section 5.

3. We make the following definition: Suppose that for some \( Q \in \mathcal{X} \) and \( x \in Q \) there is a \( Q' \in \mathcal{X} \) so that \( f^n(x) \in Q' \), but that \( F \) does not admit a combinatorial trajectory through \( Q \) with \( T_Q(n) = Q' \). In this case we say that \( F \) admits a false negative.

In terms of the graph theoretic representation of the dynamics, a false negative is said to occur if there is a point \( x \in Q \) and a positive integer \( k \in \mathbb{N} \) so that \( f^k(x) \in Q' \), yet there is no directed path in the graph connecting node \( Q \) to node \( Q' \). It is essential to the entire discretization process that the outer enclosure property prohibit false negatives.

4. The existence of a nonempty combinatorial invariant subset does not imply the existence of an \( f \)-invariant subset \( S \subset S \), as the combinatorial enclosure \( F \) may admit false positives. This is the reason that a posteriori validation is necessary in the computations described in section 1. On the other hand, we note that the combinatorial discretization of \( f \) can be used to rule out the existence of certain kinds of pointwise orbits. For example, if \( \mathcal{N} \subset \mathcal{X} \) and \( \text{Inv}(\mathcal{N}, F) = \emptyset \), then the maximal \( f \)-invariant subset in \( \mathcal{N} \) is empty, as an outer enclosure does not admit false negatives.

2.2. Set-oriented numerical algorithms. This section presents the core set-oriented algorithms used in the remainder of the paper. First we collect several simple utility functions which we use freely in what follows. The utility functions depend on a cubical complex \( \mathcal{X} \), the underlying grid \( G_{r, \alpha} \), and the cubical multivalued map \( F \), even when this dependence is not explicitly indicated.

Let \( \mathcal{X} \) be a cubical complex subordinate to the grid \( G_{r, \alpha} \) in \( \mathbb{R}^n \). The function \( \#(\mathcal{X}) \) returns the integer number of cubes in \( \mathcal{X} \). Recall that this is finite by definition. Similarly, define the functions

\[
\text{dimensionOf}(\mathcal{X}) = n
\]

and

\[
\text{resolutionOf}(\mathcal{X}) = r.
\]

Let \( \mathcal{N} \subset \mathcal{X} \) be a cubical subset. We define the wrap function

\[
\text{wrap}_\mathcal{X}(\mathcal{N}) \equiv \{ Q \in \mathcal{X} : Q \cap \mathcal{N} \neq \emptyset \}
\]

to be the function which returns \( \mathcal{N} \) and all of its neighbors in \( \mathcal{X} \). The restriction function

\[
F' \equiv \text{restrict}(F, \mathcal{N}),
\]

defined by \( F'(Q) = F(Q) \cap \mathcal{N} \) for all \( Q \in \mathcal{N} \), is the function which returns \( F' : \mathcal{N} \Rightarrow \mathcal{N} \), the cubical outer enclosure defined by \( F \) restricted to \( \mathcal{N} \). The collar function

\[
\text{collar}(\mathcal{N}) \equiv \text{wrap}(\mathcal{N}) \backslash \mathcal{N}
\]
returns the neighbors of \( \mathcal{N} \) which are not in \( \mathcal{N} \).

For a cubical complex \( \mathcal{X} \) subordinate to the grid \( G_{r, \alpha} \), let

\[
\mathcal{X}' = \text{subdivide}(\mathcal{X})
\]
be the cubical complex obtained by including \( X \) in the refined cubical grid \( G_{r/2,\alpha} \). Note that
\[
\#(\text{subdivide}(X)) = 2^n \#(X),
\]
where \( n = \text{dimensionOf}(X) \).

For a dynamical system \((X, f)\), let
\[
(X, F) = \text{enclose}(X, f)
\]
denote a discretization operator which returns a combinatorial dynamical system \((X, F)\), where \( X \) is a cubical enclosure of \( X \) and \( F \) is a combinatorial outer enclosure of \( f \). Our implementation of this operator is discussed in section 5.

Now we come to the algorithms which compute the combinatorial sets defined in section 2.1. Algorithm 2.1 below computes the maximal combinatorial invariant subset of a cubical subset as defined in section 2.1. We also explain how Algorithm 2.1 can be modified to compute stable and unstable sets. The algorithms described in this section are the fundamental building blocks of the set-oriented numerical methods that we develop in the remainder of the paper. For discussion of the convergence and complexity of the algorithms, see [11, 24, 34, 12]. We begin with a theorem.

Theorem 2.6 (combinatorial invariant sets). Let \( X \) be a cubical cover of a compact \( X \subset \mathbb{R}^n \), and let \( F : X \rightarrow X \) be an outer combinatorial enclosure of a continuous function \( f : X \rightarrow X \). Suppose that \( N \subset X \), and define the sequence
\[
S_0 = N,
S_{j+1} = F(S_j) \cap F^{-1}(S_j) \cap S_j.
\]
Then there is a \( K \in \mathbb{N} \) so that
\[
S_K = S_{K-1},
\]
and we have that
\[
S_K = \text{Inv}(N, F).
\]

For the proof, see [24, section 10.6]. Note that neither \( S_K = \emptyset \) nor \( S_K = N \) is ruled out, and that if the sequence is constant at one step, then it is constant at all future iterations.

Similar theorems can be stated and proved for the sets \( W_{s,u}^N(S) \), by considering the partial sequences
\[
W_{j+1}^u = F(W_j^u) \cap W_j^u
\]
and
\[
W_{j+1}^s = F^{-1}(W_j^s) \cap W_j^s,
\]
with \( W_0^{s,u} = N \). More precisely, if \( S = \text{Inv}(N, F) \), then the sequences \( W_j^{s,u} \) converge respectively, and in a finite number of steps, to the sets \( W_{N}^{s,u}(S) \). For more complete discussion of the stable and unstable set algorithms, see [11] or [12].

The constructive nature of Theorem 2.6 gives rise to practical set-oriented algorithms. As an example, we give the pseudocode for the invariant part of the algorithm.
Algorithm 2.1 (compute the combinatorial invariant part of \( \mathcal{N} \)).

function InvariantPart(\( \mathcal{N}, \mathcal{F} \))

\[ S := \mathcal{N}; \]
\[ A := S; \]
\[ S := S \cap \mathcal{F}(S) \cap \mathcal{F}^{-1}(S); \]
\[ \text{while } (S \neq A) \]
\[ \text{return } S \]

The pseudocodes for functions \( W_{ det}(\mathcal{N}) = \text{localStableSet}(\mathcal{N}, \mathcal{F}) \) and \( W_{ unc}(\mathcal{N}) = \text{localUnstableSet}(\mathcal{N}, \mathcal{F}) \) are similar. Note that we call these the local (un)stable set functions as they compute the stable and unstable sets of \( S \) relative to \( \mathcal{N} \), even if the maximal combinatorial invariant subset of \( \mathcal{N} \) is not explicitly given (i.e., \( S = \text{Inv}(\mathcal{N}, \mathcal{F}) \) is not an input to the functions).

2.3. A posteriori validation and discrete Conley index theory. Tools from discrete Conley index theory can be used in order to verify that the underlying dynamical system retains some combinatorial property of its set-oriented discretization. For more complete discussion of the discrete Conley index and its application to rigorous verification, see [24, 31, 26, 8, 7].

Consider a continuous map \( f : X \subset \mathbb{R}^n \to \mathbb{R}^n \) with \( X \) compact.

Definition 2.7. If \( N \subset X \) and \( \text{Inv}(N, f) \subset N^o \), then we say that \( N \) is an isolating neighborhood. A compact invariant set \( S \subset X \) is isolated under \( f \) if there exists an isolating neighborhood \( N \) such that \( S = \text{Inv}(N, f) \).

Note that the set \( \text{invariantPart}(\mathcal{F}, \mathcal{N}) \) isolates the maximal \( f \)-invariant subset of \( \mathcal{N} \), by the outer enclosure property of \( \mathcal{F} \). The question which remains is whether or not the set \( \text{Inv}(f, N) \) is empty.

Definition 2.8. A topological pair \((P_1, P_0)\) with \( P_0 \subset P_1 \subset X \) is called an index pair for \( f \) if it satisfies

- \( \text{Inv}(\text{cl}(P_1 \setminus P_0, f) \subset \text{cl}(P_1 \setminus P_0), f) \) (isolation),
- \( f(P_0) \cap P_1 \subset P_0 \) (positive invariance),
- \( \partial_f(P_1) \subset P_0 \) (exit set).

Here the \( f \)-boundary of a set \( A \subset X \) is defined to be \( \partial_f(A) \equiv \text{cl}(f(A) \setminus A) \cap A \).

Definition 2.9. A compact cubical set \( \mathcal{N} \) is called a cubical isolating neighborhood of \( S \) in \( X \) if

\[ \text{wrap}(\text{InvariantPart}(\mathcal{N}, \mathcal{F})) \subset \mathcal{N}. \]

If \( S \) is a maximal combinatorial invariant set for which there exists a cubical isolating neighborhood, then \( S \) is said to be an isolated invariant set. (We note that weaker notions of isolation appear in the literature; see [37].)

Let \( \mathcal{N} \subset X \) with \( X \) a cubical cover of \( X \), and let \( \mathcal{F} \) be a combinatorial outer enclosure of \( f \) on \( X \). The following combinatorial index pair algorithm, from [24, Chapter 10.6], always stops and produces either a “failure” or a combinatorial index pair for \( f \).
Algorithm 2.2 (compute combinatorial index pair).

```plaintext
function indexPair(N, F)
    S = invariantPart(N, F);
    M = wrapG(S);
    if (M ⊂ N);
        F = restrict(F, M);
        C = collar(S);
        P0 = F(S) ∩ C;
        do;
            lastP0 = P0;
            P0 = F(P0) ∩ C;
            P0 = P0 ∪ lastP0;
        while (P0 = lastP0);
        P1 = S ∪ P0;
        P1 = F(P1);
        P0 = P1 \ S;
        return (P1, P0, P1, P0);
    else
        return “failure”;
endif
```

Let $S = \text{InvariantPart}(N, F)$. The following theorem is from [24, section 10.6].

**Theorem 2.10.** If Algorithm 2.2 returns without failure, then we have the following:

- $S$ is an isolating neighborhood for $f$.
- The pair $(P_1, P_0)$ is an index pair for $f$ and isolates $\text{Inv}(S, f)$.
- $P_1 \subset P_1$ and $P_0 \subset P_0$.
- $f(P_1) \subset P_1$ and $f(P_0) \subset P_0$.

Note that while the inputs to Algorithm 2.2 involve the combinatorial data $(X, F)$ and $N$, the conclusions of Theorem 2.10 tell us about the dynamics of the underlying map $f$ on the topological pair $(P_1, P_0)$. This is a first step toward obtaining rigorous information about $f$ from its combinatorial enclosure. However, Algorithm 2.2 and Theorem 2.10 are insufficient, as the theorem does not guarantee that $S \neq \emptyset$.

The following lemma can be found in [24, section 10.6].

**Lemma 2.11.** Let $f$, $P_1$, $P_0$, $\bar{P}_1$, and $\bar{P}_0$ be as in Algorithm 2.2. Then the homomorphism induced on relative homology by the inclusion map $\iota : (P_1, P_0) \hookrightarrow (\bar{P}_1, \bar{P}_0)$ is in fact an isomorphism. It follows that

$$\iota_*^{-1} : H_*(\bar{P}_1, \bar{P}_0) \to H_*(P_1, P_0)$$

is well defined.

Theorem 2.10 implies that $f$ is a pair map on $(P_1, P_0)$ and induces a homomorphism on relative homology, which we denote by $f_* : H_*(P_1, P_0) \to H_*(\bar{P}_1, \bar{P}_0)$. In order to do numerical computations, it is essential that the induced homomorphism $f_*$ can be computed from only the combinatorial data $F$. See Theorem 7.15 in [24] for technical details, and see [36] for numerical implementation.
Definition 2.12. Define the index map $f_P$ of $f$ relative to the pair $P = (P_1, P_0)$ by

$$f_P = \iota^{-1} \circ f : H_s(P_1, P_0) \to H_s(P_1, P_0).$$

The index map is the basis of the definition of the discrete Conley index. See, for example, [31] or [24]. However, it is not necessary to exploit the full power of the discrete Conley index in order to establish the existence of a chaotic subsystem. Instead, one can use the following notion of Lefschetz number on pairs.

Definition 2.13. Let $S \subset \mathbb{R}^n$ be an isolated invariant set for the map $f$, and let $P$ be any index pair for $S$. The Lefschetz number of $S$ relative to $P$ is defined to be

$$L(S, f) = \sum_{j=0}^{n} (-1)^j \text{tr}(f_P^*) = \sum_{j=0}^{n} (-1)^j \text{tr}(f_{P_1, P_0}).$$

Theorem 2.14. If $L(S, f) \neq 0$, one has not only that $\text{Inv}(S, f) \neq \emptyset$ but also that the invariant set actually contains a fixed point.

See [40] for the proof and details.

We close this section with an example application of the discrete Conley index machinery to the problem of “chaos verification.”

Example 2.15 (verification of a topological horseshoe factor.) We begin with the following definition.

Definition 2.16. The dynamical system $(M, f)$ is said to admit a topological horseshoe factor if there exists a compact subset $X \subset M$ so that $f|_X$ is semiconjugate to $\Sigma_A$, where $\Sigma_A$ is a subshift of finite type with positive topological entropy.

For the formal definition of a subshift of finite type and of topological entropy, see Chapters 1.9 and 3.1, respectively, of [27].

Let $S$ and $(P_1, P_0)$ be as in Algorithm 2.2.

Theorem 2.17. Suppose that $(P_1, P_0) \subset X$ is the finite union of disjoint, compact pairs, $(P_1, P_0)_1, \ldots, (P_1, P_0)_m$, and define the sets $N_i = \text{cl}(P_1 \setminus P_0)$. Denote by $f_{N_i}$ the map $f|_{N_i}$, and by $\{i_1, \ldots, i_K = i_1\}$ a finite length-$K$ periodic sequence of the numbers $1, \ldots, m$. Then if

$$L(N_j, f_{N_{i_1}} \circ \cdots \circ f_{N_{i_K}}) \neq 0,$$

there exists a periodic orbit of period $K$ in $S$ that passes through the regions $N_{i_1}, \ldots, N_{i_K}$ in $i_1, \ldots, i_K$ order.

The result can be found in [7] and [40], along with the proof.

Theorem 2.17 can be used to verify the existence of a horseshoe factor as follows. Define the preliminary $m \times m$ transition matrix $\bar{A}$ for the index pair by

$$\bar{a}_{ij} = \begin{cases} 1 & \text{if } F(N_j) \cap N_i \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

These are the combinatorial or potential connecting orbits. Theorem 2.17 is used to verify the existence of $f$-orbits corresponding to the transitions suggested by $\bar{A}$. Since it may be impossible to verify all the potential transitions, one typically has to settle for a smaller
matrix $A$ of verified transitions, where $A$ contains the entries of $\tilde{A}$ which correspond to cycles having nonzero Lefschetz index. The topological entropy of $\sigma_A : \Sigma_A \to \Sigma_A$ is equal to the natural logarithm of the spectral radius of $\tilde{A}$ (see [27, section 3.2]). If the topological entropy of $\sigma_A$ is greater than zero, then $f$ has a horseshoe factor and hence a chaotic subsystem.

A detailed example of the argument just sketched is found in [24, Chapter 10.6]. In [7], the authors develop algorithms which automate this analysis. Using their automated procedure, they are able to verify a semiconjugacy between an index pair for the Hénon map and a subshift of finite type on 199 symbols.

3. Top-down set-oriented computation of combinatorial connecting orbits for conservative dynamical systems. In this section we present Algorithm 3.5, the main result of the paper. The algorithm executes a top-down search for combinatorial connecting orbits between hyperbolic objects. This constitutes an implementation of the heuristic set-oriented search mentioned in Algorithm 1.2. Here follows an outline.

Algorithm 3.1 (connecting orbit meta-algorithm).

Step 1: Local computation about some hyperbolic object.
Step 2: Globalize stable and unstable sets in order to cover some connecting orbits.
Step 3: Restrict, subdivide, and repeat.

In order to formalize this procedure, we focus on computing homoclinic orbits for a hyperbolic fixed point and elaborate on each step. Some modifications are discussed in section 4.

Step 1: Local computation. Let $p$ be a hyperbolic fixed point of a continuous map $f : X \subset \mathbb{R}^n \to \mathbb{R}^n$. Let $X_0$ be a uniform cubical cover of $X$, and $F_0 = \text{enclose}(X_0, f)$. Take $Q_0$ to be the smallest collection of cubes covering $p$. Throughout this section, subscripts denote the level of set-oriented subdivision, so that, for example, $X_0$ indicates the initial unsubdivided cubical domain.

We must assume that $X_0$ is of high enough resolution that the following isolation condition is met: We require that there be a connected cubical set $Q_0$ with $Q_0 \subset \overline{Q}_0$ and such that $\overline{Q}_0 = \text{invariantPart}(\text{wrap}(Q_0), F_0)$. In other words, $\overline{Q}_0$ is the smallest isolated cubical invariant set containing $p$.

Example 3.1. The following example illustrates the need for the distinction between $Q_0$ and $\overline{Q}_0$. Consider the linear map given by the matrix

$$L = \begin{pmatrix} 10/9 & 0 \\ 0 & -9/10 \end{pmatrix},$$

and take $\mathcal{X}_0$ to be a cubical complex covering $[-2,2]^2$ and having resolution $r = 0.5$, and $\mathcal{L}$ to be an enclosure of $L$. Note that $Q_0$ is the four cubes making up the square $[-0.5,0.5] \times [-0.5,0.5]$ (as these are the cubes containing the origin).

To see that $Q_0$ is not isolated, take $Q = [0,0.5] \times [0.5,1] \in \mathcal{X}_0$ and note that $Q \notin Q_0$. $Q$ contains the point $x = (0,0.8)$, and since $L^2(x) = (0,0.64) \in Q$, the cube $Q$ has combinatorial period two. While this period two trajectory is a false positive, it is nevertheless the case that $Q \in \text{Inv}(\mathcal{X}_0, \mathcal{L})$. Since $Q \in \text{wrap}(Q_0)$, the combinatorial fixed point set in this example, while invariant, is not isolated.

Since we do not want to consider $\overline{Q}_0$ as part of our initial data, we give an algorithm for finding it.
Algorithm 3.2 (compute isolated invariant component of combinatorial fixed cubes).

function invariantComponent \((\mathcal{Q}, \mathcal{X}, \mathcal{F})\)

\[ \mathcal{Q} := \mathcal{Q}; \]

\[ \text{do} \]

\[ A := \mathcal{Q}; \]

\[ B := \text{wrap}(A); \]

\[ C := \text{invariantPart}(B, \mathcal{F}); \]

\[ \mathcal{Q} := C; \]

\[ \text{while } (A \neq \mathcal{Q}) \]

\[ \text{if } (\mathcal{Q} \cap \partial \mathcal{X} \neq \emptyset); \]

\[ \text{return } \text{“failure”}; \]

\[ \text{else} \]

\[ \text{return } \mathcal{Q} \]

\[ \text{end} \]

The algorithm stops as \(\mathcal{X}_0\) is a finite collection of cubes, and fails if \(\mathcal{Q}_0\) cannot be isolated in \(\mathcal{X}_0\). Note that Algorithm 3.2 is similar to Algorithm 2 in [8], with the exception that that algorithm returns an extra layer of noninvariant cubes in order to insure that the combinatorial invariant set is isolated. We will deal with isolation separately.

Once we have located the invariant neighbors of the cubes covering the fixed point, the next step in the local computation is to compute a cubical neighborhood \(\mathcal{N}_0\), which is \emph{as large as possible}, and which isolates \(\overline{\mathcal{Q}}_0\).

Algorithm 3.3 (expand the isolating neighborhood about \(\overline{\mathcal{Q}}\)).

function maxIsolatingBlock \((\overline{\mathcal{Q}}, \mathcal{X}, \mathcal{F})\)

\[ S := \overline{\mathcal{Q}}; \]

\[ W := \text{wrap}(S); \]

\[ S' := \text{invariantPart}(W); \]

\[ \text{while } (S = S'); \]

\[ S := W; \]

\[ W := \text{wrap}(S); \]

\[ S' := \text{invariantPart}(W); \]

\[ \text{end while} \]

\[ \mathcal{N} := S; \]

\[ \text{if } (\mathcal{N} = \overline{\mathcal{Q}}); \]

\[ \text{return } \text{“failure”}; \]

\[ \text{else} \]

\[ \text{return } \mathcal{N} \]

\[ \text{end} \]

Denote the output by

\[ \mathcal{N}_0 = \text{maxIsolatingBlock}(\overline{\mathcal{Q}}_0, \mathcal{X}_0, \mathcal{F}_0). \]

This algorithm wraps \(\overline{\mathcal{Q}}_0\) until a further wrap would introduce new invariant cubes. If \(\overline{\mathcal{Q}}_0\) cannot be isolated, then the algorithm fails. However, if \(\overline{\mathcal{Q}}_0\) is the output of Algorithm 3.2, then \(\overline{\mathcal{Q}}_0\) is preconditioned for successful execution of Algorithm 3.3. Again, note that Algorithm 3.3 differs from Algorithm 2 in [8]. Their algorithm returns a set which is composed of invariant
cubes and their immediate noninvariant neighbors. In our algorithm the isolation depth is as deep as possible.

Once we have computed this maximal isolating block $\mathcal{N}_0$, we apply the stable and unstable set algorithms to obtain

$$W^u_{\text{loc},0} \equiv \text{localUnstableSet}(\mathcal{N}_0, F_0)$$

and

$$W^s_{\text{loc},0} \equiv \text{localStableSet}(\mathcal{N}_0, F_0).$$

The sets $\overline{Q}_0$, $\mathcal{N}_0$, and $W^s_{\text{loc},0}$ constitute the local data and complete Step 1 of Algorithm 3.1.

Remark 3.2 (global meaning of $\mathcal{N}_0$). Consider the $f$-dynamics on $\mathcal{N}_0$. If

$$x \in Q \in \mathcal{N}_0 \setminus (W^s_{\text{loc},0} \cup W^u_{\text{loc},0}),$$

then the pointwise orbit of $x$ under the true dynamics $f$ leaves $\mathcal{N}_0$ in a finite number of both forward and backward iterations. By Remark 2.5.3, the subdivision operation can increase neither an invariant set nor its local stable or unstable sets. Then (3.1) persists as we increase the resolution, and any homoclinic orbits for $p$ must pass through $W^s_{\text{loc},0} \cup W^u_{\text{loc},0}$. It follows that the cubes in $\mathcal{N}_0 \setminus (W^s_{\text{loc},0} \cup W^u_{\text{loc},0})$ are of no further interest during the remainder of the computation.

Step 2: Globalize. The idea now is to iterate the local stable and unstable cubical sets until we are confident that their intersection covers a homoclinic connection. An efficient globalization algorithm must strike a balance between two considerations: globalize long enough to ensure that a connection is covered, but not so long that the intersection contains too many unwanted cubes.

Our globalization scheme is guided by the $\lambda$-lemma, which says that the stable and unstable manifolds must accumulate on themselves whenever there is a tangle. This provides the following heuristic stopping condition: we globalize the stable and unstable sets until the globalization intersects the local block $\mathcal{N}_0$. Once this occurs, we are confident that a homoclinic excursion is covered. (Of course, a full verification is obtained only in the second, a posteriori stage of the computation, as discussed in the introduction.)

This globalization scheme is formalized in the following algorithm.

Algorithm 3.4 (globalization algorithm for the local unstable cover).

function globalizeUnstableSet ($\mathcal{X}$, $W^u_{\text{loc}}$, $\mathcal{N}_{\text{loc}}$, $\mathcal{F}$)

$A := \text{wrap}_X(W^u_{\text{loc}}) \cap \mathcal{N}_{\text{loc}}$;

$B := A \setminus W^u_{\text{loc}}$;

$W^u := \mathcal{F}(W^u_{\text{loc}})$;

while $(W^u \cap B = \emptyset)$;

$W_{\text{last}} = W^u$;

$W^u := \mathcal{F}(W^u)$;

if $(W^u = W_{\text{last}})$

return “failure”;

end if

end while

return $W^u$
This algorithm stops as $F^k(W^u_{\text{loc},0})$ is eventually constant (I_0 consists of only finitely many cubes). The algorithm fails if the sequence becomes constant without intersecting $B$. We denote the output of the algorithm by

$$W^u_{0} = \texttt{globalizeUnstableSet}(W^u_{\text{loc},0}, N_0, F).$$

The globalization of the stable set is simply

$$W^s_{0} = \texttt{globalizeUnstableSet}(W^s_{\text{loc},0}, N_0, F^{-1}).$$

Now let

$$I_0 = (W^u_0 \cap W^s_0) \setminus N_0$$

and

$$D_0 = W^u_{\text{loc},0} \cup W^s_{\text{loc},0}.$$

$I_0 \cup D_0$ is our candidate for a cubical cover of a connecting orbit for $p$.

\textbf{Step 3:} Restrict, subdivide, and repeat. Let

$$X_1 = \texttt{subdivide}(I_0 \cup D_0)$$

define the new domain for the next loop through Algorithm 3.1.

To loop the algorithm we repeat the steps from the base case (just described), with one important difference: there is no need to recompute the isolating neighborhood, as any subdivision of $N_0$ continues to isolate $p$. We let

$$N_1 = \texttt{subdivide}(D_0)$$

and throw away all data from the earlier stage of the computation. The general iterative step is the same.

This discussion is formalized in the following pseudocode. The input to the algorithm is the cubical cover $X_0$, the fixed cubes $Q_0$, the map $f$, and the stopping resolution $r_{\text{stop}}$.

\textbf{Algorithm 3.5 (top-down search for a combinatorial homoclinic excursion).}

\begin{verbatim}
function connectingOrbitSearch (X_0, Q_0, r_{\text{stop}}, f)
    F := enclose(X_0, f);
    Q := invariantComponent(Q_0, F);
    N := maxIsolatingBlock(Q, F);
    W^u_{\text{loc}} := localStableSet(N, F);
    W^u_{\text{loc}} := localUnstableSet(N, F);
    W^s := globalizeStableSet(W^s_{\text{loc}}, N_{\text{loc}}, F);
    W^u := globalizeStableSet(W^u_{\text{loc}}, N_{\text{loc}}, F);
    D := (W^s_{\text{loc}} \cup W^u_{\text{loc}});
    I := (W^s \cap W^u) \setminus D;
    while (diameter(I) ≥ r_{\text{stop}});
        X := subdivide(I \cup D);
        N := subdivide(D);
        F := enclose(X, f);
\end{verbatim}
\[ W^s_{\text{loc}} := \text{localStableSet}(N, F); \]
\[ W^u_{\text{loc}} := \text{localUnstableSet}(N, F); \]
\[ W^s := \text{globalizeStableSet}(W^s_{\text{loc}}, N, F); \]
\[ W^u := \text{globalizeStableSet}(W^u_{\text{loc}}, N, F); \]
\[ D := (W^s_{\text{loc}} \cup W^u_{\text{loc}}); \]
\[ I := (W^s \cap W^u) \setminus D; \]

end while

return connectingOrbit := invariantPart(I \cup D);
end;

The algorithm fails if any of its subfunctions fail. The algorithm always stops as \( r_{\text{stop}} > 0 \), and the subdivide operator increases the grid resolution at a geometric rate.

Remark 3.3.

1. In practice, \( Q_0 \) can be computed from the dynamical graph by considering nodes with self connections.
2. If \( X_0 \) contains a large number of cubes, the process of repeatedly wrapping a set and computing the invariant part of the wrap is computationally expensive. However, Algorithm 3.3 is called only once during the initial stage of Algorithm 3.5.
3. If maxIsolatingBlock returns \( X_0 \), then the only invariant cubes in \( X_0 \) are \( \overline{Q_0} \), and there is no possibility of detecting a horseshoe. This means either that \( X_0 \) does not cover a horseshoe or that all the interesting dynamics are covered by \( \overline{Q_0} \) (in which case the initial resolution is too large).
4. Algorithms 3.2 and 3.3 determine the required balance between the initial number of cubes and the initial resolution. The grid must be fine enough that isolation is achieved in Algorithm 3.2, but not so fine that Algorithm 3.3 is impractical. This trade-off is used in order to choose a satisfactory initial grid.
5. The functions in Algorithm 3.5 which might fail are invariantComponent and the globalization calls. The failure of the first indicates too high an initial resolution. The failure of the second indicates that \( X_0 \) was chosen too small to cover a homoclinic tangle.
6. The geometric stopping condition in the globalization step is decidedly different from the choice of “first intersection” employed in [15]. In [15], the unreliability of the first intersection is exploited to define a hat-function, which picks out parameter values at which bifurcations occur. In our case, however, this unreliability of first intersections will destroy our subdivision scheme. The purpose of the geometric globalization is to increase our confidence that a connecting orbit is covered, before restriction and subdivision.
7. Since we are assuming that \( \overline{Q_0} \) covers the hyperbolic fixed point \( p \), it follows that \( W^s_{\text{loc}, 0} \) actually cover the local stable and unstable manifolds of \( p \) [12].
8. Experimentation is often needed in order to choose an effective stopping condition \( r_{\text{stop}} \). For the examples in section 4 we subdivide until \( I_m \) is composed of several distinct topological components. This indicates that the invariant part of \( X_m \) covers only the desired zero dimensional set (see Figure 7). Vagueness of stopping condition is present in most automated chaos verification schemes and is not an idiosyncrasy of the present work (see [8, 7, 9]).
4. Example computations. In this section we present our main argument, which is that Algorithm 3.5 is an effective and efficient realization of Part 1 of Algorithm 1.2. The argument is empirical and proceeds by examining the output of the algorithm in a series of examples.

It is appropriate to reemphasize the following point, which has already been mentioned in the introduction. Recall that a complete set-oriented verification of chaos consists of two parts: a qualitative computation, which locates a combinatorial approximation to the horseshoe dynamics, and a topological validation stage. Once a high resolution cover of the horseshoe is obtained by qualitative computation, discrete Conley index techniques are used to complete the proof that there is a topological horseshoe. Automation of the Conley index computations is discussed in the literature (see [8, section 4], [24, Chapter 10.6], and [7]).

The exposition in the present section therefore focuses on the novelty of the present work: adaptive computation of high resolution combinatorial approximations of tangle dynamics in conservative systems. We adhere to this focus with two caveats: for each example we grow a cubical index pair for the resulting horseshoe cover, and we provide an interface between our software and certain CHomP executables [36]. Computation of the index map with CHomP is demonstrated in Example 1; however, further (well-documented) computations are required to pass from the index map to a rigorous proof of the existence of a horseshoe.

Here we highlight the most important results of the remainder of this section.

- Example 1 shows that Algorithm 3.5 successfully locates a homoclinic excursion in the standard map. Several figures illustrate the operation of the algorithm; the most important is Figure 7, which shows the output of the algorithm. We also briefly discuss the complexity of the search (relative to a brute force computation).
- In Example 2 we show that Algorithm 3.5 can be modified to compute a heteroclinic tangle in the Suris map. A plot of an index pair for the tangle is given in Figure 8.
- In Example 3 we show that our scheme automatically adapts to the topology of the underlying tangle. To see this we consider a heteroclinic excursion between two fixed points of the area preserving Hénon map. This excursion gives rise to a horseshoe factor on which the dynamics are conjugate to a shift on at least three symbols (compare Figure 9 to Figure 8).
- In Example 4 we show that the utility of our scheme is not limited to the plane, by computing a homoclinic excursion for a fixed point of the ABC map. Three dimensional plots of key steps of the algorithm are given.

4.1. Example 1: Homoclinic tangle at the origin in the standard map. Consider the family of maps $f_\epsilon : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$f_\epsilon(x, y) = \left( \begin{array}{c} x + \epsilon \sin(y) \\ x + y + \epsilon \sin(y) \end{array} \right).$$

$f_\epsilon$ is called the standard map and is the subject of a substantial literature. Figure 2 shows a sample of the phase space dynamics for $\epsilon = 1.2$. At this parameter value the phase space is dominated by the secondary KAM tori in the resonance zone about the elliptic fixed point $(0, 2\pi)$, and the Birkhoff instability zone containing the hyperbolic fixed point at $p = (0, 0)$. Note that, due to the presence of invariant circles, the dynamics of the map are neither minimal nor mixing. Nevertheless we expect an abundance of chaotic orbits in the Birkhoff zone.
Set $X = [-2, 8] \times [-2, 8]$, and let $X_0$ be a $50 \times 50$ cubical grid covering $X$. The grid has resolution $r_0 = 0.2$ and contains 2,500 cubes. Direct computation confirms that the resolution is low enough that `invariantComponent` and `maxIsolatingBlock` execute successfully, yet high enough that they run in negligible time. We impose a stopping resolution of $r_{\text{stop}} = 0.0065$.

Figure 3 shows the results of the local computation and highlights graphically Remark 3.2. The point is that any orbit homoclinic to $p$ must pass through the local stable and unstable sets of $N_0$. Then the remainder of the local block is of no interest. Figure 4 illustrates the globalized covers after the first execution of Algorithm 3.4.

Figure 5 shows the state of the computation at the end of the first block of Algorithm 3.5 (just before the while loop begins). Note that the resonance zone has already been culled from the computation, even though the location of the resonance is not an explicit input to the algorithm. This is an essential feature of our scheme (compare with Figures 13 and 14 below).

Figure 6 shows the set $I$ from Algorithm 3.5 just before exit from the while loop. Note that the covering has resolved itself into several distinct pieces. Finally, Figure 7 shows the output of Algorithm 3.5 for the standard map example.

Table 1 illustrates the complexity of the computation. We note that a “brute force” computation (similar to the computations carried out in [3, 4]) with an initial resolution of $r_0 = 0.00625$ (the final resolution of our iterative algorithm) on the same initial domain $X_0 = [-5, 5] \times [-2, 7]$ would require a $1600 \times 1600$ grid of 2,560,000 domain cubes and as many enclosure computations. Our top-down search, on the other hand, locates the homoclinic excursion with only 30,400 total enclosure computations, or roughly one percent of brute force requirements. The largest set arising in the top-down computation contained 13,400 cubes.
Figure 3. Compute local picture: The combinatorial fixed point $\mathcal{Q}_0$ (black), the local block $N_0$ (light blue), as well as the local stable and unstable sets $W^s_{N_0}(\mathcal{Q}_0)$ and $W^u_{N_0}(\mathcal{Q}_0)$ (red and dark blue, respectively).

Figure 4. Globalize: Iterate the local stable and unstable sets until they return to the local block. Note that the global unstable set (in blue) is underneath the global stable set (in red). This is confused slightly by the fact that the local unstable and stable sets (blue and red within the light blue box) are on top of the global unstable and stable sets once they return to the local (light blue) block.
Figure 5. Restrict and subdivide: The new cubical domain $X_1$ (with $Q_1$ also shown). The figure illustrates the state of the computation after the first restrict and subdivide operation.

Figure 6. In $X_5$ the intersection is resolved into several clear disjoint pieces.

This is less than one percent of the size of the brute force grid. Since our implementation of the union and intersection operations are order $N$ in the set size, this reduction makes a substantial difference in the run time of the program. The final cover $S$ of the homoclinic
Figure 7. Algorithm 3.5 output: Combinatorial approximation to the homoclinic excursion in the standard map.

<table>
<thead>
<tr>
<th>Subdivisions</th>
<th>Number of cubes</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,500</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>828</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>1,860</td>
<td>0.05</td>
</tr>
<tr>
<td>3</td>
<td>4,092</td>
<td>0.025</td>
</tr>
<tr>
<td>4</td>
<td>7,720</td>
<td>0.0125</td>
</tr>
<tr>
<td>5</td>
<td>13,400</td>
<td>0.00625</td>
</tr>
</tbody>
</table>

Remark 4.1 (postprocessing and verification). Here follows a brief discussion of the details associated with Part 2 of Algorithms 1.1 and 1.2. First we obtain an index pair for $S = \text{invariantPart}(x_5)$, by passing the set $S$ into Algorithm 2.2. This algorithm executes successfully. The result is an index pair $P = (P_1, P_0)$ for $S$ (see also the comments in section 5.3).

$P$ and $F$ can then be passed into the program $\text{homcubes}$ (part of the CHomP suite [36]), which computes the relative homology of the pair, as well as the induced homomorphism. The relative homology so obtained is

$$H_0(P_1, P_0) = 0,$$

$$H_1(P_1, P_0) = \mathbb{Z}^{24},$$

$$H_2(P_1, P_0) = 0.$$ 

Intuitively, the relative homology of the pair is isomorphic to the reduced homology of the connection contains roughly 2,200 cubes, or fewer cubes than the initial domain.
literature, it is a nontrivial exercise to piece together the necessary code. Two natural ways
the automation of the remaining Conley index computations is thoroughly covered in the
horseshoe factor, one follows the sketch described in section 2.3. We remark that while
(b) of the existence of a homoclinic excursion in the underlying dynamics.

Through a sequence of generators back to itself. The generator excursion is consistent with
We refer to the algebraic orbit as a generator excursion, as the generator $\alpha_{14}$ is mapped
to the quotient of the index pair by its exit set. The quotient has one topological component, 24 nontrivial 1-generators, and no 2-generators (a
space homotopic to the wedge of 24 circles).

Homocubes also computes an index map, which we represent by listing its action on the
24 generators of $H_1(\mathcal{P}, \mathcal{P}_0)$. The result is shown in Table 2. Note that, in this example,
generator 14 is mapped to itself. Then one expects that the topological component associated
with generator 14 covers the fixed point.

Moreover, one can check that the following orbit on generators (ignoring signs) is present
in the index map:

$$
\alpha_{14} \to \alpha_{12} \to \alpha_{21} \to \alpha_{19} \to \alpha_{6} \to \alpha_{24} \to \alpha_{18} \to \alpha_{4} \\
\to \alpha_{22} \to \alpha_{15} \to \alpha_{16} \to \alpha_{23} \to \alpha_{20} \to \alpha_{5} \to \alpha_{3} \to \alpha_{10} \\
\to \alpha_{17} \to \alpha_{1} \to \alpha_{14}.
$$

We refer to the algebraic orbit as a generator excursion, as the generator $\alpha_{14}$ is mapped
through a sequence of generators back to itself. The generator excursion is consistent with
the existence of a homoclinic excursion in the underlying dynamics.

The intuitive interpretation of the algebraic data is that (a) each topological component
of $\mathcal{P}$ gives rise to roughly one generator on the first level of the relative homology, and
(b) $f_{\mathcal{P}_0}$ describes how these generators, and hence the underlying topological components, are
mapped across each other under $f$. Point (a) holds because each topological component of the
index pair corresponds roughly to an intersection of the stable and unstable manifolds. The
exit sets are due to the stretching of components in the unstable direction. The collapse of the
exit set of a component produces a generator on the first level of homology. Then point (b)
suggests that $f_{\mathcal{P}_0}$ maps the generator $\alpha_i$ associated with a particular topological component $\mathcal{N}_i$
of $\mathcal{P}$ to the generators $f_{\mathcal{P}_0}(\alpha_i)$ of the topological components that are accessible from $\mathcal{N}_i$ under
the application of $f$.

To complete the argument and rigorously establish that the index pair isolates a topological
horseshoe factor, one follows the sketch described in section 2.3. We remark that while
the automation of the remaining Conley index computations is thoroughly covered in the
literature, it is a nontrivial exercise to piece together the necessary code. Two natural ways

\begin{table}
\centering
\caption{Induced homomorphism in the standard map: the action of $f_{\mathcal{P}_0}$ on the generators of $H_1(\mathcal{P}_1, \mathcal{P}_0)$.}
\begin{tabular}{ c c c c }
\hline
$f_{\mathcal{P}_0}(\alpha_1)$ & $\alpha_{12} + \alpha_{14}$ & $f_{\mathcal{P}_0}(\alpha_{13})$ & $\alpha_{3}$ \\
$f_{\mathcal{P}_0}(\alpha_{2})$ & $\alpha_{21}$ & $f_{\mathcal{P}_0}(\alpha_{14})$ & $\alpha_{12} + \alpha_{14}$ \\
$f_{\mathcal{P}_0}(\alpha_{3})$ & $\alpha_{10}$ & $f_{\mathcal{P}_0}(\alpha_{15})$ & $\alpha_{16}$ \\
$f_{\mathcal{P}_0}(\alpha_{4})$ & $\alpha_{22}$ & $f_{\mathcal{P}_0}(\alpha_{16})$ & $\alpha_{23}$ \\
$f_{\mathcal{P}_0}(\alpha_{5})$ & $-\alpha_{3}$ & $f_{\mathcal{P}_0}(\alpha_{17})$ & $\alpha_{1} + \alpha_{2}$ \\
$f_{\mathcal{P}_0}(\alpha_{6})$ & $\alpha_{24}$ & $f_{\mathcal{P}_0}(\alpha_{18})$ & $-\alpha_{4}$ \\
$f_{\mathcal{P}_0}(\alpha_{7})$ & $\alpha_{10}$ & $f_{\mathcal{P}_0}(\alpha_{19})$ & $-\alpha_{6}$ \\
$f_{\mathcal{P}_0}(\alpha_{8})$ & $0$ & $f_{\mathcal{P}_0}(\alpha_{20})$ & $\alpha_{5} - \alpha_{8}$ \\
$f_{\mathcal{P}_0}(\alpha_{9})$ & $\alpha_{10}$ & $f_{\mathcal{P}_0}(\alpha_{21})$ & $-\alpha_{19}$ \\
$f_{\mathcal{P}_0}(\alpha_{10})$ & $\alpha_{17}$ & $f_{\mathcal{P}_0}(\alpha_{22})$ & $\alpha_{15}$ \\
$f_{\mathcal{P}_0}(\alpha_{11})$ & $0$ & $f_{\mathcal{P}_0}(\alpha_{23})$ & $\alpha_{11} + \alpha_{20}$ \\
$f_{\mathcal{P}_0}(\alpha_{12})$ & $\alpha_{21}$ & $f_{\mathcal{P}_0}(\alpha_{24})$ & $-\alpha_{18}$ \\
\hline
\end{tabular}
\end{table}
to implement the remainder of the computation are (i) to use the C++ version of CHomP or (ii) to use the codes developed for [7].

In (i) the C++ version must be used because the executable version of homcubes does not return the generators used in the index map computation (these are computed during the program execution and then discarded). The generators can, however, be accessed using the C++ version of CHomP, and since our implementation is a C++ package, developing the necessary interface should be possible. If this were done, the code for the automated validation computation could be adapted from existing CHomP programs.

The main difficulties with (ii) are that the needed codes are not presently available online and that, once they are available, the codes are written in a combination of Matlab, C++, and GAIO [10], so that developing the necessary interface could be difficult. In any case, both approaches lie outside the scope of the present work, where our main concern is Part 1 of Algorithm 1.2.

4.2. Example 2: Heteroclinic tangle in the the Suris map. The Suris map is defined by

\[
f(\theta, r) = \left( \begin{array}{c}
\theta + r + V'(\theta) + \epsilon P(\theta) \\
r + V'(\theta) + \epsilon P(\theta)
\end{array} \right),
\]

where

\[V'(\theta) = -\frac{2}{\pi} \arctan \left( \frac{\delta \sin(2\pi \theta)}{1 + \delta \cos(2\pi \theta)} \right)\]

and

\[P(\theta) = 2\pi \cos(\pi \theta) \sin(\pi \theta).\]

This map was presented in [41] as an example of a family of area preserving twist maps which is integrable when \(\epsilon = 0\), and where the integrable map contains both elliptic and hyperbolic fixed points. In the integrable system \(W^s(p_1) = W^u(p_2)\) and \(W^u(p_1) = W^s(p_2)\), so the phase space of the integrable map is qualitatively similar to the phase space of the mathematical pendulum. Numerics suggest that for \(\epsilon > 0\) the manifolds split, and a heteroclinic tangle develops.

Consider the map at the parameter values \(\delta = 0.4\) and \(\epsilon = 0.05\). The notable difference between the present computation and Algorithm 3.5 is that here we are trying to compute a heteroclinic excursion. The present computation follows the same meta-search outline given in Algorithm 3.1, with the difference being that the local computations are performed about two distinct fixed points. Then, in the globalization step the stable and unstable sets of one fixed point are continued into the neighborhood of the other, rather than back into themselves.

The computation proceeds as in section 4.1. For this computation we take an initial region \(X = [-1.5, 1.5]^2\), use an initial resolution of \(r_0 = 0.03\), and require a final resolution less than \(r_{\text{stop}} = 0.002\). The algorithm converges after four iterations to a combinatorial connection \(S\) between \(p_1\) and \(p_2\). The resulting index pair is depicted in Figure 8.

4.3. Example 3: Heteroclinic tangle in the area preserving Hénon map. As a final planar example, consider the family of quadratic mappings

\[
f(x, y) = \left( \begin{array}{c}1 + y - ax^2 \\
x \end{array} \right).
\]
This is the area preserving Hénon family. We take $a = 1.2$. At this parameter value no elliptic behavior is readily observable in the map dynamics, and naive phase space sampling of trajectories yields little useful information about the dynamics. Nevertheless the map admits a pair of hyperbolic fixed points, $p_1 = (\frac{-1}{\sqrt{a}}, \frac{-1}{\sqrt{a}})$ and $p_2 = (\frac{1}{\sqrt{a}}, \frac{1}{\sqrt{a}})$. If the stable and unstable manifolds of these intersect transversally, there is still chaos in the region.

Even though we know exactly where the hyperbolic points are located, we do not use this information in the program. Instead we begin with a $50 \times 50$ grid on the square $X = [-7, 7]^2$ and have the program locate the combinatorial fixed cubes as well as the connecting trajectories along the lines discussed in section 3. (The point is that the algorithm is given only qualitative information about where to look for the connecting orbits.)

We take an initial resolution of $r_0 = 0.28$ and require a final resolution of less than $r_{\text{stop}} = 0.005$. The top-down search stops after six subdivisions and returns a combinatorial connection $\mathcal{S}$, which we use to grow the index pair $(\mathcal{P}_1, \mathcal{P}_0)$. The index pair, as well as a number of pointwise trajectories, are shown in Figure 9.

Remark 4.2. Observe that the symbolic dynamics in this example are somewhat more complicated than in the previous examples. As in the case of the Suris map, there is one connection from $p_1$ to $p_2$. However, for this Hénon map there are two connections from $p_2$ back to $p_1$, so that the horseshoe dynamics are semiconjugate to a subshift on three symbols.

The example illustrates the flexibility of our search. The added complexity in the topology of the intersection of the stable and unstable manifolds generates additional connecting orbits and hence more complex symbolic dynamics. The additional complexity is nevertheless handled automatically by our computational scheme.
Figure 9. A phase space sample for the area preserving Hénon map, as well as an index pair \((P_1, P_0)\) about the hyperbolic fixed points. While the phase space sample does not reveal any regular dynamics, the search is able to localize the horseshoe.

4.4. Example 4: Homoclinic tangle in the volume preserving ABC map. Although the algorithms run faster and the pictures are easier to render and view in the plane, the utility of the scheme developed in this work is not limited to the study of two dimensional maps. Consider the three parameter family of maps defined by

\[
\begin{pmatrix}
x_{n+1} \\
y_{n+1} \\
z_{n+1}
\end{pmatrix} =
\begin{pmatrix}
x_n + A \sin(z_n) + C \cos(y_n) \\
y_n + B \sin(x_{n+1}) + A \cos(z_n) \\
z_n + C \sin(y_{n+1}) + B \cos(x_{n+1})
\end{pmatrix}.
\]

This is the so-called ABC map. (Notice that if we evaluate the components in \(xyz\) order, the expression does explicitly define a mapping on \(\mathbb{R}^3\).) The map arises as a truncation of a time-\(\tau\) map for the so-called ABC flow. The ABC flow is a particular stationary solution of the Euler equation, so that the ABC map is a useful toy model of fluid dynamics. The ABC map was introduced in [16] in order to study chaos in the three dimensional volume preserving setting.

It is not hard to show that the phase space contains hyperbolic fixed points. Furthermore, it has been observed (at least empirically) that for many parameter values the ABC map admits chaotic motions [16]. We use the tools developed in section 3 to compute an index pair for a homoclinic excursion from \(p\).

Take (somewhat arbitrarily) \(A = 0.6543\), \(B = 0.562\), \(C = 0.701\), and consider the region \(X = [-3, 7] \times [-1, 9] \times [-3, 7]\) with an initial resolution of \(r_0 = 0.125\). This gives an initial cubical complex containing 512,000 cubes. The large number of initial cubes illustrates the dependence of the complexity of set-oriented methods on the dimension. Note that this is an
Figure 10. The local block $N_0$ grown for a fixed cube.

$80 \times 80 \times 80$ grid. In the plane the same resolution requires only 6,400 cubes, or roughly the same number of initial cubes as used in the standard map example.

Figure 10 shows the cubical fixed point set for this set of parameter values, as well as the local block $N_0$ for one of the fixed points. We apply a homoclinic excursion search similar to Algorithm 3.5 at the fixed point. A close-up of the initial local stable and unstable covers relative to this block is shown in Figure 11.

The images of the low resolution globalized manifolds are hard to render effectively in three dimensions. (The eye makes some sense of the pictures if they are rotating, but still projections onto the page are of little help.) Nevertheless, on the level of algorithms, the search proceeds as before. Figure 12 shows a cover of a homoclinic excursion which was obtained after five subdivisions.

5. Performance and implementation. A C++ implementation of the computational scheme discussed in this work is available at [30]. All references below to timing, performance, and source code refer to this implementation.

5.1. Discretization of a $C^2$ dynamical system. We implement a cubical complex as a vector of cubes, a cube as a vector of intervals, and an interval as a pair of double precision floating point numbers. Two intervals are the same if their end points agree to some specified precision, which we usually take to be $10^{-14}$. Vector containers are implemented using the C++ standard template library class vector (for the standard template library, see [28]). The source code for these elementary data structures is in the files doubleCubical.h and doubleCubical.cc.

To implement the discretization operator $\mathcal{F} = \text{enclose}(\mathcal{X}, f)$ it is necessary to compute, for each cube $Q \in \mathcal{X}$, a combinatorial image with $f(Q) \subset [\mathcal{F}(Q)]^\circ$. To compute the enclosure,
Figure 11. Initial coarse local stable and unstable covers about a fixed cube.

Figure 12. A cover of a homoclinic excursion for the ABC map. The fixed cube component can still be seen in black.

we begin with the linear approximation of $f$ by its derivative and then refine this approximation by bounding the nonlinearities. Assume then that $f \in C^2(\mathcal{X})$. (In the applications considered in this work, $f$ is in fact real analytic.)
Let \( r > 0 \) be the resolution of \( X \), and \( x_Q \) be the center of \( Q \). For any \( x \in Q \) we have, by Taylor’s theorem,
\[
f(x) = f(x_Q) + Df(x_Q) \cdot [x - x_Q] + R(x_Q, x-x_Q),
\]
where the Taylor remainder \( R : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \) can be computed explicitly and has
\[
\sup_{x \in Q} \| R(x,r) \| \leq \frac{1}{2} \| D^2 f \|_{C^0} \frac{nr^2}{4} \equiv R(r).
\]
We call \( R \) the truncation error estimate. Then \( f(Q) \) is approximated by the parallelogram
\[
f(Q) \approx P_Q \equiv f(x_Q) + Df(x_Q)[Q-x_Q],
\]
and the truncation error in this approximation is explicitly known.

These observations form the basis for the following algorithms. The algorithms require that we have analytic expressions for \( f \), \( Df \), and the truncation error estimate \( R \).

**Algorithm 5.1 (compute linear image).**

\[
function \ \text{linearImageOf} \ (Q,f,Df);
\]
\[
x_Q = \text{center of } Q;\]
\[
y = f(x_Q);
\]
for (each \( v_i \) a vertex of \( Q \));
\[
e_i = v_i - x_Q;
\]
\[
T_i = Df(x_Q) e_i;
\]
\[
P_i = y + T_i;
\]
end for
\[return \ \{P_i\};\]

**Algorithm 5.2 (compute an \( n \)-rectangle enclosing the image).**

\[
function \ \text{computeEnclosure} \ (X,Q,f,Df,R);
\]
\[
n = \text{dimensionOf}(X);
\]
\[
r = \text{resolutionOf}(X);
\]
\[
x_Q = \text{centerOf}(Q);
\]
\[
\epsilon = \| R(x_Q,\sqrt{nr}/2) \|;
\]
\[
P_i = \text{linearImageOf}(Q,f,Df);
\]
\[
P = \text{parallelogram} (\{P_i\}_{i=1}^{2^n});
\]
for (1 \( \leq j \leq n \));
\[
I_j = \pi_j(P);
\]
\[
a = \text{leftEnd}(I_j);
\]
\[
b = \text{rightEnd}(I_j);
\]
\[
I'_j = [a-\epsilon, b+\epsilon];
\]
end for
\[
B = I'_1 \times \cdots \times I'_n;
\]
\[
R_Q = \text{include}(X,B);
\]
return \( R_Q \);
Algorithm 5.3 (minimize the combinatorial enclosure).

**function** minimizeEnclosure \((\mathcal{X}, Q, f, Df, R)\);

\(n = \text{dimensionOf}(\mathcal{X})\);

\(r = \text{resolutionOf}(\mathcal{X})\);

\(x_Q = \text{centerOf}(Q)\);

\(\epsilon = R(x_Q, \sqrt{nr}/2)\);

\(P_i = \text{linearImageOf}([Q], f, Df)\);

\(P = \text{parallelogram}([P_i]^{2n}_{i=1})\);

\(P' = \text{expand}(P, \delta)\);

\(R_Q = \text{computeEnclosure}(\mathcal{X}, Q, f, Df, R)\);

\(\text{for (each } Q_j \in R_Q)\);

\(\text{if } (Q_j \text{ is outside } P')\);

\(\text{delete}(Q_j, R_Q)\);

\(\text{end if}\);

\(\text{end for}\);

**return** \(F(Q) = R_Q\);

To compute the combinatorial outer enclosure for a map \(f\) on a cubical domain \(\mathcal{X}\), we call Algorithm 5.3 for every \(Q \in \mathcal{X}\). Note that Algorithms 5.2 and 5.3 make use of some shared data. In practice the algorithms can be implemented so that this data is computed only once.

Algorithms 5.1–5.3 are implemented as member functions of a class \(C^2\text{diffeo}\). The maps \(f, Df\), and the truncation estimator \(R\) are passed into the \(C^2\text{diffeo}\) object as template parameters. (Passing the function definitions as template parameters forces the compiler to inline the function calls. See [44, Chapter 7.7].) The source code for this data structure is found in the files \(C^2\text{diffeomorphism.h}\).

The class \(\text{parallelogram}\) is defined in the files \(\text{parallelogram.h}\) and \(\text{parallelogram.cc}\). A parallelogram is implemented as a vector containing the center, a collection of \(2^n\) vertices, and a collection of \(2n\) faces. The class \(\text{face}\) holds a collection of \(2n-1\) references to the vertices of the parallelogram, as well as the outward unit normal vector to the face. The outward direction is defined by requiring that the center of the parallelogram have negative projection.

The call \(P = \text{parallelogram}([P_i]^{2n}_{i=1})\) instantiates a parallelogram \(P\) whose vertices are defined by the collection \(\{P_i\}^{2n}_{i=1}\). In this case the vertices are the images of the vertices of \(Q\) under the linear approximation. The function call \(P' = \text{expand}(P, \delta)\) returns a parallelogram \(P'\) enclosing the parallelogram \(P\), and whose faces are a distance \(\delta\) from and parallel to the faces of \(P\).

Expanding the parallelogram makes use of the outward unit normal directions of the faces. Similarly, in order to test whether a grid cube \(Q_j \in R_Q\) is outside the parallelogram \(P'\), we test to see whether there is a face \(F\) of \(P'\) so that all the vertices of \(Q_j\) are on one side of \(F\). This is tested by choosing a point \(w \in F\) and for each vertex \(v_k \in Q_j\) testing the sign of the projection of \(v_k - w\) onto the unit outward normal of \(F\).

The functions \(\pi_j\) are the canonical projection operators, which in this case return the intervals onto which \(P\) projects. The functions \(\text{leftEnd}(I)\) and \(\text{rightEnd}(I)\) return the left and right end points, respectively, of an interval \(I\). The function \(\text{include}(\cdot, \cdot)\) is best thought of as a combinatorial outer enclosure of the identity map. It is a map which, given a cubical complex \(\mathcal{X}\) and a set of points \(V\), returns the minimal cubical subcomplex of \(\mathcal{X}\) which
covers the points in \( V \). The \textit{include} function is implemented as a member function of the \textit{doubleCubicalSet} class. All parallelogram objects are in scope only during the execution of Algorithm 5.1.

For the maps used in this paper, we have explicit expressions of \( f \) in terms of elementary functions, so that we can explicitly derive formulas for the differential and the error estimate. The standard map, for example, has differential
\[
Df_\epsilon(x, \theta) = \begin{pmatrix} 1 & \epsilon \cos(\theta) \\ 1 & 1 + \epsilon \cos(\theta) \end{pmatrix}
\]
and truncation error estimate
\[
R(x_Q, |h|) = 2\epsilon (|\cos(h_2)| - 1 + |\sin(h_2) - h_2|).
\]
Similar estimates can be worked out for all the examples used in this paper. The expressions for the differentials and the error estimators of all maps used throughout the present work, as well as the expressions of maps themselves, are found in the file \textit{discreteDynamicalSystems.h}.

Rather than employing interval arithmetic to account for round-off error, the software overestimates the number of floating point operations in the discretization process and adjusts the truncation error accordingly. For maps, the computation of a combinatorial image involves only a few hundred floating point operations, so that it is reasonable to expect round-off errors on the order of no more than \( 10^{-14} \). In practice we simply double the (rigorous) truncation error estimate throughout the computation. In the examples in section 4 the highest resolutions occurring in the computations are on the order of \( r \approx 10^{-4} \), so that truncation errors are on the order of \( 10^{-8} \), and doubling the truncation error estimate more than compensates for round-off error. Nevertheless, the inclusion of interval arithmetic into the software could be easily accomplished by modifying only Algorithm 5.1.

\textbf{5.2. Implementation of combinatorial data structures.} The combinatorial data \( \mathcal{F} \) is stored as a vector of ordered sets of integers. For each \( Q \in \mathcal{X} \) we store the index set \( i_1, \ldots, i_n \), where \( \mathcal{F}(Q) = \{Q_{i_1}, \ldots, Q_{i_n}\} \). The sets \( \{i_1, \ldots, i_n\} \) are implemented using the standard template library ordered set class. The source code for the combinatorial enclosure data structure is found in the files \textit{dynamics.h} and \textit{dynamics.cc}.

Once a system is discretized, the remaining combinatorial analysis is based on the operations of union, intersection, and difference for ordered sets of integers. Define the function \( \text{Index} : \mathcal{X} \rightarrow \mathbb{N}^+ \) by
\[
\text{Index}(Q_i) = i,
\]
where \( i \) is the vector index of a grid cube, and for \( \mathcal{A} \subset \mathcal{X} \) let
\[
\text{Indices}(A) = \{i = \text{Index}(Q_i) \mid Q_i \in A\}
\]
be the function that returns the ordered set of indices of \( \mathcal{A} \). If \( S \) is an ordered set of integers with \( S \subset \text{Indices}(\mathcal{X}) \), let
\[
\text{Indices}^{-1}(S) = \{Q_i \in \mathcal{X} \mid i \in S\}
\]
be the cubical subcollection whose indices are the elements of $S$. We implement all of the set operations used in our algorithms on the index level. So, for example, when an algorithm calls for the computation of $A \cup B$, this is evaluated simply by computing the union $\text{Indices}(A) \cup \text{Indices}(B)$.

For integer sets $A$ and $B$ with small cardinality we implement the set operations of union, intersection, and difference using standard template library algorithms. The complexity of the standard template library set operations is $N \log(N)$, with $N$ the cardinality of the larger set. For sets with a large number of elements we implement the set operations using an auxiliary hash, in which case the complexity is reduced to $N$. (This is a standard trick; see [6].)

5.3. Interface with CHomP and graph algorithms. The output of Algorithm 3.5 is a cubical set which presumably covers some homoclinic orbits. To rigorously verify the existence of a topological horseshoe, it is necessary to grow an index pair which can then be processed by CHomP. While Algorithm 2.2 is sufficient for computing the index pair, in practice we use a somewhat more involved procedure which grows an index pair that is preconditioned for the homcubes program and which is minimal in some sense. The interested reader should consult the source code for the functions $\text{indexPairSurgery}$ and $\text{growIndexPairNeighborhood}$. These are defined in the header file $C2diffeomorphism.h$.

In order to process this index pair using CHomP, it is necessary to convert both the index pair $(P_1, P_0)$ and the combinatorial enclosure $F$ into a form appropriate for input into CHomP. The reader interested in this interface can consult the source code for the member function $\text{outToChomp}$ in the class “combinatorialEnclosure.” The class and function are defined and implemented in the files $dynamics.h$ and $dynamics.cc$.

One more comment on implementation. For several computations in the appendix we need to call Dijkstra’s shortest path algorithm. To implement the dynamical graph we use the Boost Graph Library [29], which provides templates for directed graph data structures as well as templates for the standard graph searches, including Dijkstra’s algorithm. Since we have already implemented the combinatorial enclosure data structure as a vector of ordered sets of integers, it is a trivial matter to convert this data into a directed graph. The graph is implemented as an adjacency list, and the Boost Graph Library is equipped with constructors which facilitate this conversion. The source code for our implementation of the dynamical graph class is found in $dynamicalGraph.h$ and $dynamicalGraph.cc$.

5.4. Performance. With these comments on implementation in place, we briefly discuss the performance of our algorithms. We give only the timing results for the discretization of the dynamical systems and the connecting orbit search of Algorithm 3.5. We do not include performance data for the postprocessing and Conley computation stages, as these computations are well documented elsewhere in the literature.

Table 3 gives the timing data for the example computations in section 4. The total program time, time spent in the discretization and set-oriented stages, as well as the percentage of time in each stage of the computation is shown for all four examples. The complexity of the computations depends strongly on the number of cubes in the domain. In order to tie the timing results to domain size, we give the initial and final grid sizes and resolutions in Table 4.

We remark that the differences between the initial resolutions required in each of the examples are related to the dynamics of the given map. The fewest initial cubes are needed
Table 3

<table>
<thead>
<tr>
<th>System</th>
<th>Total run time</th>
<th>Discretization time</th>
<th>Set operations time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hénon map</td>
<td>2.02 sec</td>
<td>1.08 sec (53.46%)</td>
<td>0.72 sec (35.64%)</td>
</tr>
<tr>
<td>Standard map</td>
<td>9.92 sec</td>
<td>5.01 sec (50.5%)</td>
<td>4.91 sec (49.4%)</td>
</tr>
<tr>
<td>Suris map</td>
<td>13.84 sec</td>
<td>9.65 sec (69.73%)</td>
<td>3.59 sec (25.94%)</td>
</tr>
</tbody>
</table>

Table 4

<table>
<thead>
<tr>
<th>System</th>
<th>Initial grid size</th>
<th>Initial $h$</th>
<th>Final grid size</th>
<th>Final $h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hénon map</td>
<td>2,500 cubes</td>
<td>0.28</td>
<td>558 cubes</td>
<td>0.004</td>
</tr>
<tr>
<td>Standard map</td>
<td>6,400 cubes</td>
<td>0.15</td>
<td>4,703 cubes</td>
<td>0.005</td>
</tr>
<tr>
<td>Suris map</td>
<td>10,000 cubes</td>
<td>0.03</td>
<td>2,470 cubes</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Figure 13. Cubical cover of resonance island in the standard map, computed using the invariant part algorithm.

for the area preserving Hénon map, while the Suris map requires the largest initial grid. The reason for this is that we chose to study the Hénon map toward the anti-integrable limit and to study the Suris map near the perturbative regime, while the parameter values for the standard map were taken between the integrable and anti-integrable limits.

Appendix. Direct application of graph theoretic methods to conservative maps. In this appendix we give three examples of how standard set-oriented strategies fail for conservative systems. These examples illustrate explicitly the need for the approach considered in the body of the paper.

Example A.1. One stumbling block for applying the methods of [26] to conservative systems is the presence of chain recurrent subsets of large measure. Figure 13 shows the results of applying Algorithm 2.1 to the standard map, with a $50 \times 50$ cubical covering of the domain $[-3, 3] \times [-1, 7] \subset \mathbb{R}^2$. The resulting invariant cubes are shown in blue.
There is a topological horseshoe factor near the resonance, which we computed in section 3. However, subdividing the chain recurrent set and recomputing the invariant part brings us no closer to isolating the horseshoe. Figure 14 illustrates this: it shows the combinatorial invariant set after two additional subdivisions.

While the subdivision process culls cubes near the boundary of the set, it can never cull cubes in the interior of the resonance due to the existence of invariant circles. If we continue to subdivide this set, the number of cubes grows by roughly a factor of four at each step, but there is no improvement in the culling of the resonance. Compare this to the fast convergence of the subdivision scheme to the Hénon attractor, as discussed in [9] and [7]. The presence of invariant tori in volume preserving maps makes this problem even more pronounced in three dimensions.

Another possible strategy is the direct computation of graph theoretic search procedures for the chain recurrent subset containing the resonance zone. The next two examples illustrate the fact that graph searches are especially unreliable for conservative systems.

Example A.2. In this example we attempt to compute a homoclinic excursion for the hyperbolic fixed point at the origin in the standard map, using Dijkstra’s algorithm. We take the same 50 × 50 grid of 2,500 cubes as in Example A.1. Denote the output of Dijkstra’s shortest path algorithm by $SP$. The result of the search is shown in Figure 15.

The resulting combinatorial excursion is a false positive, as can be verified either graphically (by noting that the orbit passes from the interior to the exterior of an invariant circle) or by subdividing the connection and recomputing the invariant part. If the latter is done, the connection disappears, and it is clear that the shortest path algorithm has failed to locate a true homoclinic excursion.

It could be argued that we have chosen too poor an initial resolution; however, this is the same initial grid used in section 4.1 to compute a horseshoe factor for the standard map. The point of the example is that while the initial grid may be too coarse for the straightforward
Figure 15. Dijkstra shortest path from the fixed cube back to itself (after the self path has been disallowed).

Figure 16. Fixed cubes shown in red and blue. Black cubes are Dijkstra’s shortest path from fixed set to fixed set.

application of graph theoretic algorithms, the grid is not so coarse that we cannot proceed by more geometrical means.

Example A.3. A final example highlights problems associated with the abundance of recurrence in measure preserving systems. The abundance of recurrent orbits tends to make the dynamical graph associated with a conservative system strongly connected, regardless of resolution. Then we subdivide the domain from Example A.2 twice and obtain a cubical grid composed of 40,000 cubes. Applying the Dijkstra shortest path algorithm to the elliptic and hyperbolic fixed points gives the result shown in Figure 16. The red cubes in the center and the blue cubes at the bottom of the figure are the combinatorial fixed cubes, covering neigh-
borhoods of the elliptic and hyperbolic fixed points, respectively. The black cubes show a combinatorial shortest path from the hyperbolic to the elliptic fixed point. It is obvious that, while no such orbit is possible in the underlying dynamics, we must expect an abundance of such false positives near rotational dynamics in conservative systems.

Acknowledgments. The author would like to thank Rafael de la Llave and Konstantin Mischaikow for many useful conversations and much encouragement as this work developed. Thanks also to Maorong Zou, Patrick Goetz, and Jeffery Strunk for substantial help with the computer resources of the UT Math Department. Thanks to Pawel Pilarczyk and J. T. Halbert for carefully reading an earlier draft of the manuscript and for many helpful comments and corrections. Thanks to Hector Lomelí for suggesting the Suris map as an example of an a priori unstable area preserving twist mapping, and to Phil Morrison for suggesting the ABC-mapping as an interesting example of a discrete volume preserving dynamical system. The final draft of the paper also benefited greatly from the thoughtful and thorough comments of two anonymous reviewers.

REFERENCES


