CS 6170: Computational Topology, Spring 2019
Lecture 21

Topological Data Analysis for Data Scientists

Dr. Bei Wang

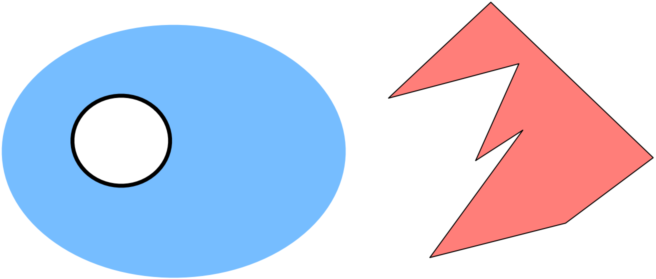
School of Computing
Scientific Computing and Imaging Institute (SCI)
University of Utah
www.sci.utah.edu/~beiwang
beiwang@sci.utah.edu

March 26, 2019

Simply Connectedness

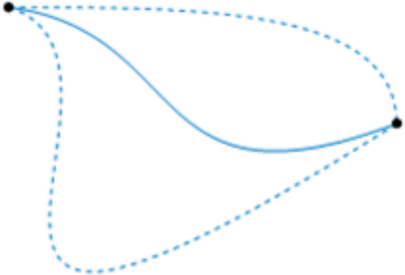
Simply connected topological space

- A topological space is simply connected if it is path-connected and every path between two points can be *continuously* transformed into any other such path while preserving the endpoints.



Homotopy

- A *homotopy* between two continuous functions $f, g : \mathbb{X} \to \mathbb{Y}$ is defined to be a *continuous* function $H : \mathbb{X} \times [0,1] \to \mathbb{Y}$ such that, if $x \in \mathbb{X}$ then $H(x,0) = f(x)$, and $H(x,1) = g(x)$.
- A *homotopy* between two continuous functions $f, g : \mathbb{X} \to \mathbb{Y}$ is a family of continuous functions $h_t : \mathbb{X} \to \mathbb{Y}$ for $t \in [0,1]$ such that $h_0 = f$ and $h_1 = g$, and the map $(x,t) \mapsto h_t(x)$ is *continuous* from $\mathbb{X} \times [0,1]$ to $\mathbb{Y}$.
- The two versions coincide by setting $h_t(x) = H(x,t)$.



https://en.wikipedia.org/wiki/Homotopy

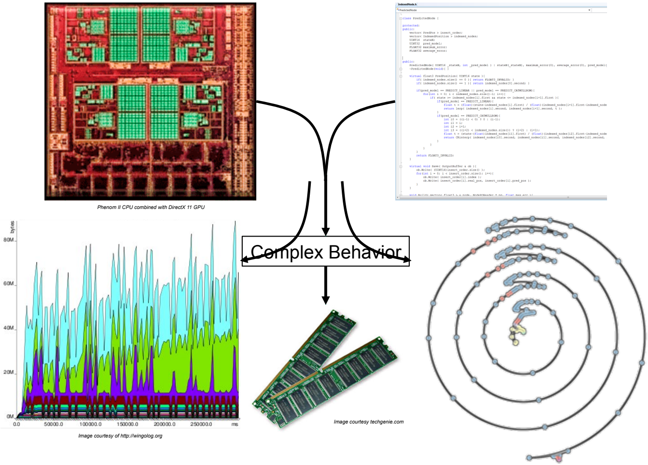
Simply connected topological space

- A topological space $\mathbb{X}$ is *simply connected* if it is path-connected and any loop in $\mathbb{X}$ defined by $f : \mathbb{S}^1 \to \mathbb{X}$ can be contracted to a point.
- $\mathbb{X}$ is *simply connected* if and only if it is *path-connected*, and whenever $p : [0,1] \to \mathbb{X}$ and $q : [0,1] \to \mathbb{X}$ are two paths (i.e.: continuous maps) with the same start and endpoint $p(0) = q(0)$ and $p(1) = q(1)$, then p can be continuously deformed into q while keeping both endpoints fixed.
- Explicitly, there exists a continuous homotopy $F : [0,1] \times [0,1] \to \mathbb{X}$ such that $F(x,0) = p(x)$ and $F(x,1) = q(x)$.
- $\mathbb{X}$ is *path-connected* if there is a path joining any two points in $\mathbb{X}$.

TDA: Relation to Time Series Analysis
Choudhury et al. (2012)

A story from software visualization

- Detecting circular structures in memory reference traces
- Takens embedding
- Capture recurrent nature of the program



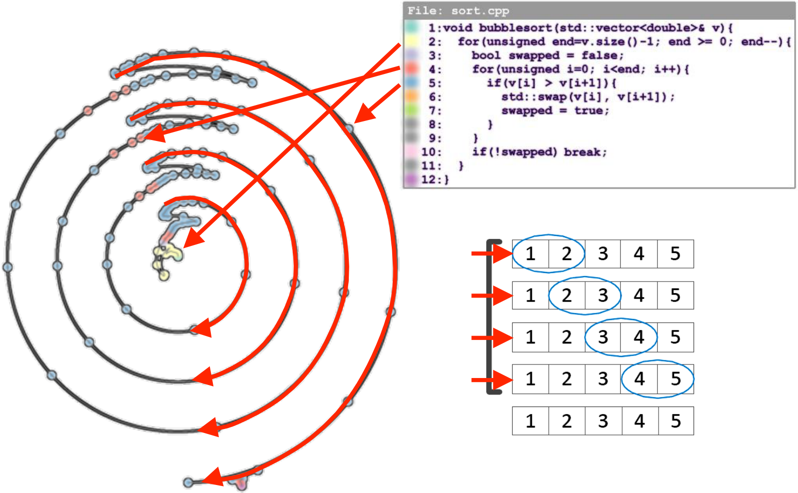
Phenom II CPU combined with DirectX 11 GPU

Image courtesy of http://wingolog.org

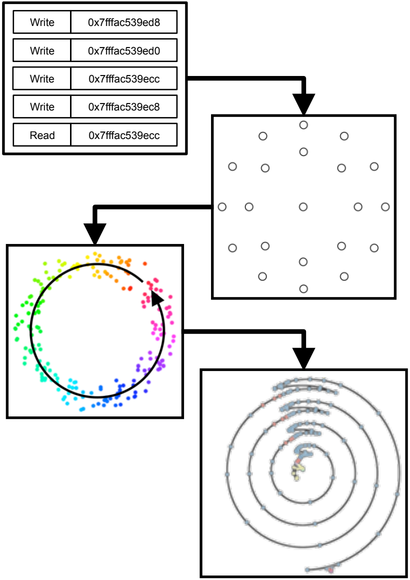
Image courtesy techgenie.com

Example

Visualize circular structures in memory access patterns



System pipeline



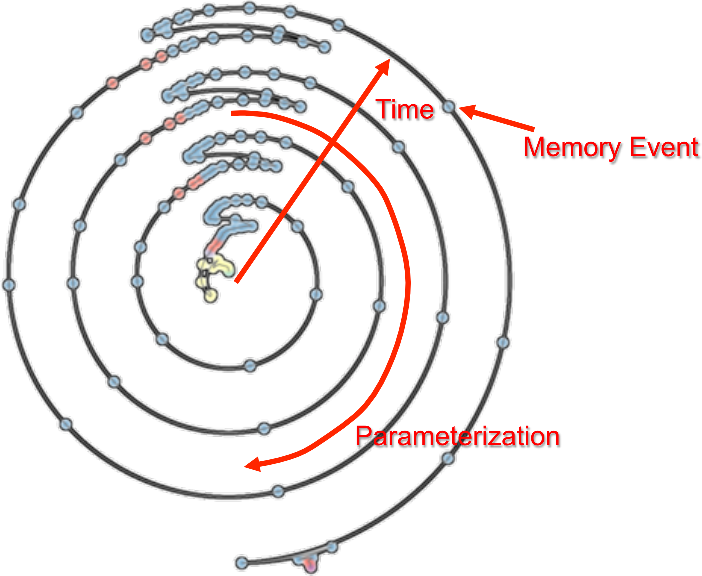
Capture a memory reference trace

File: sort.cpp

```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```

Write	0x7fffac539ed8
Write	0x7fffac539ed0
Write	0x7fffac539ecc
Write	0x7fffac539ec8
Read	0x7fffac539ecc
Read	0x7fffac539ec8
Write	0x7fffac539eb8
Write	0x7fffac539eb0

Visualization



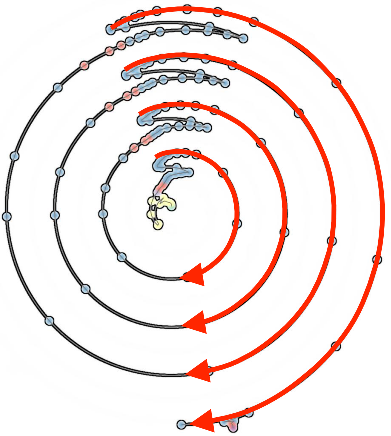
Data dependent structure

$\#$ loops $= \#$ comparisons

File: sort.cpp

```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

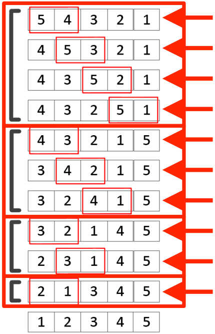


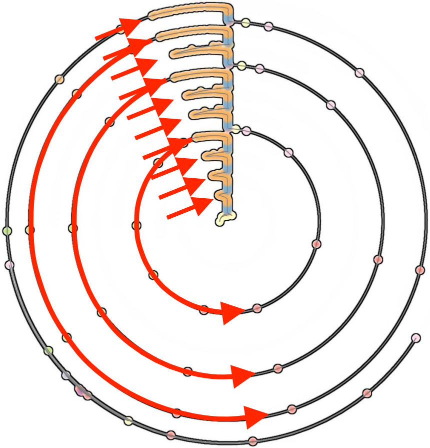
Data dependent structure

$\#$ teeth $= \#$ swaps

File: sort.cpp

```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```





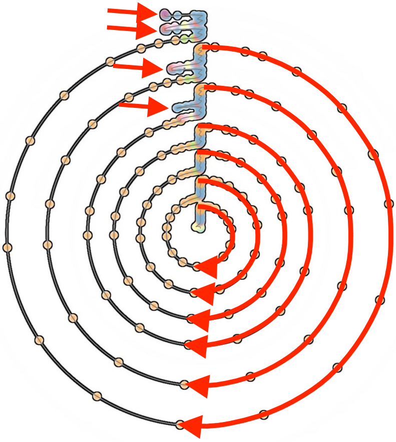
Data dependent structure

$\#$ teeth $= \#$ comparisons

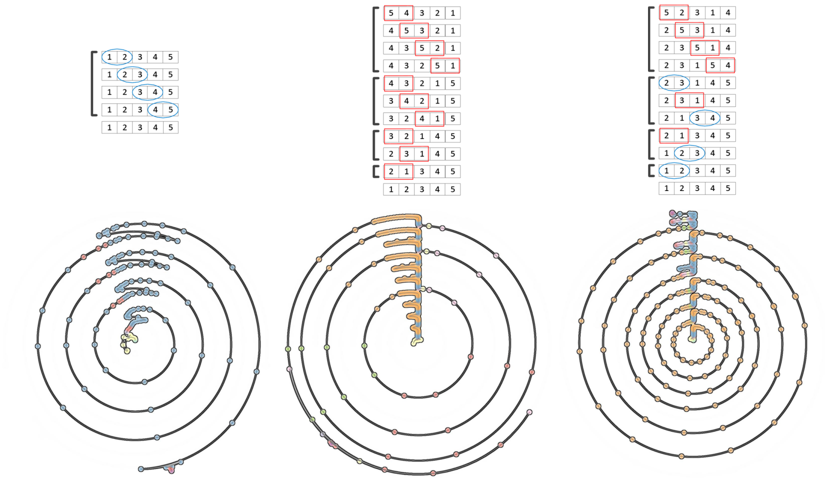
$\#$ loops $= \#$ comparisons and swaps

```
File: sort.cpp
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```

5	2	3	1	4
2	5	3	1	4
2	3	5	1	4
2	3	1	5	4
2	3	1	4	5
2	3	1	4	5
2	1	3	4	5
2	1	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5



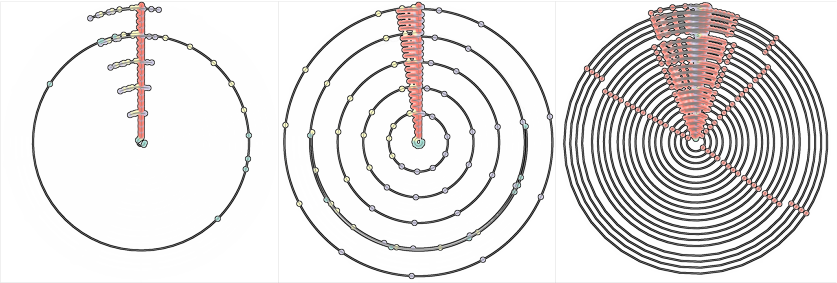
Data dependent structure



Algorithm dependent structure

File: matmult.cpp

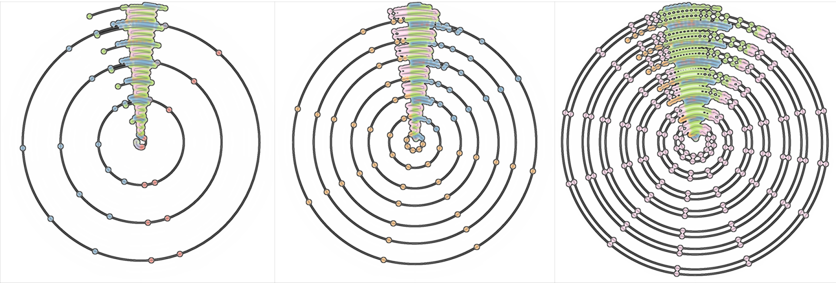
```
unsigned int i, j, k;
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
      linC[i*N + j] += linA[i*N + k] * linB[k*N + j];
```



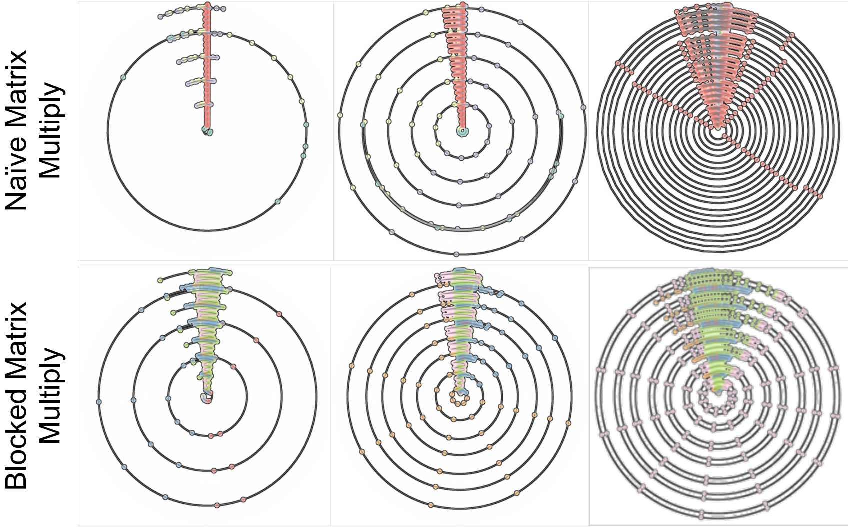
Algorithm dependent structure

File: blocked-matmult.cpp

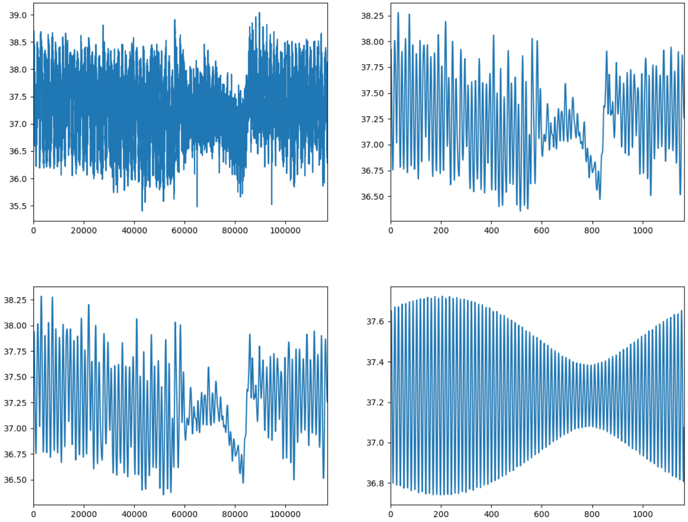
```
unsigned int i, j, k, j0, k0;
for (k0 = 0; k0 < N; k0 += b)
  for (j0 = 0; j0 < N; j0 += b)
    for (i = 0; i < N; i++)
      for (k = k0; k < min(k0 + b, N); k++) {
        r = linA[i*N + k];
        for (j = j0; j < min(j0 + b, N); j++)
          linC[i*N + j] += r*linB[k*N + j];
      }
```



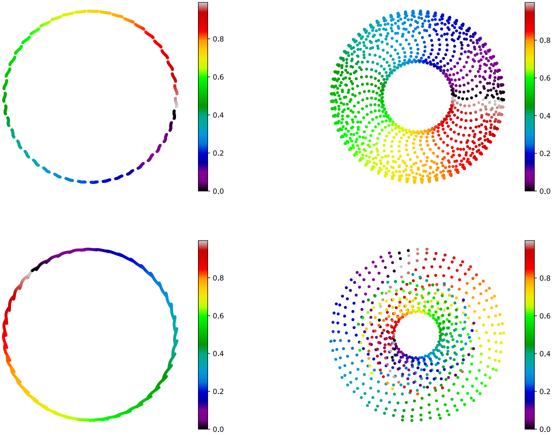
Algorithm dependent structure



A story from mice pregnancy detection

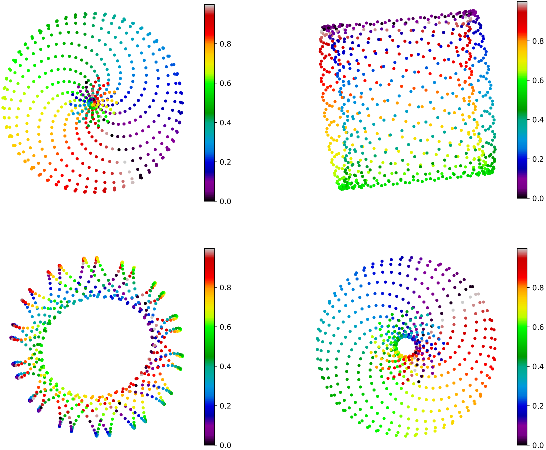


A story from mice pregnancy detection



Which mice are pregnant? Joint work with Benjamin Smarr.
Data from Smarr et al. (2016).

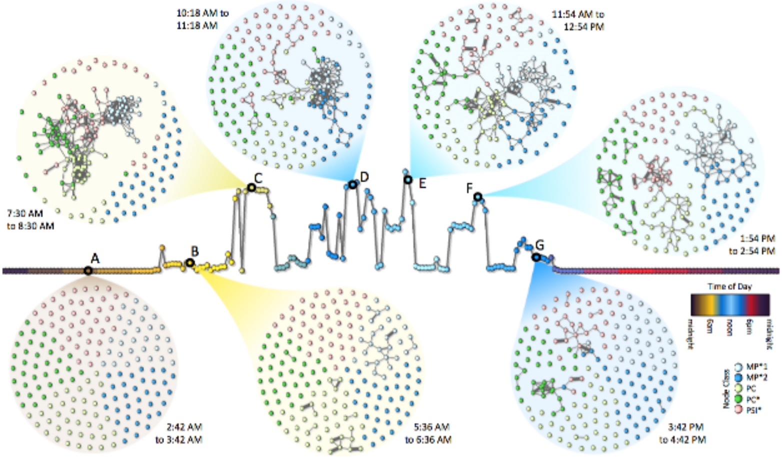
Jet lagged mice



Which jet lagged mice are pregnant?

Persistent homology of time-varying networks

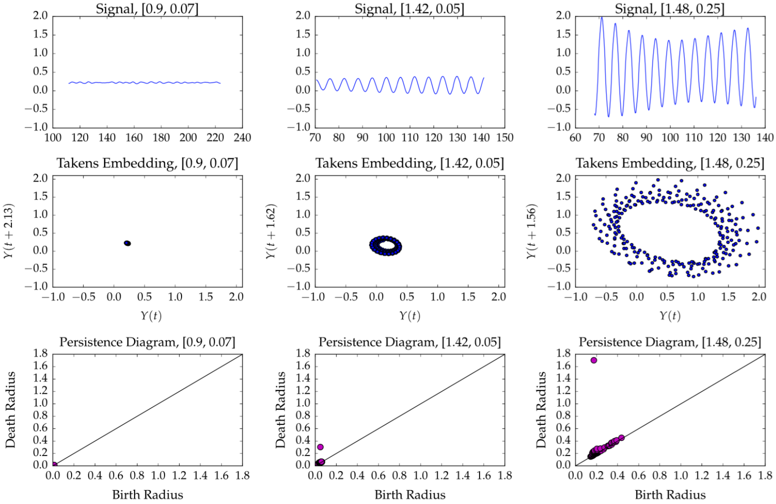
Visual detection of structural changes in time-varying graphs using persistent homology.



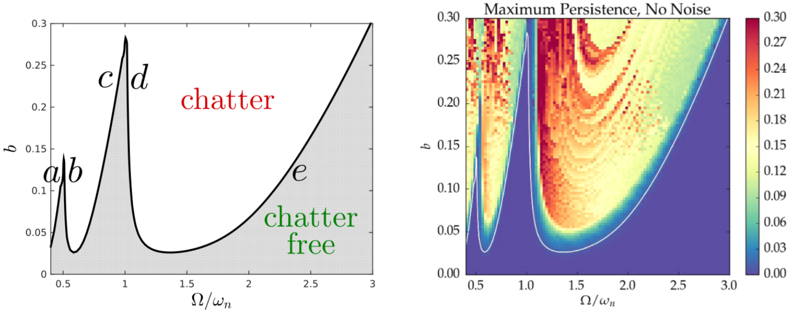
Hajij et al. (2018)

Takens embedding

Maximal persistence: Khasawneh and Munch (2016)



Chatter detection



Khasawneh and Munch (2016)

Reconstruct dynamics using witness complexes

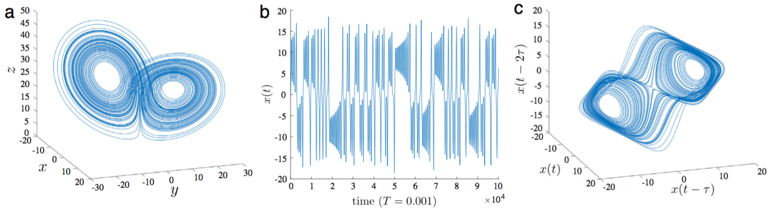
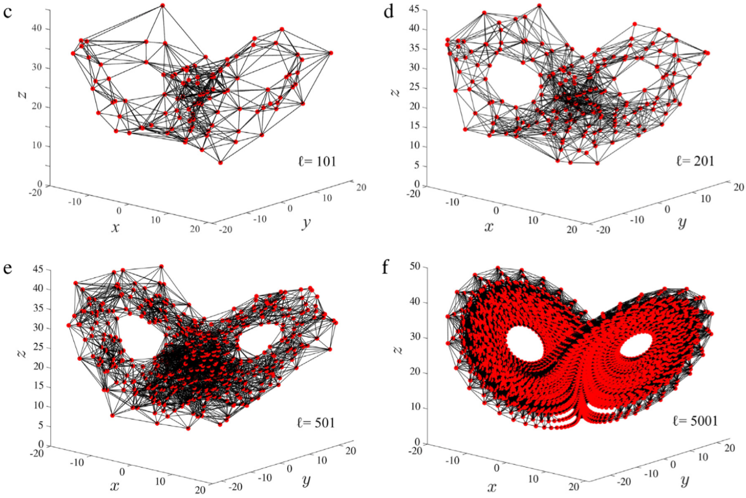


Fig. 1. Classic Lorenz attractor ($r = 28$, $b = 8/3$, $\sigma = 10$): (a) A 10^5-point trajectory in $\mathbb{R}^3$ generated using fourth-order Runge–Kutta with a time step of $T = 0.001$. (b) A time-series trace of the x coordinate of that trajectory. (c) A 3D projection of a delay-coordinate embedding with dimension $m = 5$ and delay $\tau = 174T$, following (1).

Garland et al. (2016)

Reconstruct dynamics using witness complexes



Garland et al. (2016)

References I

Choudhury, A. I., Wang, B., Rosen, P., and Pascucci, V. (2012). Topological analysis and visualization of cyclical behavior in memory reference traces. *IEEE Pacific Visualization Symposium (PacificVis)*, pages 9–16.

Garland, J., Bradley, E., and Meiss, J. D. (2016). Exploring the topology of dynamical reconstructions. *Physica D: Nonlinear Phenomena*, 334(1):49–59.

Hajij, M., Wang, B., Scheidegger, C., and Rosen, P. (2018). Visual detection of structural changes in time-varying graphs using persistent homology. *IEEE Pacific Visualization Symposium (PacificVis)*.

Khasawneh, F. A. and Munch, E. (2016). Chatter detection in turning using persistent homology. *Mechanical Systems and Signal Processing*, 70-71:527–541.

Smarr, B. L., Zucker, I., and Kriegsfeld, L. J. (2016). Detection of successful and unsuccessful pregnancies in mice within hours of pairing through frequency analysis of high temporal resolution core body temperature data. *PLoS One*.