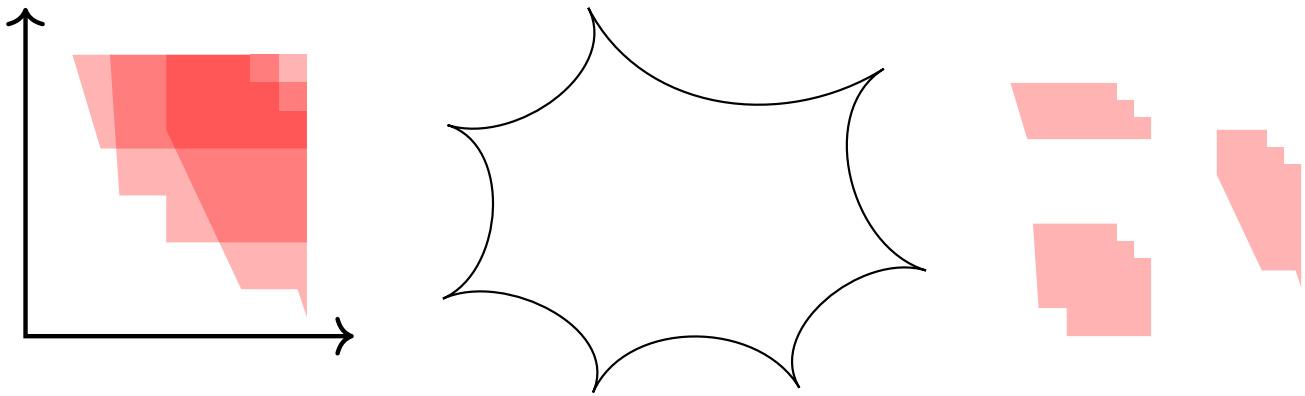
UNIVERSITY AT ALBANY
State University of New York

Computing approximate decompositions of multiparameter modules

Håvard Bakke Bjerkevik, Jan Jendrysiak and Fabian Lenzen

University at Albany, SUNY

May 19, 2026



Joining forces

- Stabilizing Decomposition of Multiparameter Persistence Modules (B.)

Joining forces

- Stabilizing Decomposition of Multiparameter Persistence Modules (B.)
- Decomposing Multiparameter Persistence Modules/AIDA (Dey, Jendrysiak, Kerber)

Joining forces

- Stabilizing Decomposition of Multiparameter Persistence Modules (B.)
- Decomposing Multiparameter Persistence Modules/AIDA (Dey, Jendrysiak, Kerber)
- Current project (B., Jan Jendrysiak, Fabian Lenzen)

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,
- good for computation.

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,
- good for computation.

Advantages of multiparameter persistence:

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,
- good for computation.

Advantages of multiparameter persistence:

- Can deal with outliers in a more principled way,

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,
- good for computation.

Advantages of multiparameter persistence:

- Can deal with outliers in a more principled way,
- richer, more flexible.

The persistence barcode

In one-parameter persistence, we can decompose modules algebraically to get a barcode/persistence diagram.

Advantages of the barcode:

- Easy to visualize,
- interpretable,
- good for computation.

Advantages of multiparameter persistence:

- Can deal with outliers in a more principled way,
- richer, more flexible.

We would like a multiparameter barcode!

Failure of decomposition

Theorem (Bauer, Scoccola (2022))

For any $\epsilon > 0$ and finitely presented two-parameter module M, there is an indecomposable module M' that is ϵ-interleaved with M.

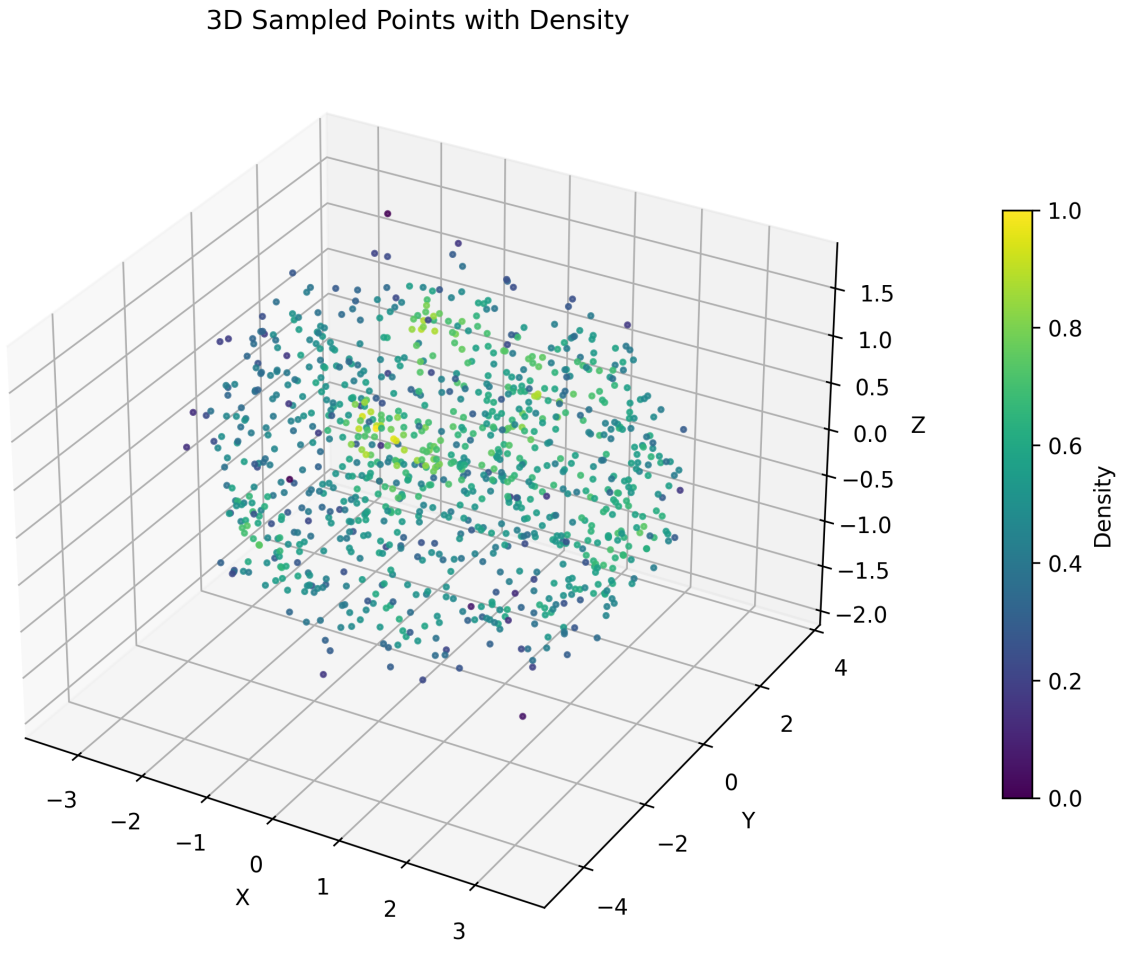
Failure of decomposition

Theorem (Bauer, Scoccola (2022))

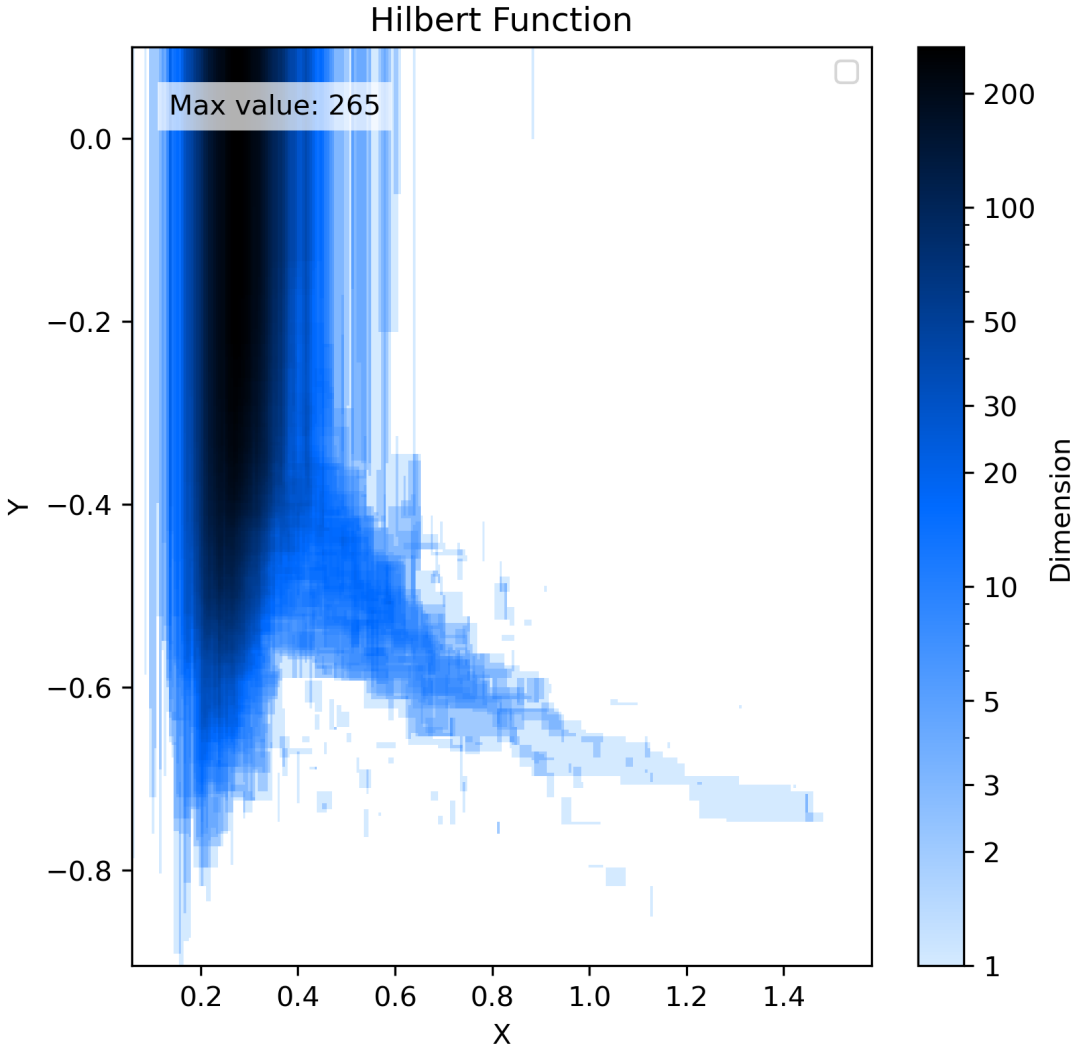
For any $\epsilon > 0$ and finitely presented two-parameter module M, there is an indecomposable module M' that is ϵ-interleaved with M.

Do these problems show up in practice?

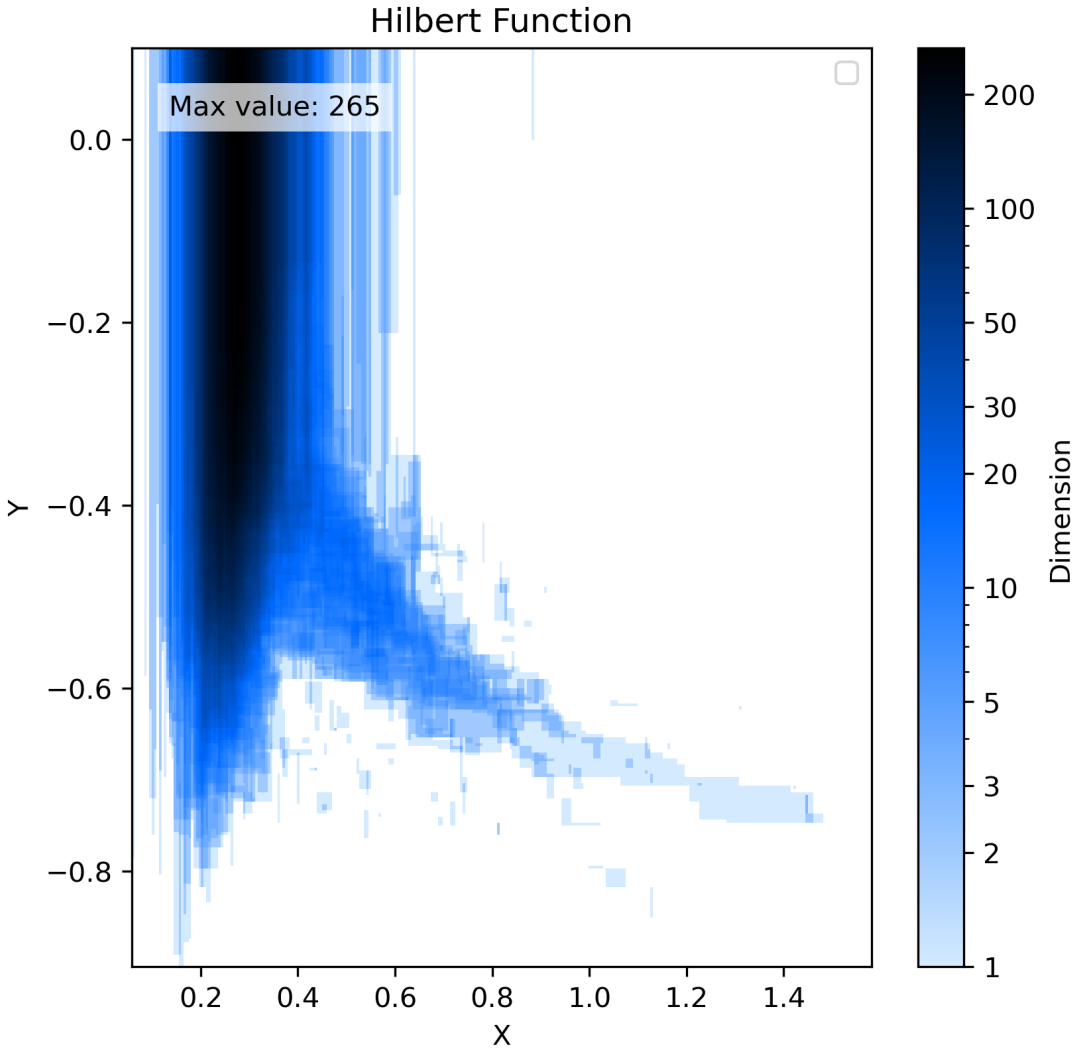
Failure of decomposition



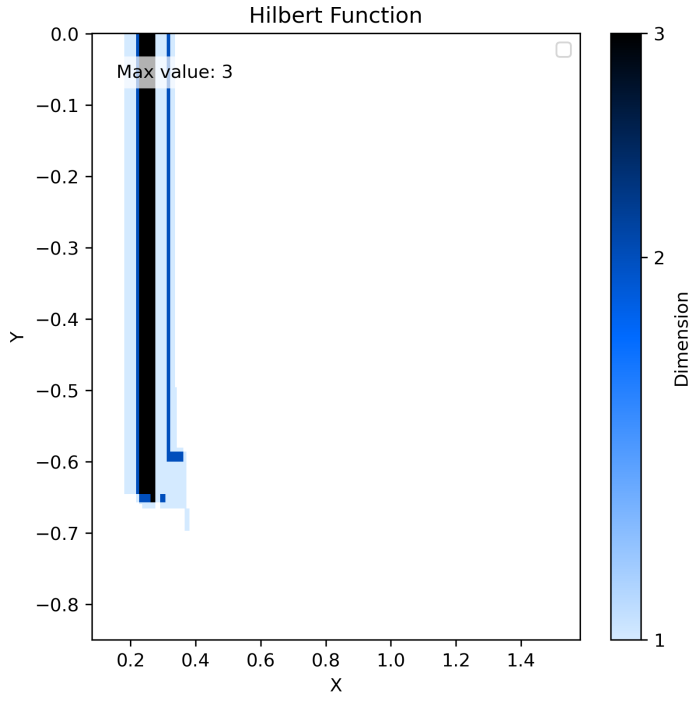
Failure of decomposition

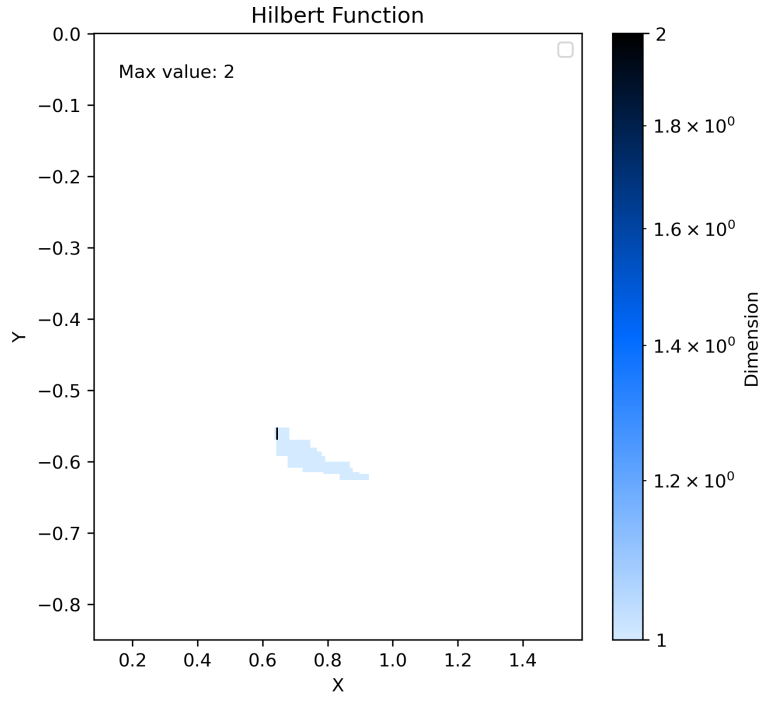


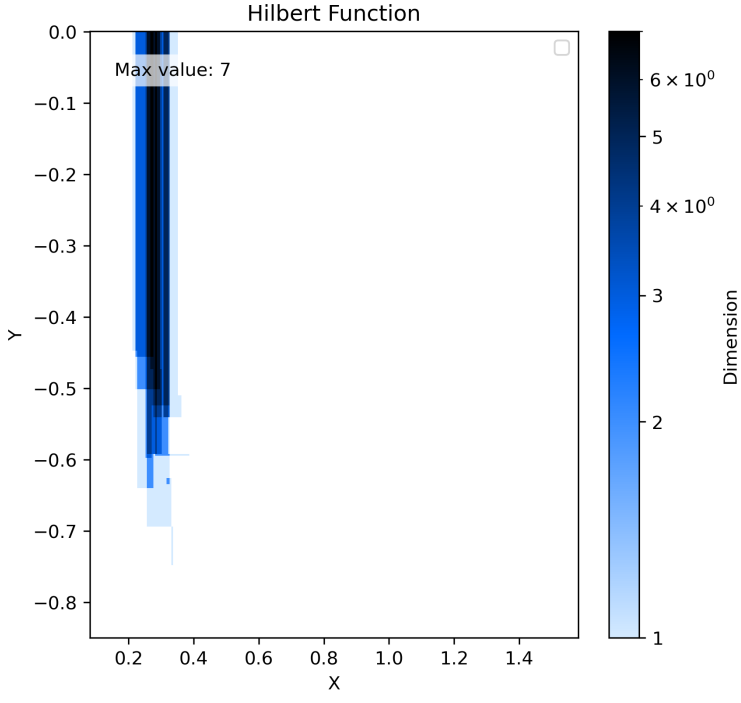
Failure of decomposition

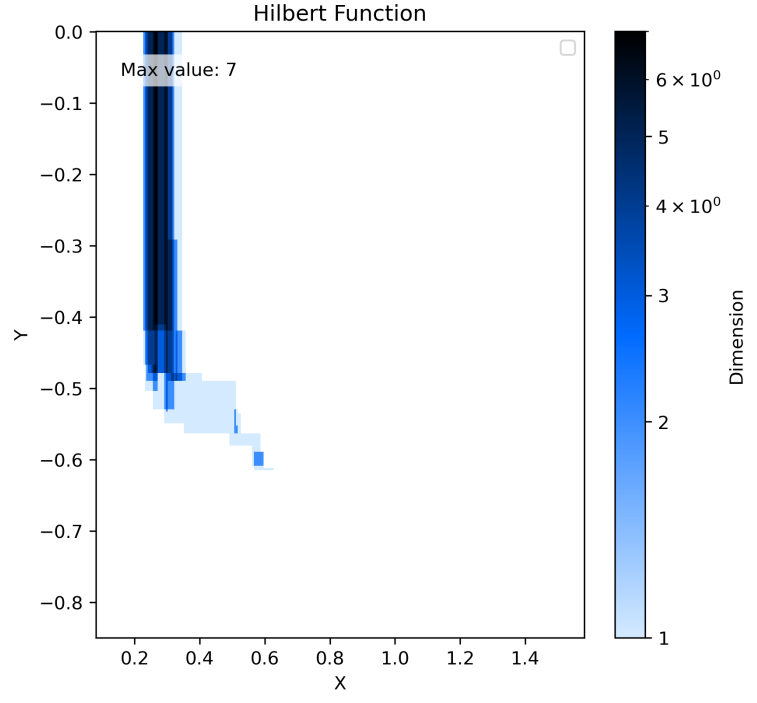


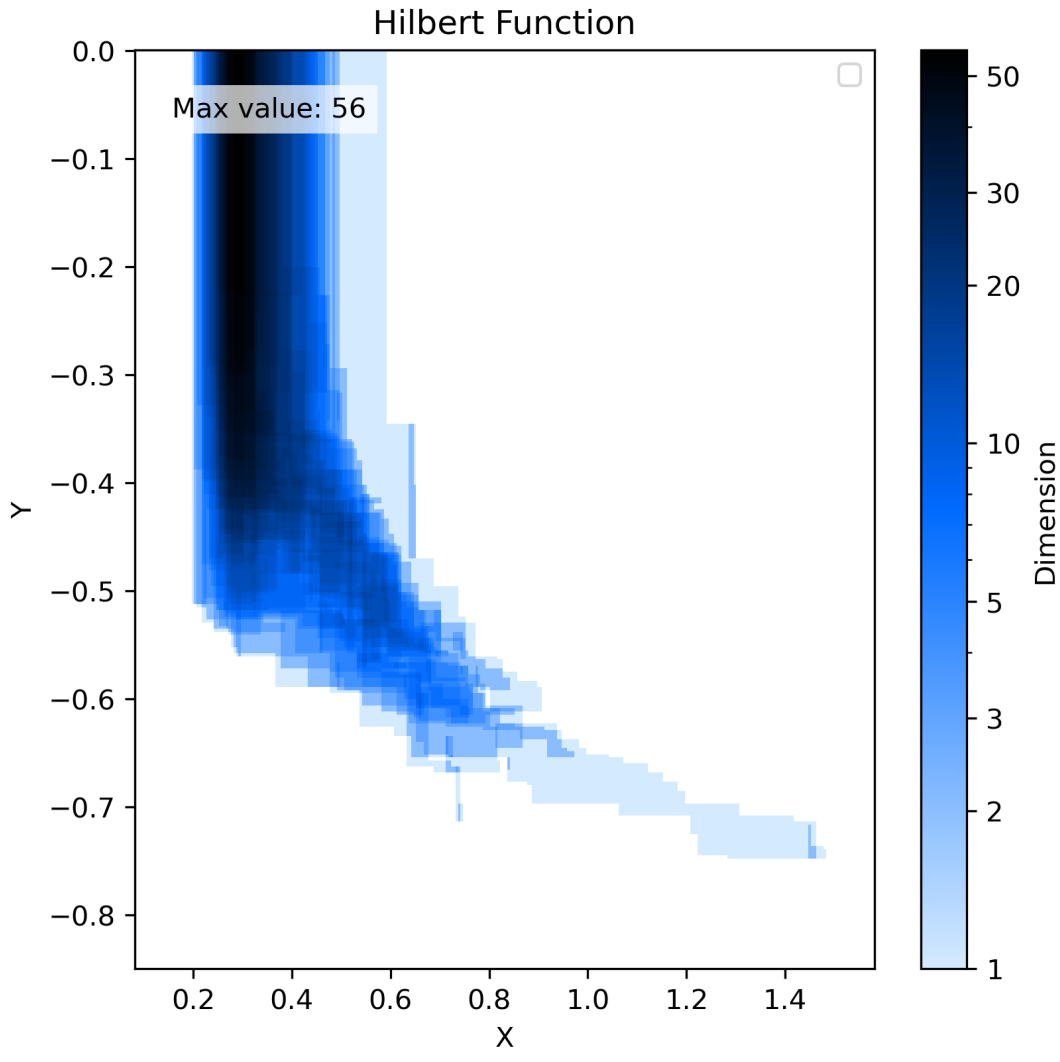
2742 components.

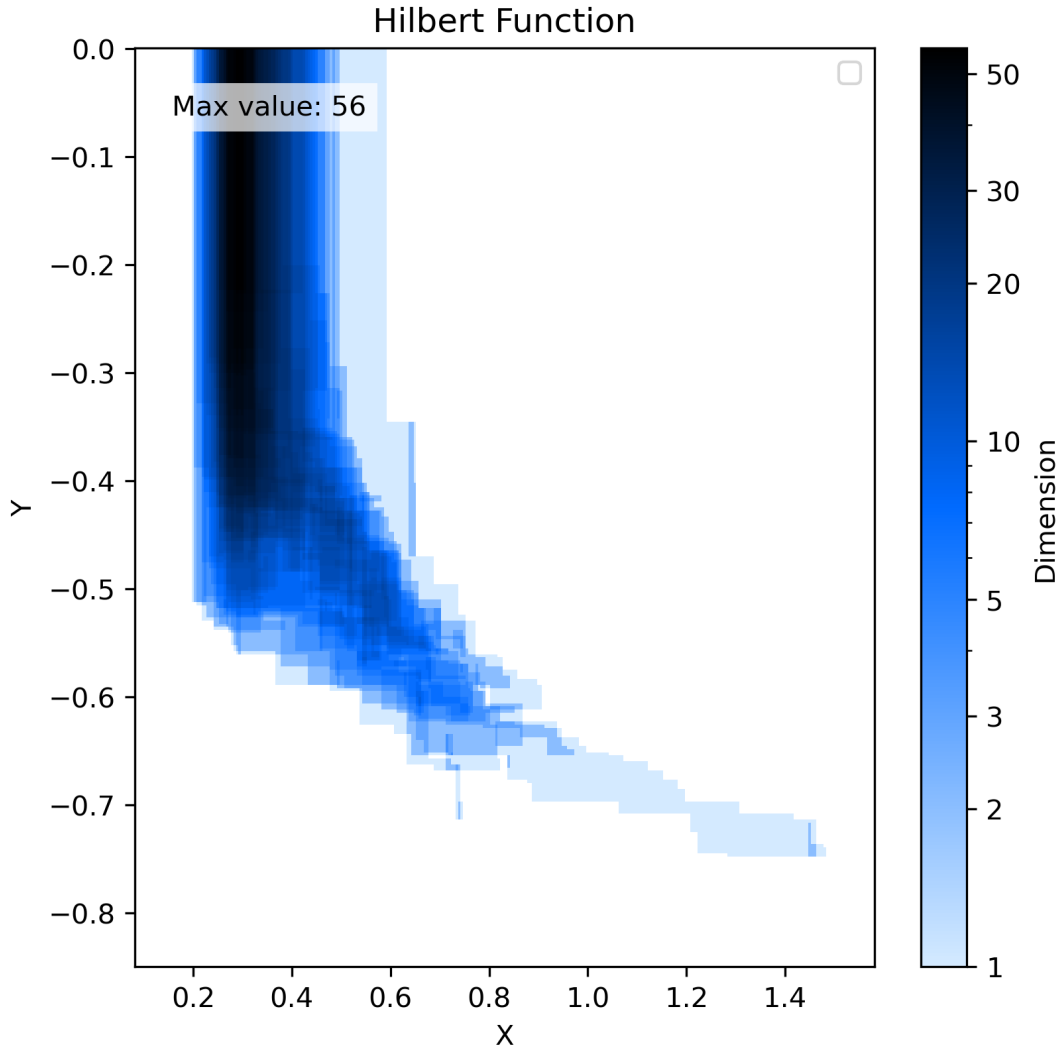

Hilbert Function
Max value: 3
Dimension
X
Y


Hilbert Function
Max value: 2
Dimension
X
Y


Hilbert Function
Max value: 7
Dimension
X
Y


Hilbert Function
Max value: 7
Dimension
X
Y


Hilbert Function
Max value: 56
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
Y
0.2
0.4
0.6
0.8
1.0
1.2
1.4
X
50
30
20
10
5
3
2
1
Dimension



There is a “giant component”.

Pruning

Idea: Even if a module is indecomposable, it might be *almost* decomposable.

Pruning

Idea: Even if a module is indecomposable, it might be *almost* decomposable.

Q: How do we decompose an almost-decomposable module?

Pruning

Idea: Even if a module is indecomposable, it might be *almost* decomposable.

Q: How do we decompose an almost-decomposable module?

A: We prune it!

Pruning

Idea: Even if a module is indecomposable, it might be *almost* decomposable.

Q: How do we decompose an almost-decomposable module?

A: We prune it!

Definition (B. (2024))

Let M be a module. Let I be the largest submodule of M such that for any morphism $f\colon M \to M(2\epsilon)$, $f(I) \subseteq M_{0\to 2\epsilon}(I)$. Let K be the smallest submodule of I such that for any morphism $f\colon M \to M(2\epsilon)$, $I^{-1}_{0\to 2\epsilon}(f(K)) \subseteq K$. We define the ϵ**-pruning pair** of M to be (I, K), and the ϵ**-pruning** of M to be $\mathrm{Pru}_\epsilon(M) := (I/K)(-\epsilon)$.

Pruning

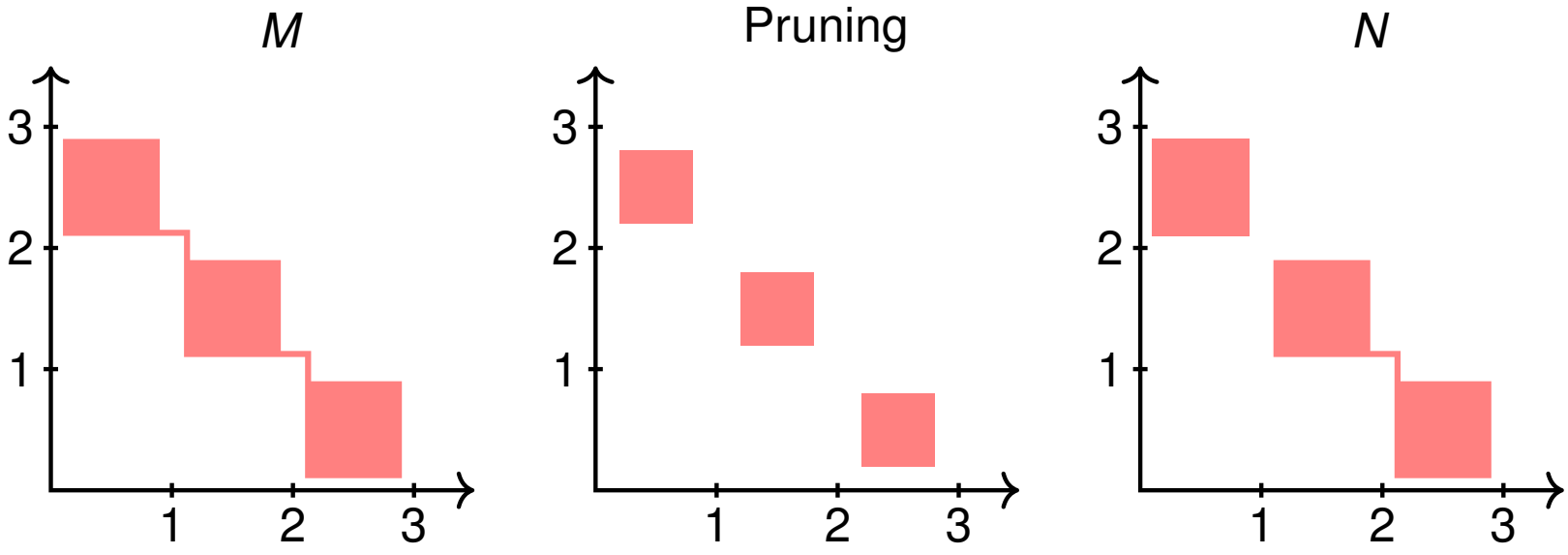
Theorem (B. (2024))

Let $\epsilon \geq 0$, and let M and N be ϵ-interleaved pfd modules, and let $r = \operatorname{supdim} M < \infty$. Then $\operatorname{Pru}_\epsilon(M)$ is a $2r\epsilon$-refinement of N.

Pruning

Theorem (B. (2024))

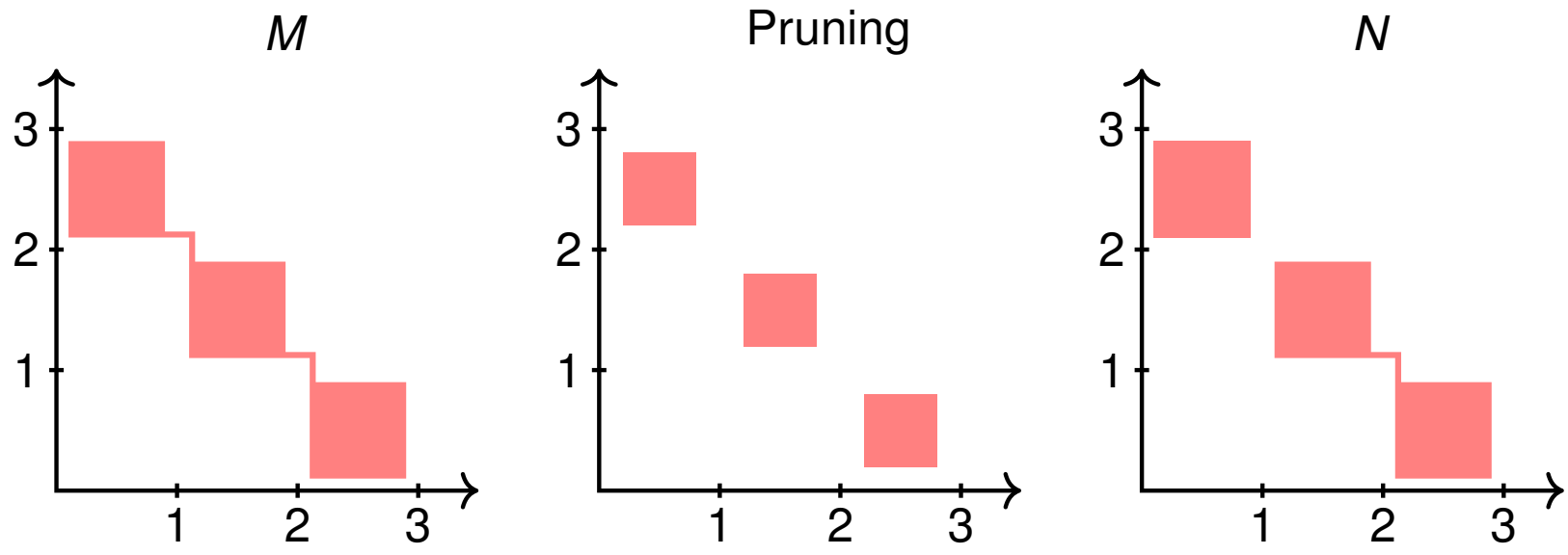
Let $\epsilon \geq 0$, and let M and N be ϵ-interleaved pfd modules, and let $r = \operatorname{supdim} M < \infty$. Then $\operatorname{Pru}_\epsilon(M)$ is a $2r\epsilon$-refinement of N.



Pruning

Theorem (B. (2024))

Let $\epsilon \geq 0$, and let M and N be ϵ-interleaved pfd modules, and let $r = \operatorname{supdim} M < \infty$. Then $\operatorname{Pru}_\epsilon(M)$ is a $2r\epsilon$-refinement of N.

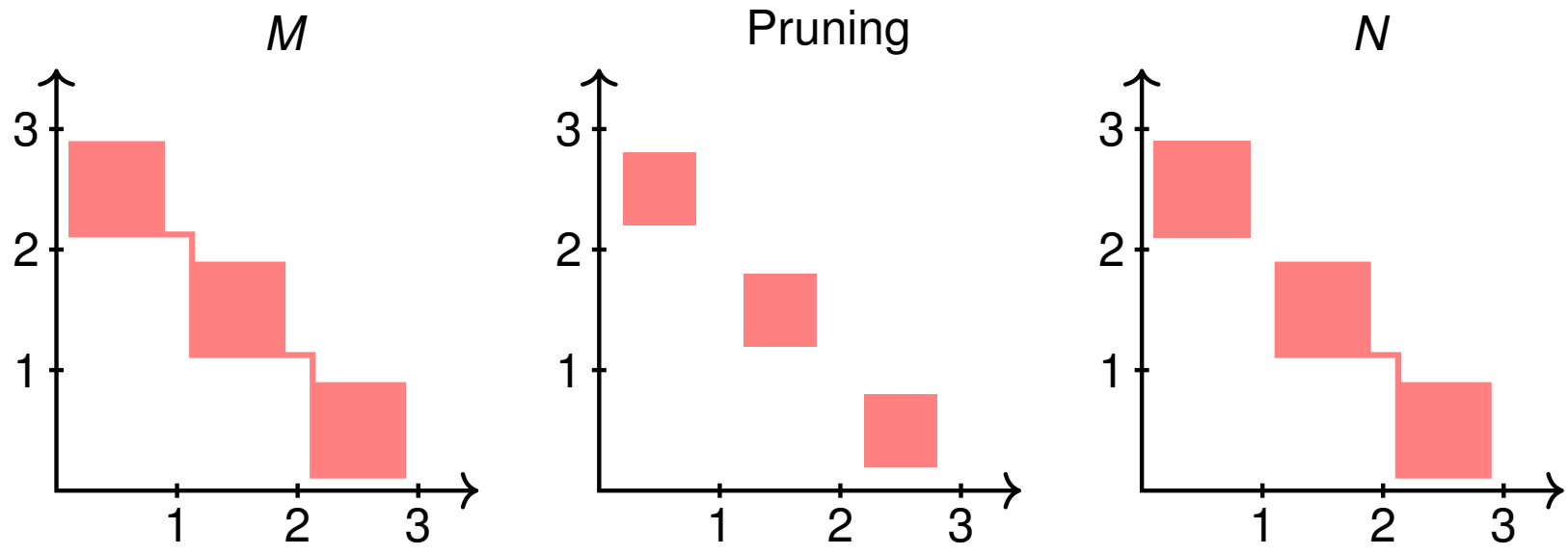


The ϵ-pruning of M detects all features appearing ϵ-close to M!

Pruning

Theorem (B. (2024))

Let $\epsilon \geq 0$, and let M and N be ϵ-interleaved pfd modules, and let $r = \operatorname{supdim} M < \infty$. Then $\operatorname{Pru}_\epsilon(M)$ is a $2r\epsilon$-refinement of N.



The ϵ-pruning of M detects all features appearing ϵ-close to M!

Drawback: The pruning shrinks M. In the worst case, it shrinks it by a lot.

Let's compute the pruning! But how?

Definition

Let M be a module. Let I be the largest submodule of M such that for any morphism $f\colon M \to M(2\epsilon)$, $f(I) \subseteq M_{0\to 2\epsilon}(I)$. Let K be the smallest submodule of I such that for any morphism $f\colon M \to M(2\epsilon)$, $I^{-1}_{0\to 2\epsilon}(f(K)) \subseteq K$. We define the ϵ**-pruning pair** of M to be (I, K), and the ϵ**-pruning** of M to be $\mathrm{Pru}_\epsilon(M) := (I/K)(-\epsilon)$.

A more practical description

Lemma

Let (I, K) be the ϵ-pruning pair of M. Let $\Gamma = \mathrm{Hom}(M, M(2\epsilon))$. Then $I = \bigcap_{i=0}^{\infty} I_i$, where we define $I_0 = M$, and for all $i \geq 1$,

- $I_i = \bigcap_{f \in \Gamma} f^{-1}(M_{0 \to 2\epsilon}(I_{i-1}))$.

$K = \bigcup_{i=0}^{\infty} K_i$, where we define $K_0 = 0$, and for all $i \geq 1$,

- $K_i = \sum_{f \in \Gamma} I^{-1}_{0 \to 2\epsilon}(f(K_{i-1}))$.

A more practical description

Lemma

Let (I, K) be the ϵ-pruning pair of M. Let $\Gamma = \mathrm{Hom}(M, M(2\epsilon))$. Then $I = \bigcap_{i=0}^{\infty} I_i$, where we define $I_0 = M$, and for all $i \geq 1$,

- $I_i = \bigcap_{f \in \Gamma} f^{-1}(M_{0 \to 2\epsilon}(I_{i-1}))$.

$K = \bigcup_{i=0}^{\infty} K_i$, where we define $K_0 = 0$, and for all $i \geq 1$,

- $K_i = \sum_{f \in \Gamma} I^{-1}_{0 \to 2\epsilon}(f(K_{i-1}))$.

Observation:

A more practical description

Lemma

Let (I, K) be the ϵ-pruning pair of M. Let $\Gamma = \mathrm{Hom}(M, M(2\epsilon))$. Then $I = \bigcap_{i=0}^{\infty} I_i$, where we define $I_0 = M$, and for all $i \geq 1$,

- $I_i = \bigcap_{f \in \Gamma} f^{-1}(M_{0 \to 2\epsilon}(I_{i-1}))$.

$K = \bigcup_{i=0}^{\infty} K_i$, where we define $K_0 = 0$, and for all $i \geq 1$,

- $K_i = \sum_{f \in \Gamma} I^{-1}_{0 \to 2\epsilon}(f(K_{i-1}))$.

Observation: We only need a basis of $\mathrm{Hom}(M, M(2\epsilon))$.

A more practical description

Lemma

Let (I, K) be the ϵ-pruning pair of M. Let $\Gamma = \mathrm{Hom}(M, M(2\epsilon))$. Then $I = \bigcap_{i=0}^{\infty} I_i$, where we define $I_0 = M$, and for all $i \geq 1$,

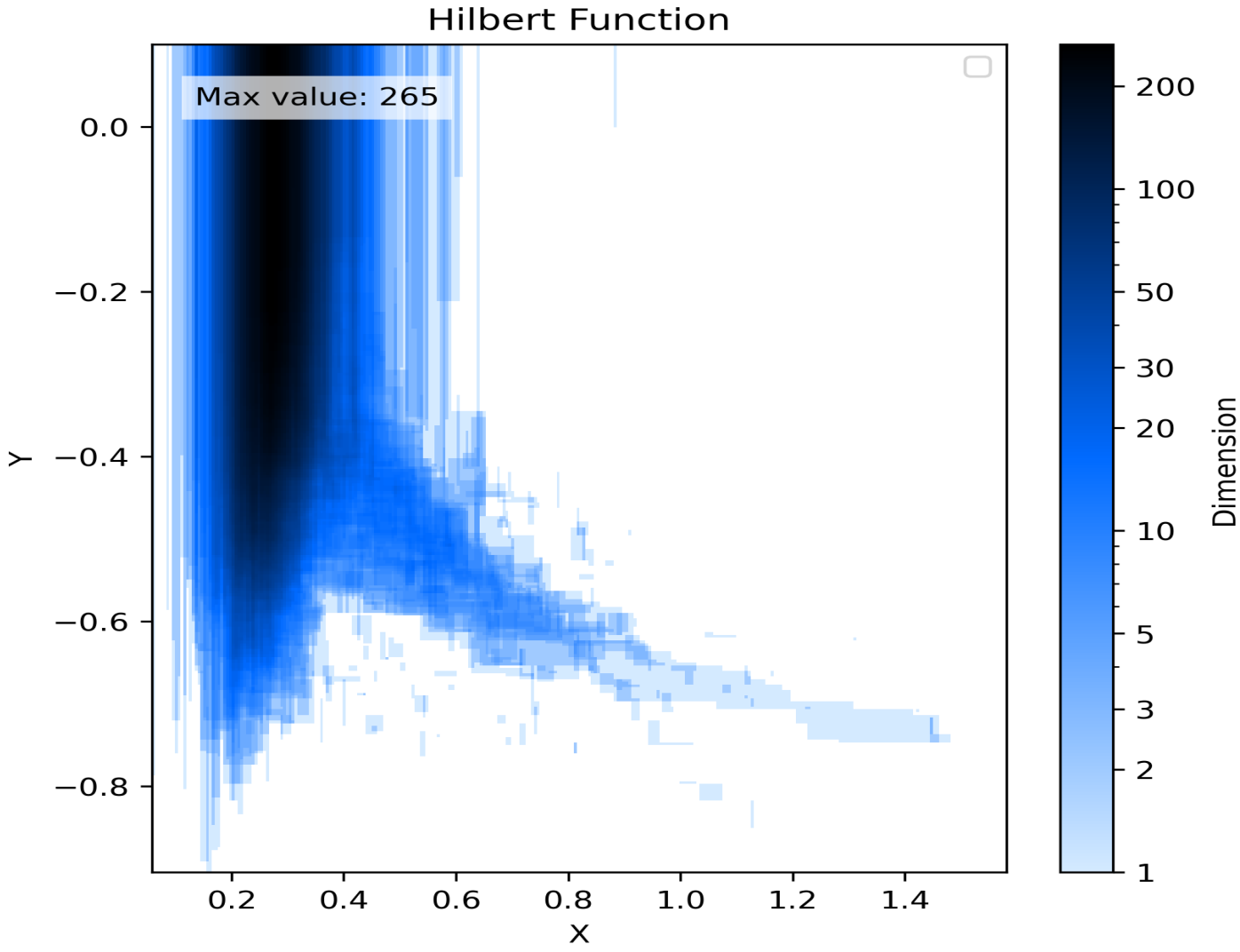
- $I_i = \bigcap_{f \in \Gamma} f^{-1}(M_{0 \to 2\epsilon}(I_{i-1}))$.

$K = \bigcup_{i=0}^{\infty} K_i$, where we define $K_0 = 0$, and for all $i \geq 1$,

- $K_i = \sum_{f \in \Gamma} I^{-1}_{0 \to 2\epsilon}(f(K_{i-1}))$.

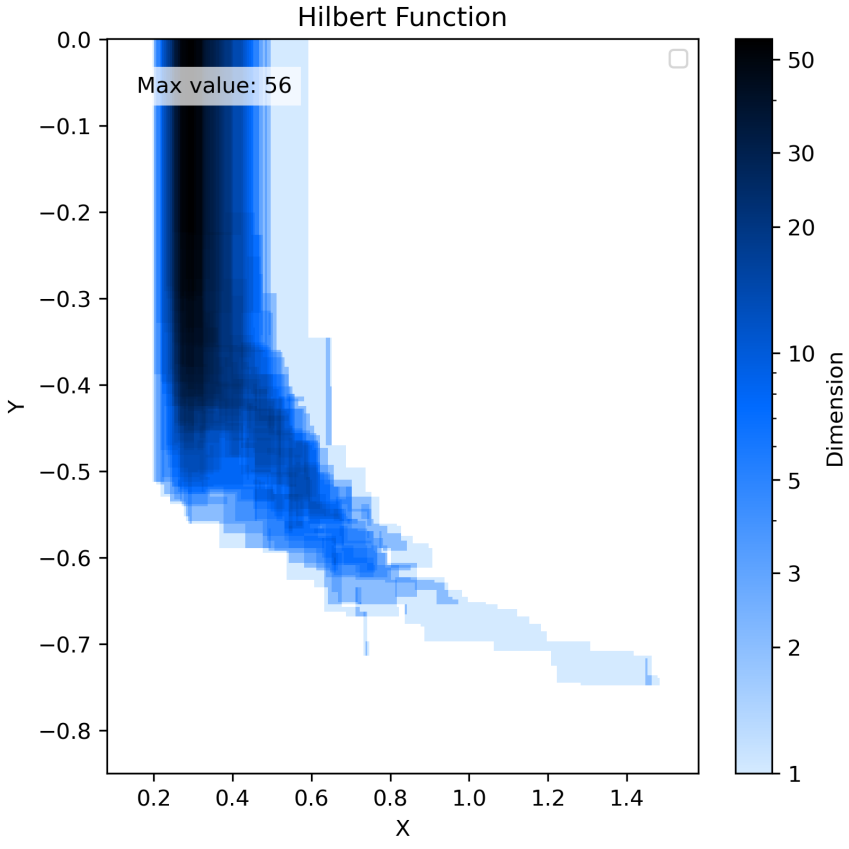
Observation: We only need a basis of $\mathrm{Hom}(M, M(2\epsilon))$.

Plus further simplifications. Some implemented, some not.


Hilbert Function
Max value: 265
0.0
−0.2
−0.4
−0.6
−0.8
Y
0.2
0.4
0.6
0.8
1.0
1.2
1.4
X
200
100
50
30
20
10
5
3
2
1
Dimension

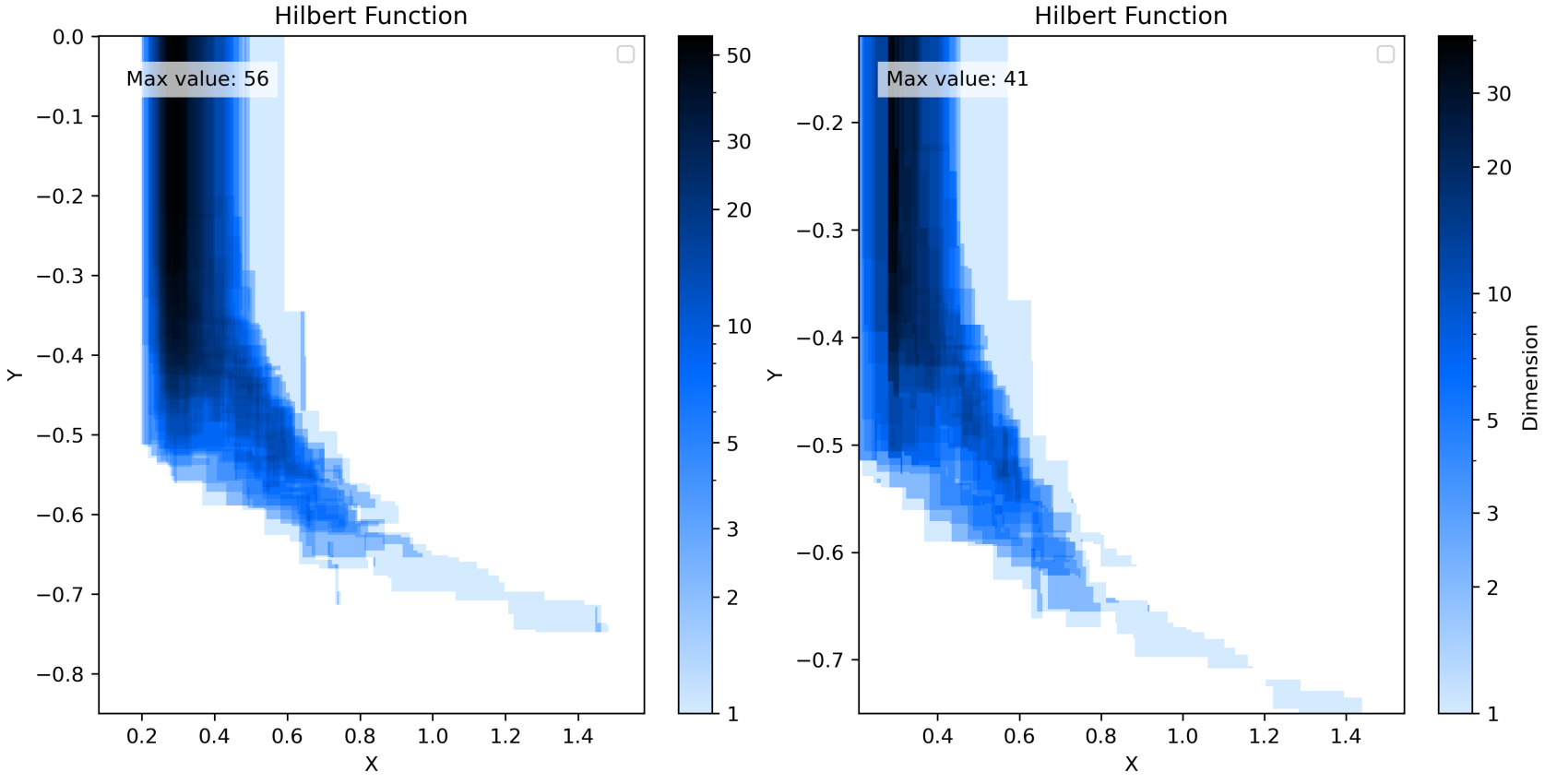
Results

Its largest component and the 0.02-pruning of the largest component:



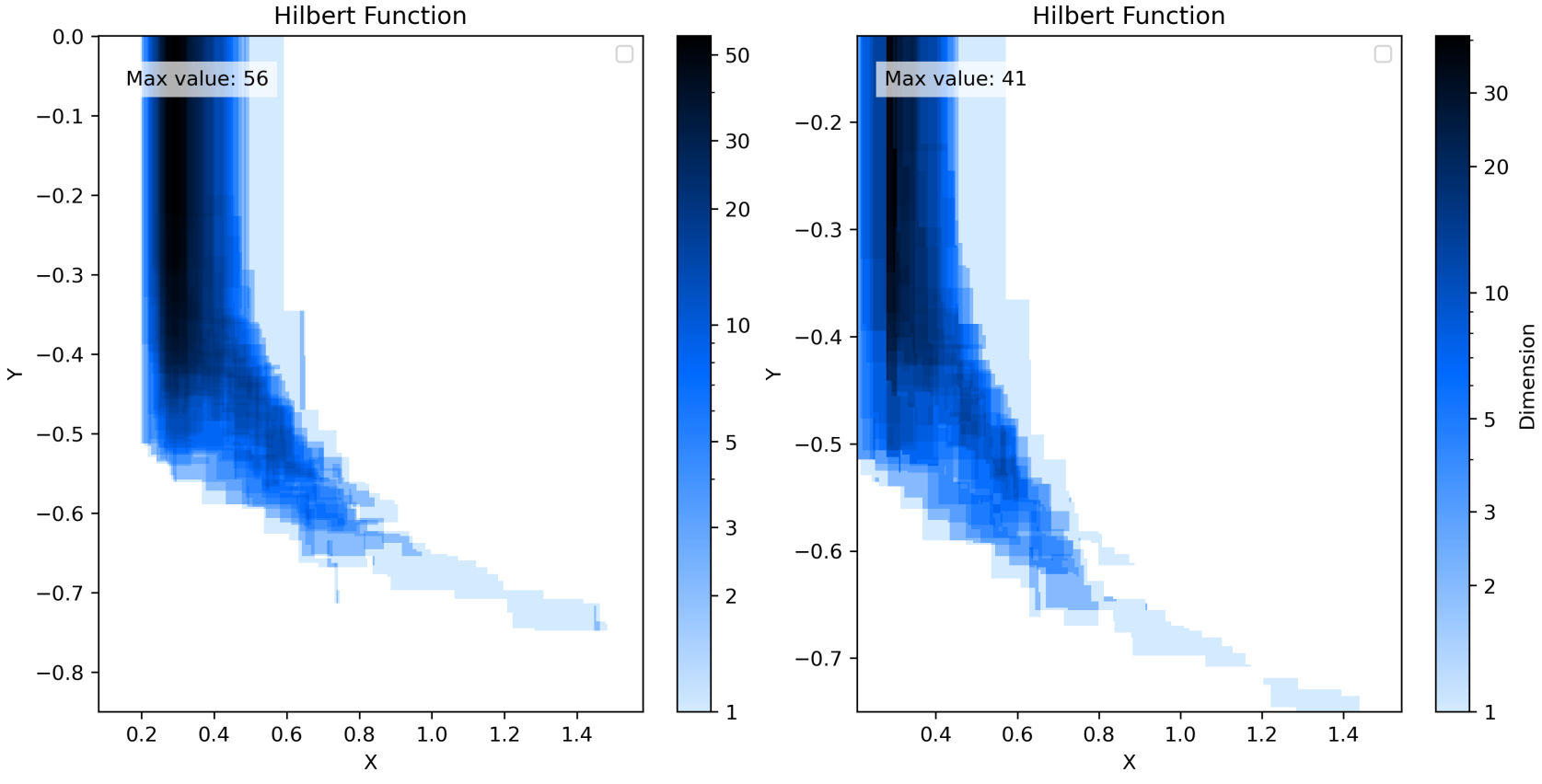
Results

Its largest component and the 0.02-pruning of the largest component:

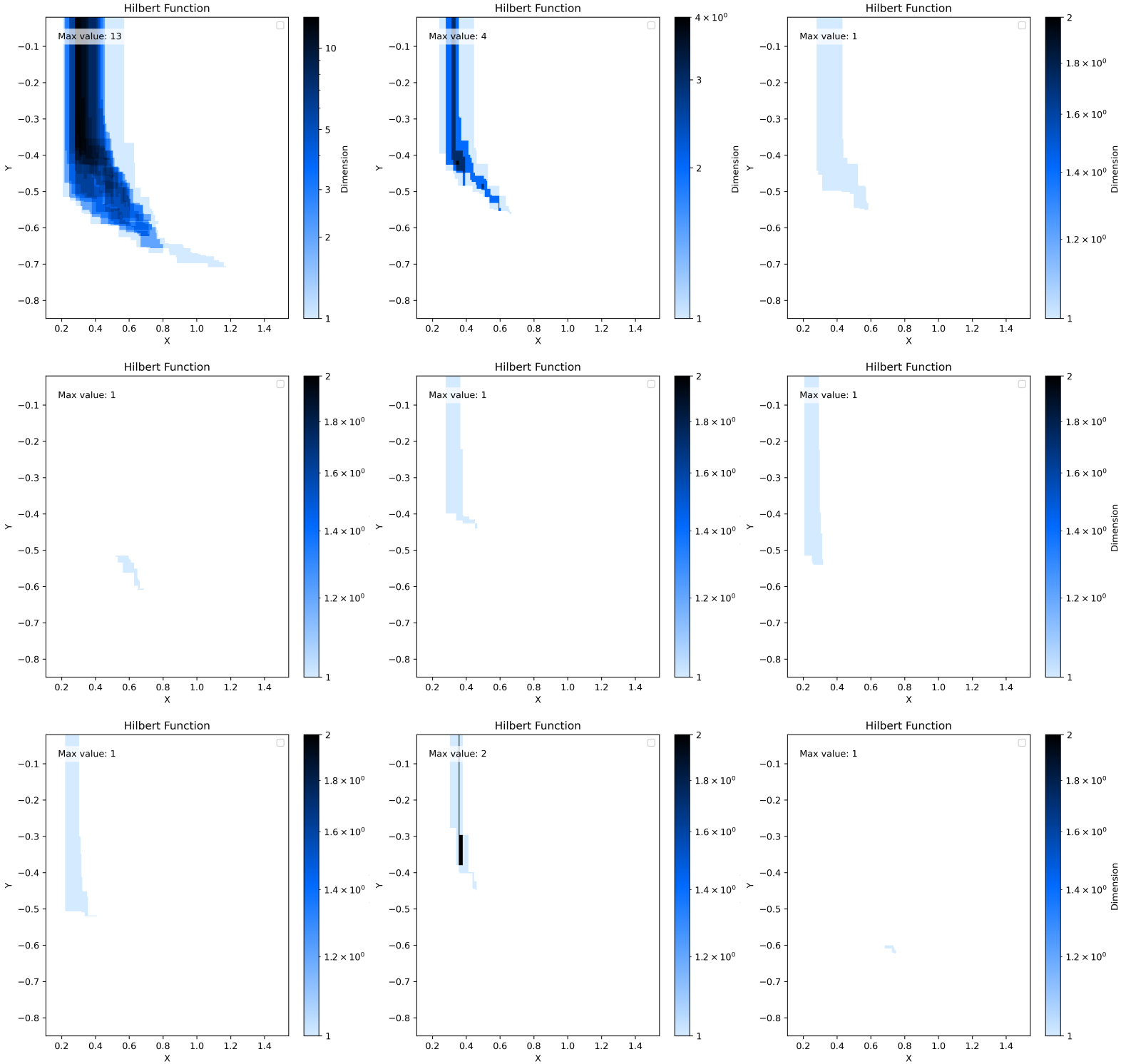


Results

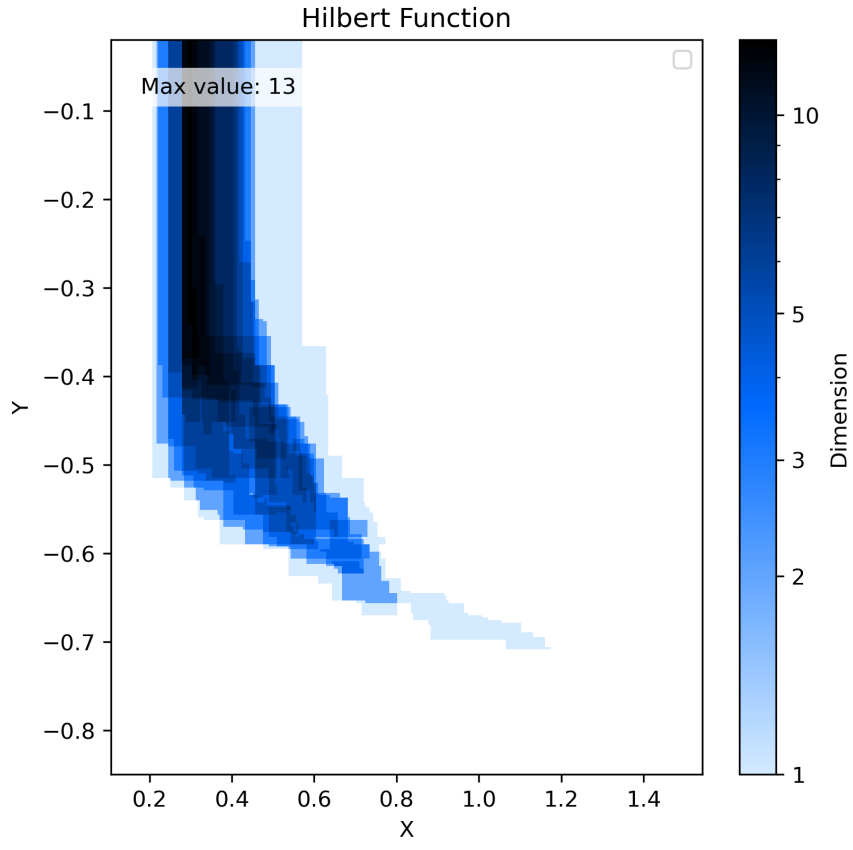
Its largest component and the 0.02-pruning of the largest component:



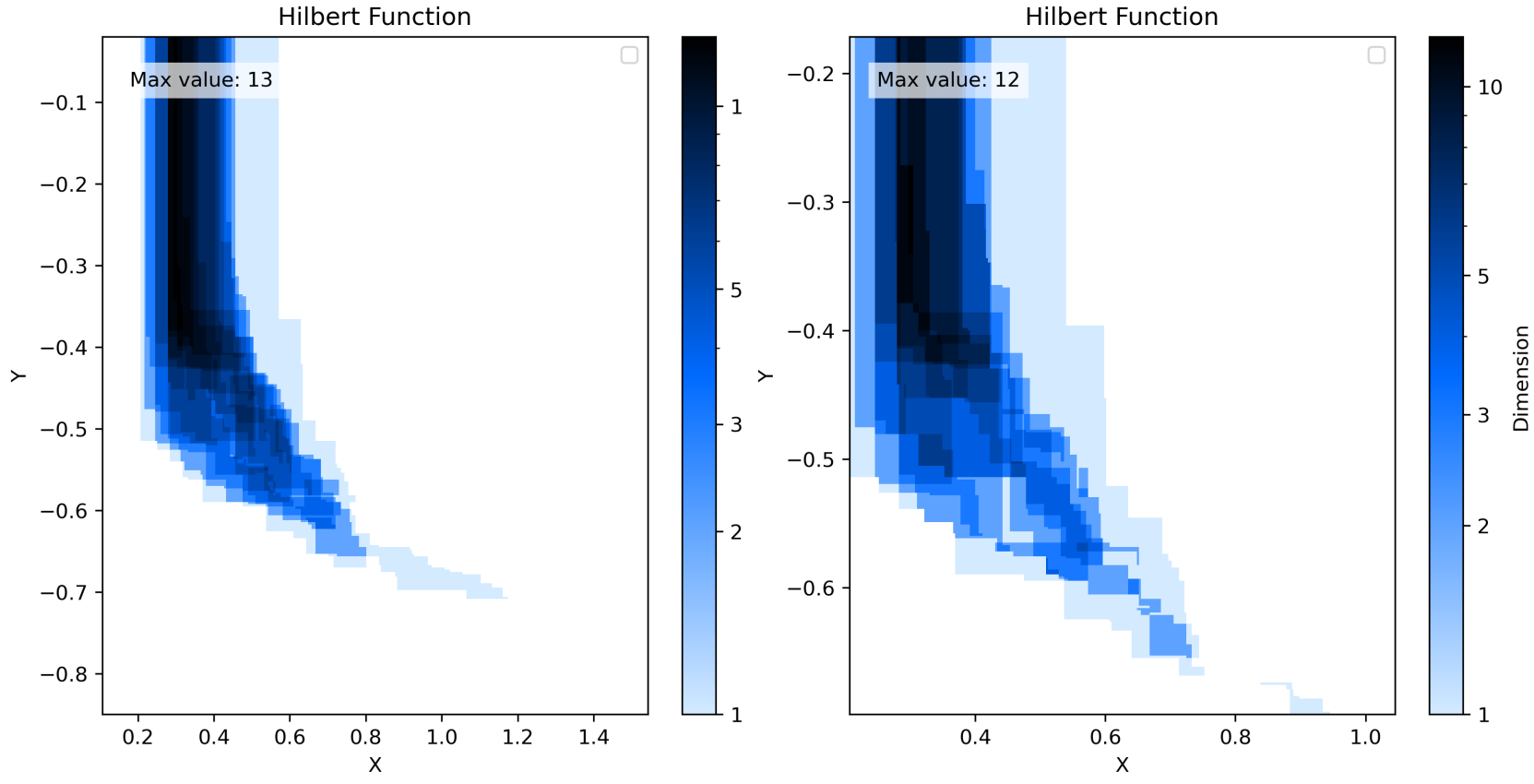
86 components


Hilbert Function
Max value: 13
Hilbert Function
Max value: 4
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 2
Hilbert Function
Max value: 1
X
Y
Dimension

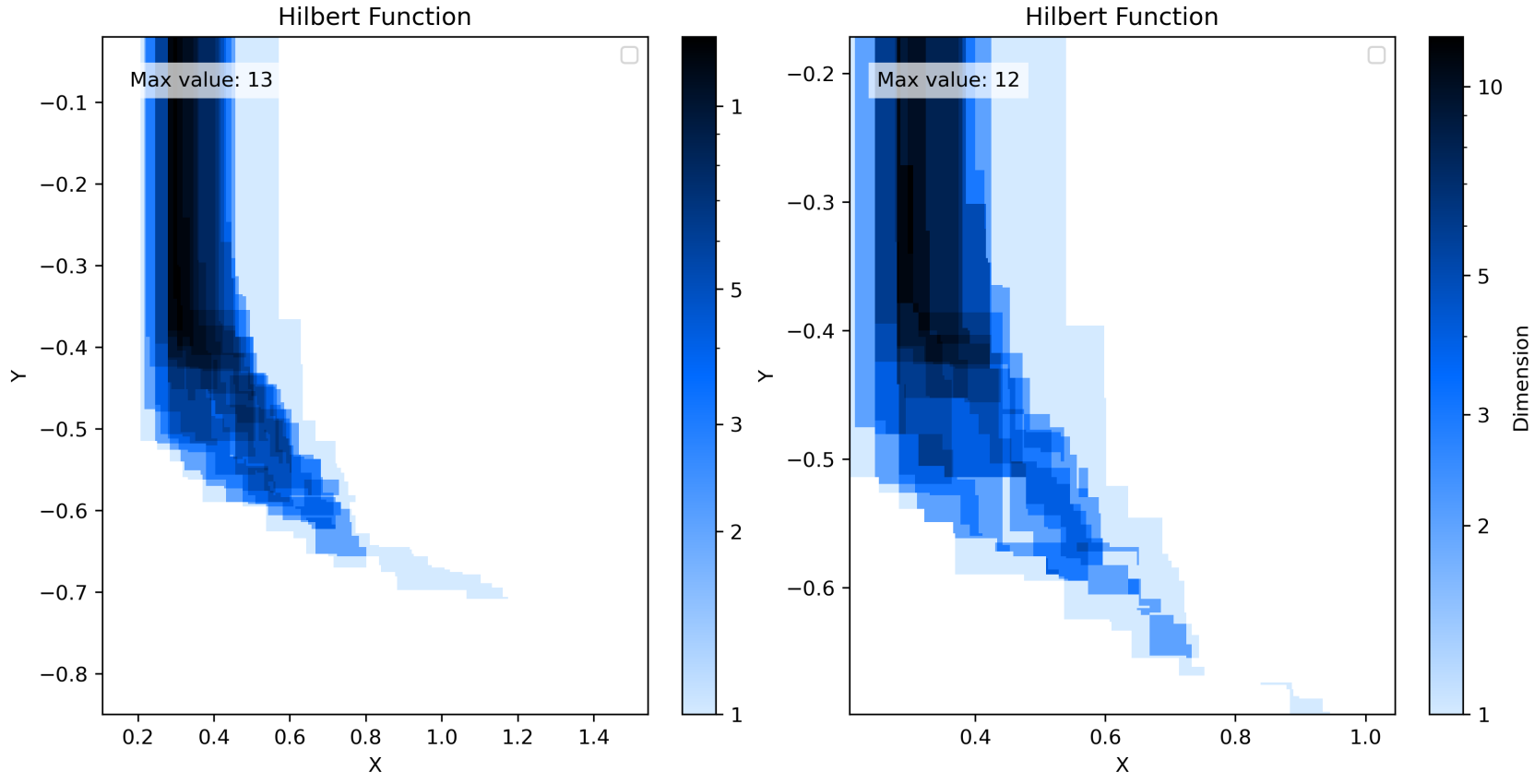
The largest component of the 0.02-pruning and its 0.03-pruning:



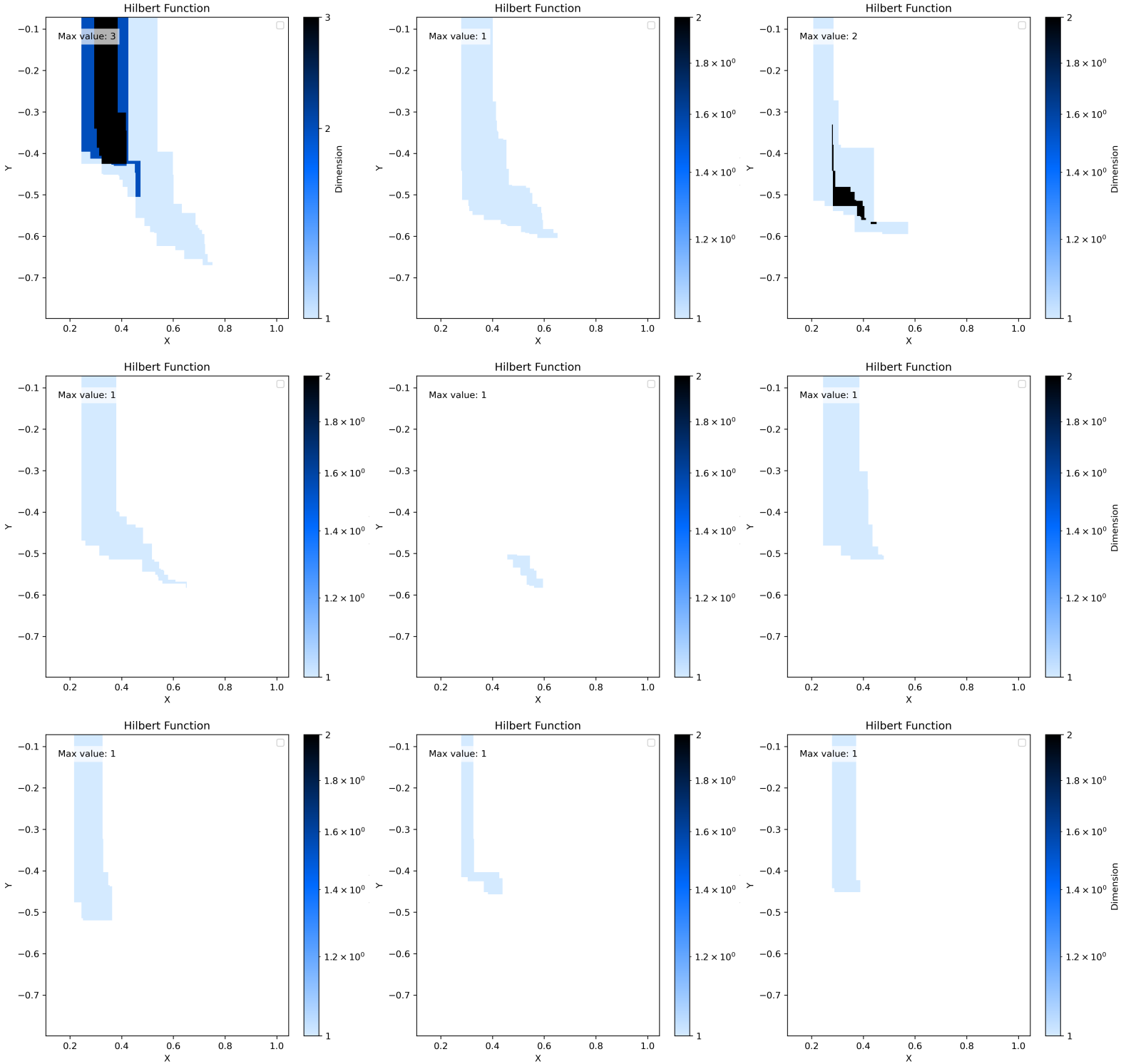
The largest component of the 0.02-pruning and its 0.03-pruning:



The largest component of the 0.02-pruning and its 0.03-pruning:

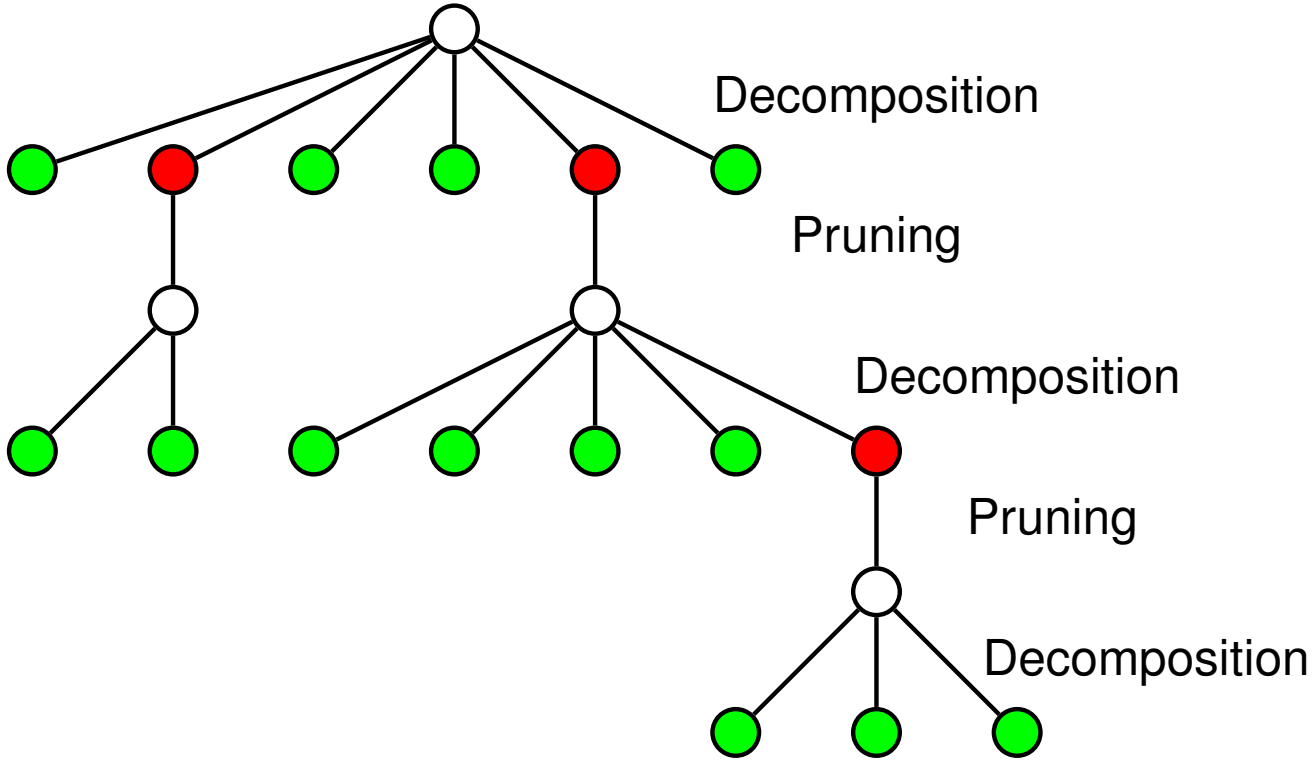


20 components


Hilbert Function
Max value: 3
Hilbert Function
Max value: 1
Hilbert Function
Max value: 2
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
Hilbert Function
Max value: 1
X
Y
Dimension

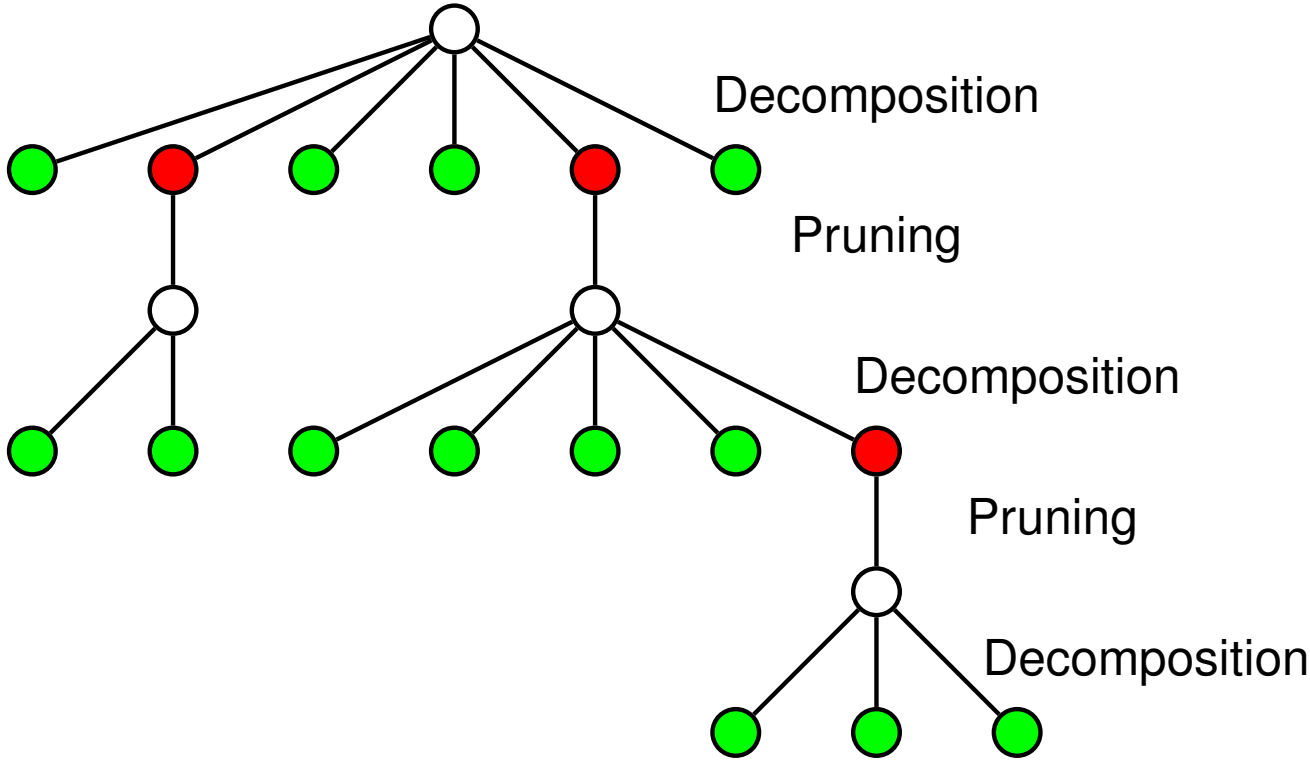
Tree of decompositions? (Barkcode?)

One could do one big pruning or a sequence of smaller prunings to prune everything into simple pieces.



Tree of decompositions? (Barkcode?)

One could do one big pruning or a sequence of smaller prunings to prune everything into simple pieces.



Resulting barcode = {leaf nodes}.

Next steps and questions

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?
- exploratory work:

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?
- exploratory work:
  - what does a module look like when it comes from a data set with a few clear, significant features?

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?
- exploratory work:
  - what does a module look like when it comes from a data set with a few clear, significant features?
  - how do the modules depend on the filtration that is being used?

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?
- exploratory work:
  - what does a module look like when it comes from a data set with a few clear, significant features?
  - how do the modules depend on the filtration that is being used?

Code: https://github.com/JanJend/Stable-Decomposition

Next steps and questions

- Clean up the code/implement optimizations/make the software more user-friendly
- measure how much of the module we delete when pruning a module
- representative cycles?
- exploratory work:
  - what does a module look like when it comes from a data set with a few clear, significant features?
  - how do the modules depend on the filtration that is being used?

Code: https://github.com/JanJend/Stable-Decomposition

Thank you!