Automatic Dynamic Parallelotope Bundles for Reachability of Nonlinear Dynamical Systems

Edward Kim[1]

[1]University of North Carolina at Chapel Hill

August 2, 2022

Background: Reachability Computation

1. Reachable set computation is an instrumental tool in performing safety analysis over dynamical systems.
2. These computations involve taking an initial set of states and propagating them for a set amount of time to understand the possible states the initial states could reach within the alloted time.
3. Techniques for good over-approximation of reachable sets over classes of *non-linear* systems are topics of active research.

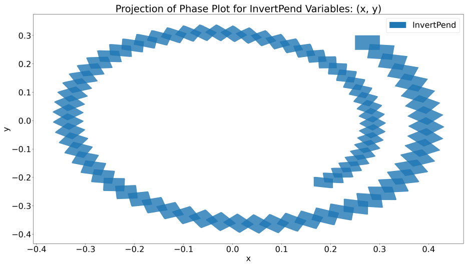


Figure: Plot of Inverted Pendulum System under Initial Conditions $x \in [0.25, 0.3], y \in [0.25, 0.3]$

Background: Reachability Computation

1. The state of a system, denoted as x, lies in a domain $D \subseteq \mathbb{R}^n$. A discrete-time nonlinear system is denoted as

$$x^+ = f(x) \tag{1}$$

   where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear function.
2. The trajectory of a system that evolves according to f, denoted as $\xi(x_0)$ is a sequence $x_0, x_1, \ldots$ where $x_{i+1} = f(x_i)$.
3. The k^{th} element in this sequence x_k is denoted as $\xi(x_0, k)$.

Background: Reachability Computation

1. Given an initial set $\Theta \subseteq \mathbb{R}^n$, the *reachable set* at step k, denoted as Θ_k is defined as

$$\Theta_k = \{\xi(x, k) \mid x \in \Theta\}$$

2. Setting number of steps n and $\Theta_0 = \Theta$, the total reachable set for n steps can be defined as

$$\mathcal{R} = \bigcup_{k=0}^{n} \{\xi(x, k) \mid x \in \Theta\}$$

Background: Reachability with Template Polyhedra

1. One of many techniques in computing the overapproximation of the reachable sets for discrete non-linear systems is to use **template polyhedra** to bound the reachable set.
2. Study of template polyhedra find motivations in the static analysis of programs
3. We are particularly interested in *parallelotopes* as our template polyhedra.

Background: Reachability with Template Polyhedra

Definition

For $\mathbb{R}^n$, a *template polyhedron* is expressed a tuple $\langle H, \mathbf{d} \rangle$ where H is an $m \times n$ real-valued matrix and $\mathbf{d} \in \mathbb{R}^n$ is a real-valued vector. The polyhedron is defined by the conjunction of linear inequalities

$$\bigwedge_{i=1}^{m} H_i \cdot \mathbf{x} \leq d_i$$

where H_i is the i^{th} row of template matrix H and $\mathbf{x} = (x_1, \cdots, x_m)$

Background: Parallelotope Intuition

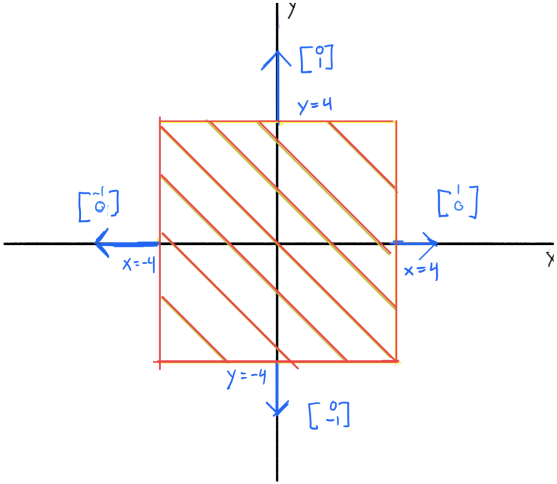
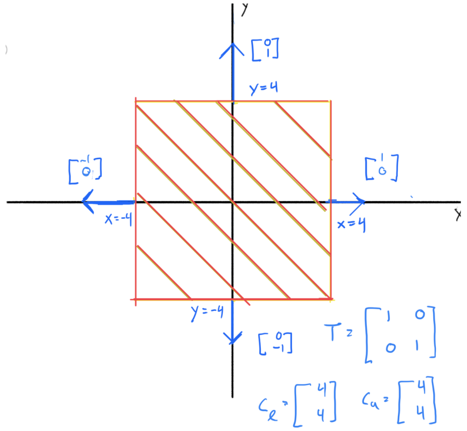

y
$\begin{bmatrix}0\\1\end{bmatrix}$
y=4
$\begin{bmatrix}-1\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
x=-4
x=4
x
y=-4
$\begin{bmatrix}0\\-1\end{bmatrix}$

Figure: Example Initial Parallelotope with offset distance of 4 for all template directions

Background: Parallelotope Matrices



y
$\begin{bmatrix}0\\1\end{bmatrix}$
y=4
$\begin{bmatrix}-1\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
x=-4
x=4
x
y=-4
$\begin{bmatrix}0\\-1\end{bmatrix}$
$T = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$
$c_l = \begin{bmatrix}4\\4\end{bmatrix}$
$c_u = \begin{bmatrix}4\\4\end{bmatrix}$

Figure: Example Initial Parallelotope with template matrix and offset vectors.

Background: Parallelotope Intuition

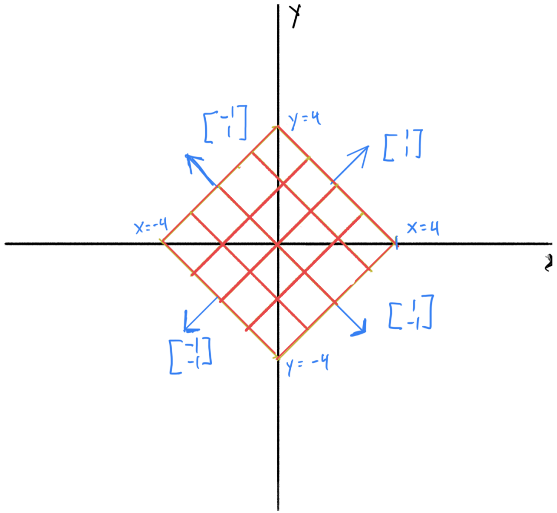

y
x
y=4
y=-4
x=-4
x=4
$\begin{bmatrix}-1\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}-1\\-1\end{bmatrix}$
$\begin{bmatrix}1\\-1\end{bmatrix}$

Figure: Example Initial Parallelotope rotated 45 degrees clockwise

Background: Parallelotope Matrices

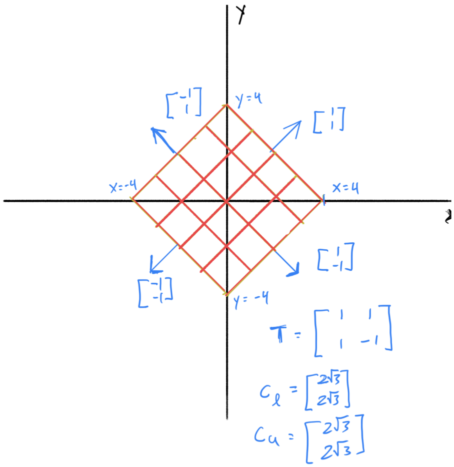


Figure: Example Rotated Parallelotope with template matrix and offset vectors.

Background: Parallelotopes

Definition

A *parallelotope* in $\mathbb{R}^n$ is represented as a tuple $\langle \mathcal{T}, c_l, c_u \rangle$ where $\mathcal{T} \in \mathbb{R}^{n \times n}$ are called *template directions* and $c_l, c_u \in \mathbb{R}^n$ such that $\forall_{1 \le i \le n}\ c_l[i] \le c_u[i]$ are called *bounds*. The half-space representation defines the set of states

$$P = \{\, x \in \mathbb{R}^n \mid c_l[i] \le \mathcal{T}_i x \le c_u[i],\ 1 \le i \le n\}.$$

- The rows $\mathcal{T}_i$ are called *template directions* in the parallelotope.
- Parallelotopes can be thought of cases of template polyhedra where each template direction $\mathcal{T}_i$ has its negative $-\mathcal{T}_i$ also included.

Background: Parallelotope Bundles

Definition

A *parallelotope bundle* Q is a set of parallelotopes $\{P_1, \ldots, P_m\}$. The set of states represented by a parallelotope bundle is given as the intersection

$$Q = \bigcap_{i=1}^{m} P_i$$

- We can think of the bundle's template matrix $\mathcal{T}^Q$ as the conjunction of all the template directions $\mathcal{T}_j^{P_i}$ for the $P_i = \langle \mathcal{T}^{P_i}, c_l^{P_i}, c_u^{P_i} \rangle$
- **Takeaway:** Any convex initial set can be expressed as a parallelotope bundle.

Background: Convex sets can be expressed as Parallelotope Bundles

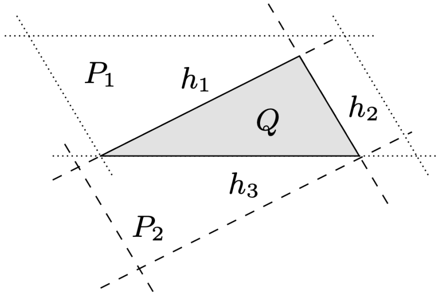


Figure: Figure 1 from Dreossi et. al: Parallelotope Bundles for Polynomial Reachability (2016)

Background: Non-linear Optimization

1. Consider a parallelotopes P in the bundle Q and a non-linear function $f : \mathbb{R}^n \to \mathbb{R}^n$. We want to be able to bound the image $f(P)$ using the template directions defining the parallelotopes in our bundle Q.
2. This amounts to finding new upper and lower bound vectors $c'_u, c'_l \in \mathbb{R}^n$ respectively such that

$$c'_l[i] \leq \mathcal{T}_i^Q \cdot f(x) \leq c'_u[i],\; x \in P$$

for $1 \leq i \leq n$

Background: Non-linear Optimization

1. The bounds can be formulated as non-linear optimization problems:

$$c'_u[i] = \max_{x \in Q} \mathcal{T}_i^Q \cdot f(x)$$
$$c'_l[i] = \min_{x \in Q} \mathcal{T}_i^Q \cdot f(x)$$

2. The reachable set computation is done by iterating this optimization problem to overapproximate the images of the bundles in each step of the computation.

Background: Non-linear Optimization

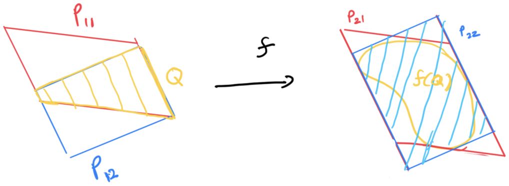


Figure: Toy example of the transformation of a bundle containing two parallelotopes

Background: Algorithm

```
Input: Dynamics $f$, Initial Parallelotope bundle $Q_0$, Step Bound $S$, indexes
       for parallelotopes $I$
Output: Reachable Set Overapproximation $\overline{\Theta}_k$ for each step $k$

for $k \in [1, 2, \dots, S]$ do
    $Q_k = \texttt{TransformBundle}\ (f,\ Q_{k-1},\ \Lambda^{Q_{k-1}})$
    $\overline{\Theta}_k = Q_k$
end
return $\overline{\Theta}_1 \dots \overline{\Theta}_S$

Proc TransformBundle($f$, $Q$, $\Lambda$):
    $Q' \leftarrow \{\}$; $c_u \leftarrow +\infty$; $c_l \leftarrow -\infty$
    for each parallelotope $P$ in $Q$ do
        for each template direction $\Lambda_i$ in the template directions $\Lambda$ do
            $c'_u[i] \leftarrow \mathsf{min}\{\mathsf{\Lambda_i} \cdot \mathsf{f}, c'_u[i]\}$
            $c'_l[i] \leftarrow \mathsf{max}\{-1 \times \mathsf{\Lambda_i} \cdot \mathsf{f}, c'_l[i]\}$
        end
    end
    Construct parallelotopes $P'_1, \dots, P'_k$ from $\Lambda, c'_l, c'_u$ and indexes from $I$
    $Q' \leftarrow \{P'_1, \dots, P'_k\}$
    return $Q'$
```

Background: Bernstein Expansion

1. Previous work focused on reachabilty under discrete non-linear *polynomial dynamics* i.e the non-linear function $f : \mathbb{R}^n \to \mathbb{R}^n$ is polynomial.
2. Our optimization problems become one of optimizing over polynomial $\mathcal{T}_i \cdot f(x)$ which is generally a costly operation.
3. The insight comes from leveraging *Bernstein polynomials*. These polynomials yield an efficient manner of bounding the solution to the aforementioned polynomial optimization problems.
4. An extra linear transformation taking the unit-box domain $[0, 1]^n$ to the parallelotope P is required to use Bernstein polynomials.

Background: Drawbacks

1. Hitherto only *static* parallelotopes have been considered. In other words, the template directions specifying the parallelotopes are to be given as user input at the beginning of the reachable set computation.
2. The template directions chosen are generally the axis-aligned, diagonal directions. However, it's not clear that these directions necessary yield good overapproximations.
3. Since the template directions are set at the beginning, they cannot adapt to the behavior of the dynamics. This could yield overapproximations which are too conservative for any practical use.

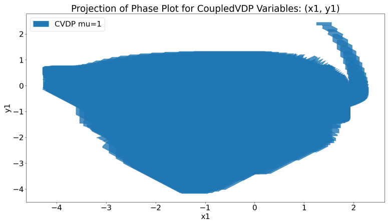


Figure: The effect of choosing inappropriate template directions for the Coupled VanDerPol system.

Contributions

1. We present a method which is both *dynamic* and *automatic*. Our method utilizes the Principal Component Analysis (PCA) and Local Linear Approximations.
2. We extend our tool *Kaa* to leverage NASA's *Kodiak* to perform the optimization procedure.
3. We parallelize our implementation to scale with an increasing number of parallelotopes in our bundles.

Contributions: Automatic, Dynamic Algorithm

1. We choose template directions at each step by two procdure working in tandem
2. One procedure generates new template directions via Principal Component Analysis (PCA) and the other generates template directions through Local Linear Approximations.
3. Each procedure adds one parallelotope to the bundle with a set *template lifespan.* This lifespan dictates the future number of steps the parallelotope and its template directions will exist in the bundle.

Contributions: PCA Directions

1. To generate the PCA directions from bundle Q, we first calculate the *support points* of all template directions in $\mathcal{T}_i^Q$ over the bundle Q.

$$p_i^u = \max_{x \in Q} T_i^Q \cdot x, \quad p_i^\ell = \min_{x \in Q} T_i^Q \cdot x$$

2. We then propagate these support points to the next step by the given dynamics f. This will gives us set of trajectories.

Contributions: PCA Directions

1. We perform PCA on the endpoints of these trajectories to yield a real-valued $n \times n$ matrix whose rows are the PCA directions. These rows are used as template directions to define a single parallelotope P'.
2. The parallelotope P' with template lifespan $T_{P'}$ is added to the bundle Q to give another bundle $Q' = Q \cup P'$.

Contributions: Linear Approx. Directions

1. Observe that the if the dynamics f were linear i.e $x^+ = Ax$ for some linear transformation A, then the image of the parallelotope $c_l \leq \mathcal{T}x \leq c_u$, will be the set $c_l \leq \mathcal{T} \cdot A^{-1}x \leq c_u$.
2. Since the initial parallelotope will be defined by the axis-aligned directions ($\mathcal{T} = I_n$), the image after the first step will be given by

   $$c_l \leq A^{-1}x \leq c_u$$

   Similarly, after the exact image of the initial parallelotope after two steps will problems:

   $$c_l \leq (A^{-1})^2 x \leq c_u$$

Contributions: Linear Approx. Directions

1. Begin computation with $\mathcal{T}_{lin} = I_n$
2. To generate the Linear Approximation templates, we once again calculate the support points of all template directions, $\mathcal{T}_i^Q$, over the bundle Q.
3. Propagate the support points using given dynamics f to the next step. This gives us a set of trajectories S.

Contributions: Linear Approx. Directions

1. Calculate the best-fit linear approximation $\mathcal{A}$ using the start and endpoints of the trajectories. Note that $\mathcal{A}$ will be a real-valued $n \times n$ matrix.
2. Multiply the template directions $\mathcal{T}_{lin}$ by $\mathcal{A}^{-1}$ to yield $\mathcal{T}_{lin} \cdot \mathcal{A}^{-1}$.
3. Define a parallelotope P' with template lifespan $T_{P'}$ and add it to bundle Q to get $Q' = Q \cup P'$.
4. $\mathcal{T}_{lin} \leftarrow \mathcal{T}_{lin} \cdot \mathcal{A}^{-1}$

Contributions: Template Lifespan

1. The template lifespan parameter dictates the number of steps after its creation the parallelotope's template directions exist in the bundle.
2. The intuition is that certain dynamics may be more amenable to directions generated previously in the reachable set computation.
3. Having a lifespan allows the algorithm to be adaptive while attenuating the computational cost of optimizing over a large number of parallelotopes.
4. This is a tunable parameter we vary in our experiments.

Contributions: Template Lifespan

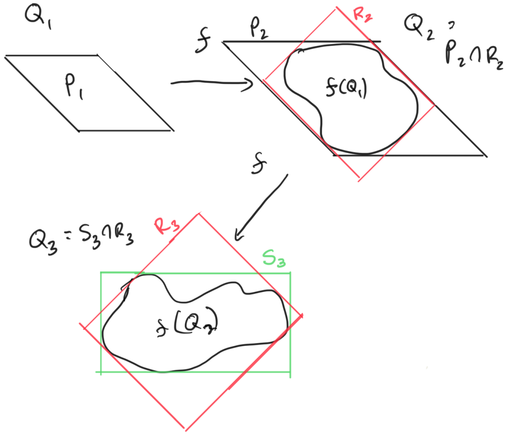


Figure: Instance of the template directions of parallelotope P having a template lifespan of two.

Contributions: Dynamic Algorithm

```
**Input:** Dynamics $f$, Initial Parallelotope $P_0$, Step Bound $S$
**Output:** Reachable Set Overapproximation $\overline{\Theta}_k$ at each step $k$
$Q_0 = \{P_0\}$
$\mathcal{T} = I$ // Init Template Directions
**for** $k \in [1, 2, \ldots, S]$ **do**
    $P_{supp}$ = GetSupportPoints $(Q_{k-1})$ (support points of $Q_{k-1}$)
    $P_{prop}$ = PropagatePointsOneStep $(P_{supp}, f)$ (image of support points)
    $A$ = ApproxLinearTrans $(P_{supp}, P_{prop})$
    $\mathcal{T} = \mathcal{T} \cdot A^{-1}$
    $\mathcal{T}_k^{\text{lin}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$
    $\mathcal{T}_k^{\text{pca}} = \{\mathtt{PCA}(P_{prop})\}$
    $\mathcal{T}_k = \mathcal{T}_k^{\text{lin}} \cup \mathcal{T}_k^{\text{pca}}$

    /* For lifespan $L > 0$, instead call TransformBundle with
       $\mathcal{T}_k \cup \mathcal{T}_{k-1} \cup \ldots \cup \mathcal{T}_{k-L}$ */
    $Q_k$ = TransformBundle $(f, Q_{k-1}, \mathcal{T}_k)$
    $\overline{\Theta}_k \leftarrow Q_k$
**end**
**return** $\overline{\Theta}_1 \ldots \overline{\Theta}_S$
```

Contributions: Kaa

1. Kaa is written in Python and relies on the *numpy* library for matrix computations, *sympy* library for all symbolic subsitution, and *scipy*, *matplotlib* for plotting the reachable sets and computing the volume for lower-dimensional systems.
2. The optimization procedure for finding the upper and lower bounds of template directions is performed through the *Kodiak* library. Finally, parallelization of the offset calculation procedures is implemented through the *multiprocessing* module.
3. Kodiak allows for more general dynamics beyond the polynomial ones considered in a previous tool, *Sapo*

Results: Benchmarks

1. For benchmarking, we select six non-linear models with polynomial dynamics.
2. Many of these models are also implemented in *Sapo* which explored reachability with **static** parallelotope bundles. In these cases, we directly compare the performance of our dynamic strategies with the Sapo's static parallelotopes defined by *diagonal* directions.
3. To provide meaningful comparisions, we set the number of dynamic parallelotopes to be equal to the number of static ones excluding the initial box.

Results: Benchmarks

Model	Dimension	Parameters	# steps	Δ
Vanderpol	2	-	70 steps	0.08
Jet Engine	2	-	100 steps	0.2
Neuron	2	-	200 steps	0.2
SIR	3	$\beta = 0.05$ $\gamma = 0.34$	150 steps	0.1
Coupled Vanderpol	4	-	40 steps	0.08
COVID	7	$\beta = 0.05$ $\gamma = 0.0$ $\eta = 0.02$	200 steps	0.08

Table: Benchmark models and truncated information

Results: COVID Supermodel

A modified SIR model accounting for the possibility of asymptomatic patients. Given by the following discretized seven-dimensional dynamics:

$$\begin{aligned}
S'_A &= S_A - (\beta S_A(A+I)) \cdot \Delta \\
S'_I &= S_I - (\beta S_I(A+I)) \cdot \Delta \\
A' &= A + (\beta S_I(A+I) - \gamma I) \cdot \Delta \\
I' &= I + (\beta S_I(A+I) - \gamma I) \cdot \Delta \\
R'_A &= R_A + (\gamma A) \cdot \Delta \\
R'_I &= R_I + (\gamma I) \cdot \Delta \\
D' &= D + (\eta I) \cdot \Delta
\end{aligned}$$

where the variables denote the fraction of a population of individuals designated as *Susceptible to Asymptomatic* (S_A), *Susceptible to Symptomatic* (S_I), *Asymptomatic (A)*, *Symptomatic (I)*, *Removed from Asymptomatic* (R_A), *Removed from Symptomatic* (R_I), and *Deceased (D)*.

Results: COVID Supermodel

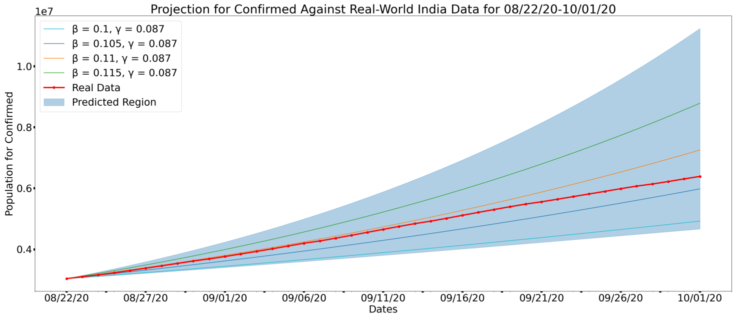


Figure: Plot of COVID Supermodel reachable set against real-world Indian COVID data.

Results: Comparision against Static Parallelotopes

1. For models previously defined in Sapo, we set the static parallelotopes to be exactly those found in Sapo.
2. If a model is not implemented in Sapo, we simply use the static parallelotopes defined in a model of equal dimension.
3. To address the unavailability of a four dimensional model implemented in Sapo, we sampled random subsets of five static parallelotopes defined with diagonal directions and chose the flowpipe with smallest volume.

Strategy	Total Volume
5 LinApp	0.227911
1 PCA, 4 LinApp	0.225917
2 PCA, 3 LinApp	0.195573
3 PCA, 2 LinApp	**0.188873**
4 PCA, 1 LinApp	1.227753
5 PCA	1.509897
5 Static Diagonal(Sapo)	2.863307

(a) Vanderpol

Strategy	Total Volume
5 LinApp	58199.62
1 PCA, 4 LinApp	31486.16
2 PCA, 3 LinApp	**5204.09**
3 PCA, 2 LinApp	6681.76
4 PCA, 1 LinApp	50505.10
5 PCA	84191.15
5 Static Diagonal (Sapo)	66182.18

(b) Jet Engine

Strategy	Total Volume
5 LinApp	154.078
1 PCA, 4 LinApp	136.089
2 PCA, 3 LinApp	73.420
3 PCA , 2 LinApp	**73.126**
4 PCA, 1 LinApp	76.33
5 PCA	83.896
5 Static Diagonal (Sapo)	202.406

(c) FitzHugh-Nagumo

Strategy	Total Volume
2 LinApp	**0.001423**
1 PCA, 1 LinApp	0.106546
2 PCA	0.117347
2 Static Diagonal (Sapo)	0.020894

(d) SIR

Strategy	Total Volume
5 LinApp	5.5171
1 PCA, 4 LinApp	**5.2536**
2 PCA, 3 LinApp	5.6670
3 PCA, 2 LinApp	5.5824
4 PCA, 1 LinApp	312.2108
5 PCA	388.0513
5 Static Diagonal (Best)	3023.4463

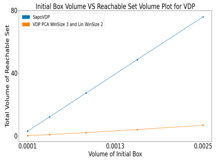
(e) Coupled Vanderpol

Strategy	Total Volume
3 LinApp	$2.95582227 * 10^{-10}$
1 PCA, 2 LinApp	$2.33007583 * 10^{-10}$
2 PCA, 1 LinApp	$4.02751770 * 10^{-9}$
3 PCA	$4.02749571 * 10^{-9}$
3 Static Diagonal (Sapo)	$4.02749571 * 10^{-9}$

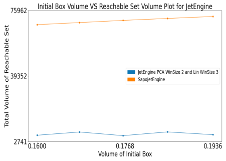
(f) COVID

Results: Performance under Increasing Initial Sets

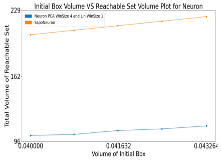
1. We vary the initial box dimensions to gradually increase the box's volume, then plot the total flowpipe volumes after running the benchmark on the static strategies and best-performing dynamic strategies.
2. Our dynamic strategies perform better than static ones in controlling the total flowpipe volume as the initial set becomes larger.
3. The performance of static parallelotopes tends to degrade rapidly as we increase the volume of the initial box



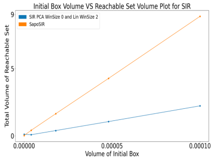
(a) Vanderpol



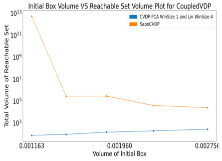
(b) Jet Engine



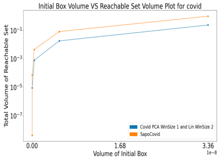
(c) Neuron



(d) SIR



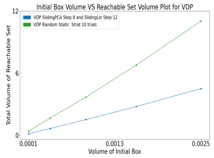
(e) Coupled Vanderpol



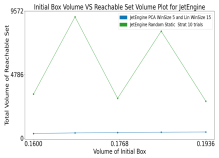
(f) COVID

Results: Performance against Random Static Bundles

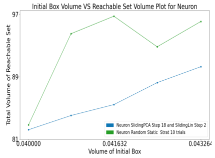
1. We compare our dynamic strategies against static random parallelotope bundles under initial sets with increasing volume.
2. We sample such parallelotopes in n dimensions by first sampling a set of n directions uniformly on the surface of the unit $(n-1)$-sphere, then defining our parallelotope using these sampled directions
3. We sample twenty of these parallelotopes for each trial, average the total flowpipe volumes, then plot this against the best-performing dynamic strategy.



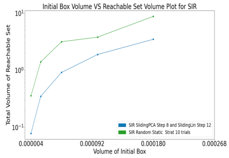
(a) Vanderpol



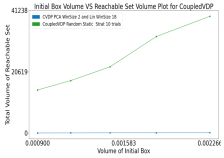
(b) Jet Engine



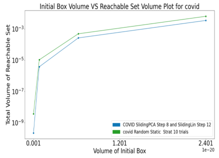
(c) Neuron



(d) SIR



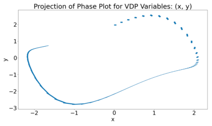
(e) Coupled Vanderpol



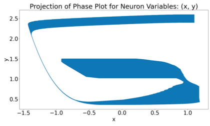
(f) COVID

Results: No Universal Ratio

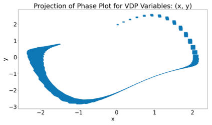
1. There seems to be no universal optimal ratio between the number of dynamic parallelotopes defined by PCA and Linear Approximation directions which perform well on all benchmarks.
2. In the Vanderpol model, using parallelotopes defined by Linear Approximation directions is more effective than those defined by PCA directions.
3. Neuron model shows the opposite trend where using parallelotopes defined by PCA directions is more effective than using those defined by Linear Approximation directions.



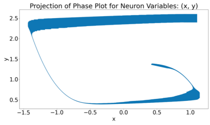
(a) 5 Lin



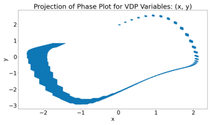
(b) 1 PCA 5 Lin



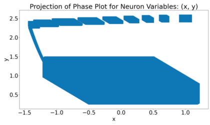
(c) 5 PCA



(d) 5 PCA 1 Lin



(e) Sapo



(f) Sapo

Future Directions

1. Are there other adaptive methods we can utilize to generate template directions dynamically?
2. What about more sophisticated linear approximation methods through Koopman linearization techniques?
3. The use of massively parallelized reachability techniques through HPC hardware is applicable here. Specifcally, can we leverage GPUs to parallelize the optimization procedure for larger parallelotope bundles?
4. Why do PCA directions perform better than Linear Approximation directions on certain systems?

Concluding Remarks

1. We investigated two techniques for generating templates dynamically: first using linear approximation of the dynamics, and second using PCA.
2. We also observed that both these techniques improve the accuracy of the reachable sets for different benchmarks.
3. We extend Kaa with Kodiak to handle more general dynamics. Systems with more general dynamics are considered in the extended version on arXiv.

Relevant Links

- **Github Repository for Kaa:** https://github.com/Tarheel-Formal-Methods/kaa-dynamic
- **Github Repository for Kodiak:** https://github.com/nasa/Kodiak
- **COVID Modeling Blog Post:** https://sigbed.org/2021/06/21/sidbed-blog-covid-formal-verification/