Dynamic cycle representatives

Chains undergoing change in persistent homology

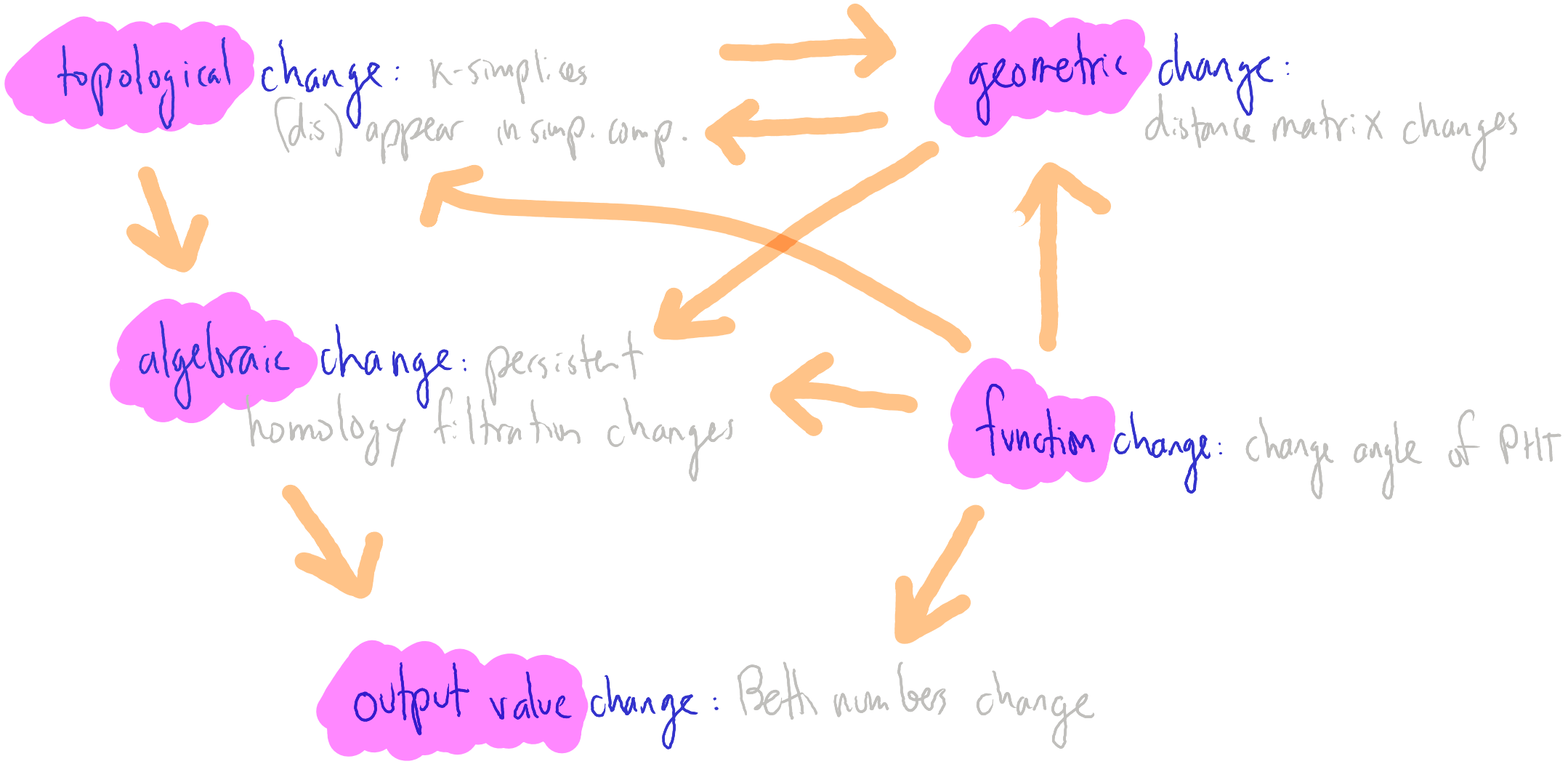
August 12, 2025
Jānis Lazovskis, University of Latvia
BIRS "Cycle representatives in applied homological algebra"

1. Introduction
2. Development of dynamics
3. Computation

Līdzfinansē Eiropas Savienība
2027 Nacionālais attīstības plāns
Latvijas Zinātnes padome
LATVIJAS UNIVERSITĀTE
PostDoc Latvia

Funding provided by Activity 1.1.1.9 "Post-doctoral Research" of the Specific Objective 1.1.1 "Strengthening research and innovative capacities and introduction of advanced technologies in the common R&D system" of the European Union's Cohesion Policy Programme for 2021-2027 research application No 1.1.1.9/LZP/1/24/ 125 of the Activity "Post-doctoral Research" "Efficient topological signatures for representation learning in medical imaging"

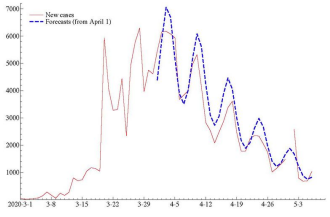
What is "dynamic"?

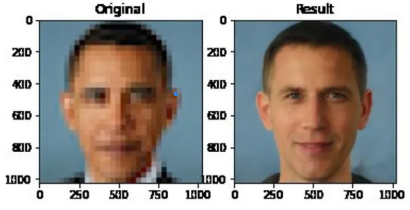


Where does "dynamic" come from? What should "dynamic" be?

*Motivated by the real world

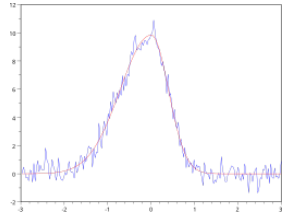
- flocking behaviour (motion tracking, social networks)
- time evolution (Takens embedding, sliding windows)
- streaming data (noise management, updates)

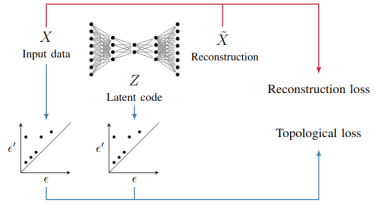




*Aligned to existing paradigms

- stable, vectorizable objects
- computable, efficient algorithms
- standard barcode algorithm





*This talk will not be about:

- choice of distance function
- categorical perspective
- dynamical systems

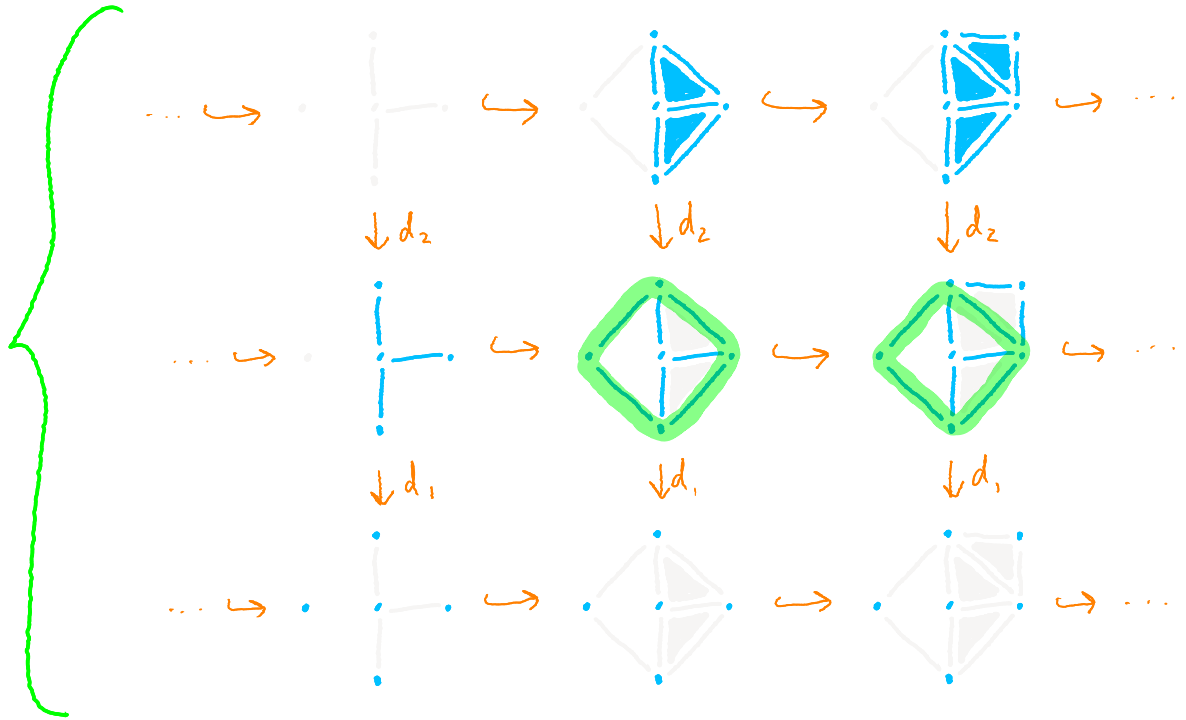
Definitions:

chain complex : $C_\bullet : \cdots \to C_n \xrightarrow{d_n} C_{n-1} \xrightarrow{d_{n-1}} C_{n-2} \to \cdots$

filtration : $\cdots \to K_{r_1} \to K_{r_2} \to K_{r_3} \to \cdots$

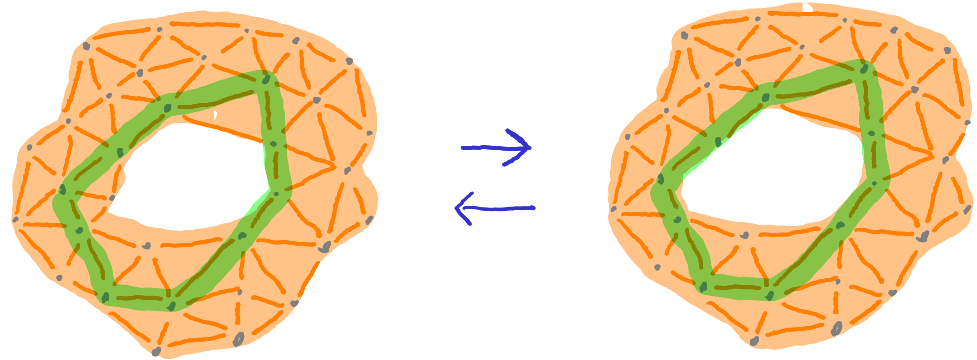
} persistence module at deg. n:
Chain complex homology at deg. n., joined by filtration maps

dynamic cycle (example of)

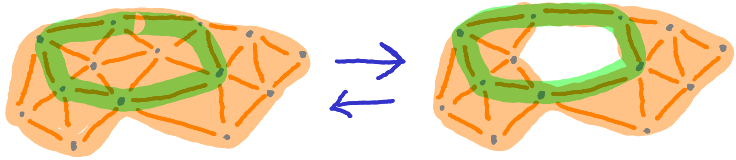


Cycle settings:

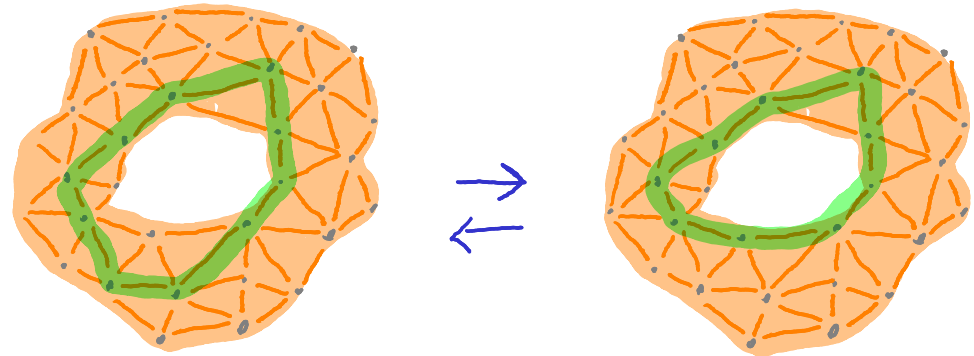
1. same cycle, different class:



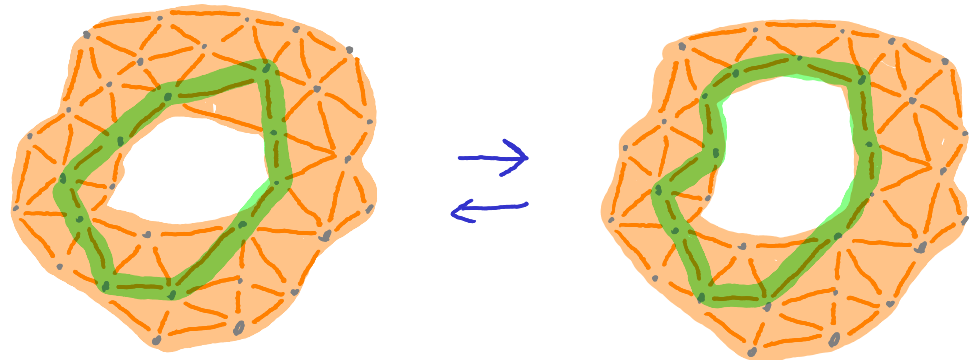
*or:



2. different cycle, same class:



3. different cycle, different class:



Algebraic changes :

Dynamics in a filtration

$$\mathcal{F} : Q \xrightarrow{\nabla_0} K_0 \xrightarrow{\nabla_1} K_1 \xrightarrow{\nabla_2} K_2 \xrightarrow{\nabla_3} K_3 \xrightarrow{\nabla_4} K_4 = K$$

• swap :

$$\mathcal{F}_S : Q \xrightarrow{\nabla_0} K_0 \xrightarrow{\nabla_1} K_1 \xrightarrow{\nabla_3} K_3 \xrightarrow{\nabla_2} K_2 \xrightarrow{\nabla_4} K_4 = K$$

Vines and Vineyards by Updating Persistence in Linear Time *

David Cohen-Steiner, INRIA, 2004 Route des Lucioles, BP93, Sophia-Antipolis, France, dcohen@sophia.inria.fr
Herbert Edelsbrunner, Dept Computer Science, Duke University, Durham, Geomagic, RTP, North Carolina, USA, edels@cs.duke.edu
Dmitriy Morozov, Dept Computer Science, Duke University, Durham, North Carolina, USA, morozov@cs.duke.edu

SoCG 2006

Home > Computing and Combinatorics > Conference paper

Tracking a Generator by Persistence

Conference paper

pp 278–287 | Cite this conference paper

Computing and Combinatorics

(COCOON 2010)

Oleksiy Busaryev, Tamal K. Dey & Yusu Wang

Part of the book series: Lecture Notes in Computer Science ((LNTCS,volume 6196))

Access this chapter

Log in via an institution →

COCOON 2010

"moves" as successive swaps

• remove :

$$\mathcal{F}_R : Q \xrightarrow{\nabla_0} K_0 \xrightarrow{\nabla_2} K_2^- \xrightarrow{\nabla_3} K_3^- \xrightarrow{\nabla_4} K_4^- = K^-$$

arXiv > math > arXiv:2312.03925

Mathematics > Algebraic Topology

(Submitted on 6 Dec 2023 (v1), last revised 22 May 2025 (this version, v2))

Pruning vineyards: updating barcodes and representative cycles by removing simplices

Barbara Giunti, Jānis Lazovskis

The barcode of a filtration and its representative cycles encode rich information often useful in data analysis. However, obtaining them can be computationally expensive. Therefore, it is useful to have methods that update them if the associated filtration undergoes small changes. There are already efficient algorithms updating a barcode if simplices exchange entrance order or are added, but not if simplices are removed. We provide an implementation to update a reduced boundary matrix when simplices in the filtration are removed. Our algorithm, the Simplicial Removal Update Procedure (SiRUP), intrinsically updates also the representative cycles, and is compatible with the twist optimizations. We show that the complexity of our algorithm is lower than recomputing the barcode from scratch and that the number of executed matrix column additions is minimal, with both theoretical and experimental methods.

arXiv 2025

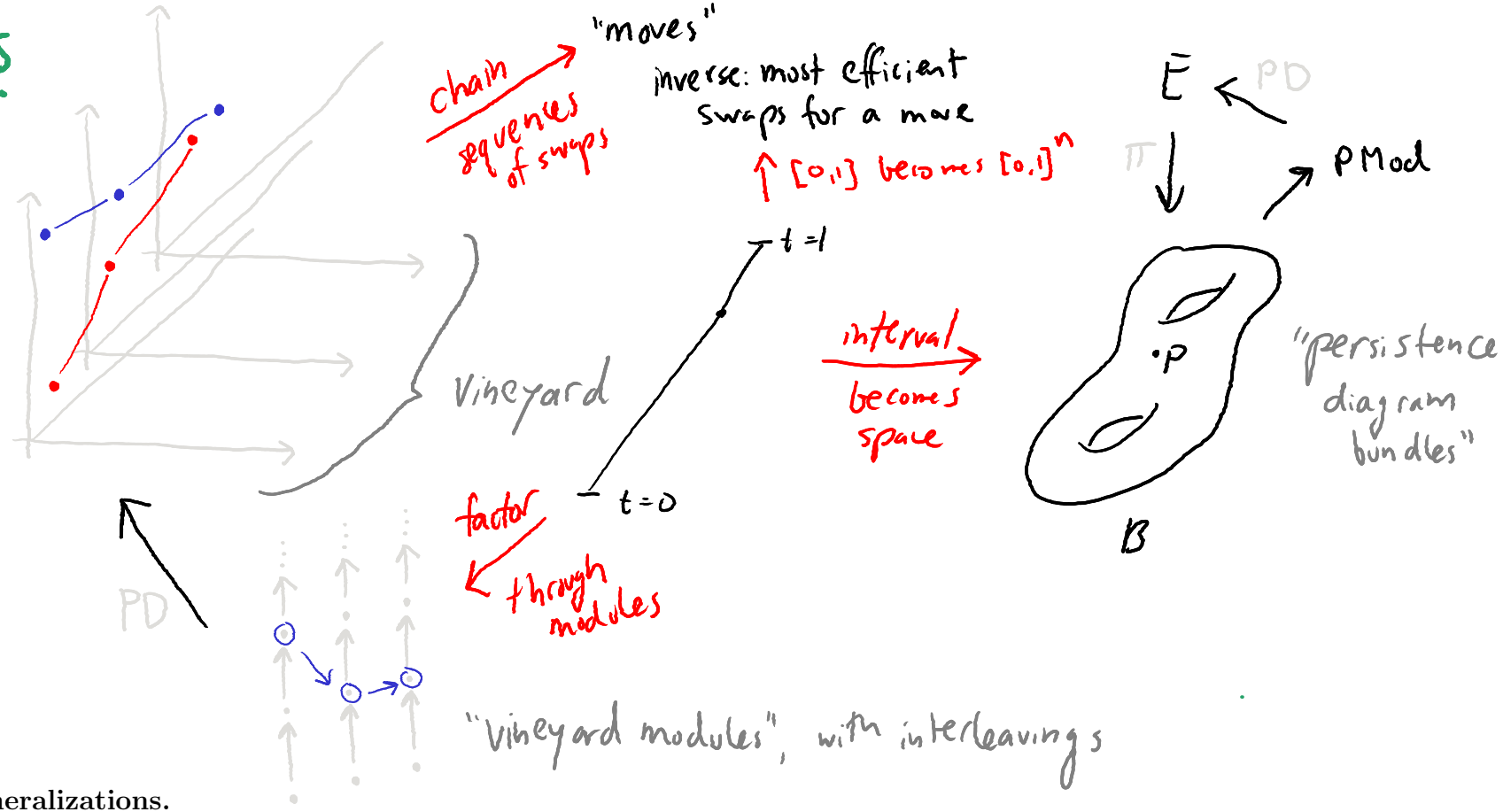
• add :

$$\mathcal{F}_A : Q \xrightarrow{\nabla_0} K_0 \xrightarrow{\tau} K_\tau \xrightarrow{\nabla_1} K_1^+ \xrightarrow{\nabla_2} K_2^+ \xrightarrow{\nabla_3} K_3^+ \xrightarrow{\nabla_4} K_4^+ = K^+$$

Developing dynamics

Vine: [birth, death) moving in persistence diagram

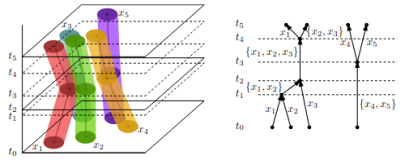
E is the "space of p.d.'s"
$\cong Conf_{\leq n}(\mathbb{R}^2_+)$



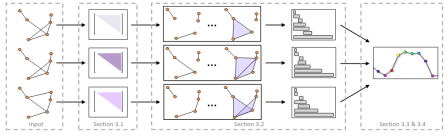
Vines, vineyards, their topological generalizations.

introduces "swaps":	(2006 Cohen–Steiner, Edelsbrunner, Morozov) *Vines and vineyards by updating persistence in linear time*
proves stability for swaps:	(2013 Munch) *Applications of Persistent Homology to Time Varying Systems*
chains swaps into "moves":	(2010 Busaryev, Dey, Y.Wang) *Tracking a Generator by Persistence*
improves efficiency of "moves":	(2024 Piekenbrock, Perea) *Move schedules: fast persistence computations in coarse dynamic settings*
generalizes from interval to arbitrary space:	(2022 Hickok) *Persistence Diagram Bundles: A Multidimensional Generalization of Vineyards*
computational details:	(2022 Hickok) *Computing Persistence Diagram Bundles*
factors vineyards through modules:	(2023 Turner) *Representing Vineyard Modules*

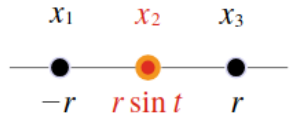
Developing dynamics



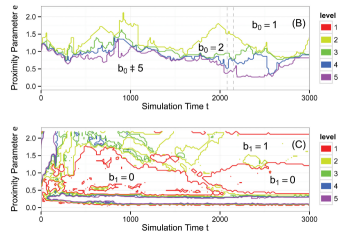
Reeb graphs for dynamic clusters



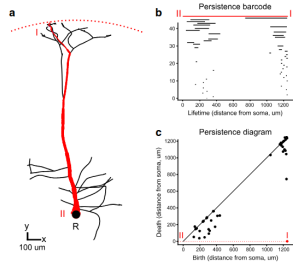
PH on dynamic edge-weighted graphs



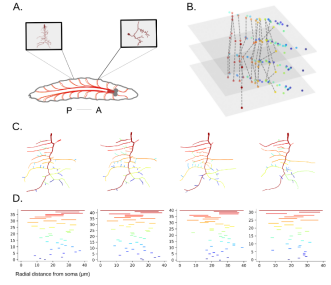
dynamic metric space (X,d_t)



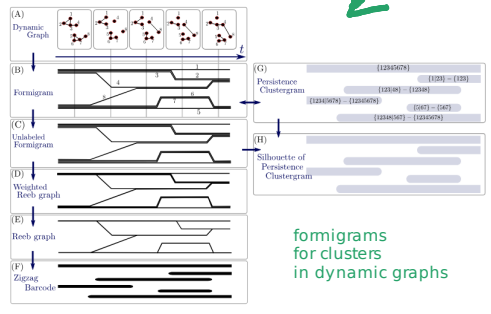
CROCKER plots



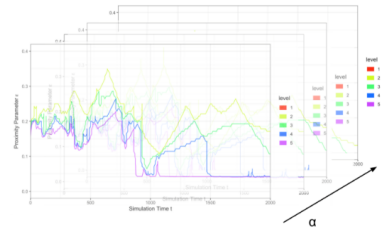
topological morphology descriptor



TMDs over biological development



formigrams for clusters in dynamic graphs

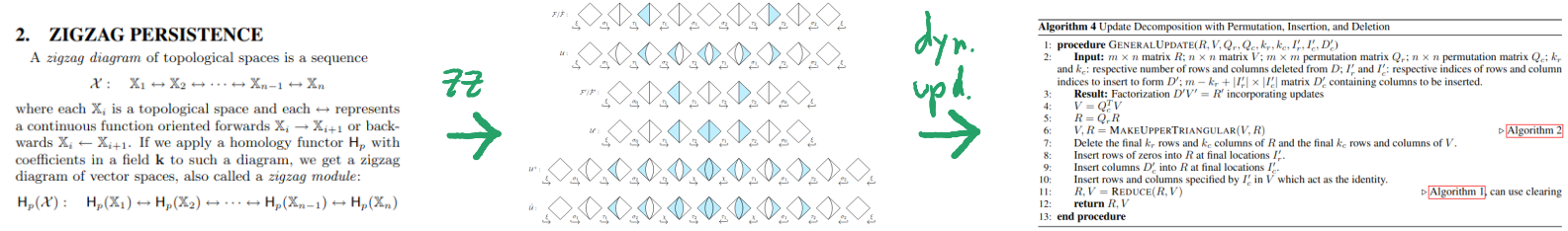


crocker stacks

Dynamic graphs and dynamic spaces.

dynamic edge-weighted graphs:	(2018 Hajij, B.Wang, Scheidegger, Rosen) *Visual Detection of Structural Changes in Time-Varying Graphs Using Persistent Homolog*
generalized dynamic spaces:	(2021 Kim, Memoli) *Spatiotemporal Persistent Homology for Dynamic Metric Spaces*
Reeb graphs for dynamic clusters:	(2013 Buchin, Buchin, van Kreveld, Speckmann, Staals) *Trajectory Grouping Structure*
dynamic clusters in spaces:	(2018 Kim, Memoli) *Formigrams: Clustering Summaries of Dynamic Data*
dynamic clusters in PDs:	(2023 Kim, Memoli) *Extracting Persistent Clusters in Dynamic Data via Möbius Inversion*
topological morphology descriptor:	(2018 Kanari, Dłotko, Scolamiero, Levi, Shillcock, Markram) *A Topological Representation of Branching Neuronal Morphologies*
TMDs over time:	(2025 Rigaux, Castro, Kanari) *Quantifying neuronal differentiation using temporal topological persistence*
CROCKER plot:	(2015 Topaz, Ziegelmeier, Halverson) *Topological data analysis of biological aggregation models*
statstics with crocker plots:	(2019 Ulmer, Ziegelmeier, Topaz) *A topological approach to selecting models of biological experiments*
crocker stack:	(2022 Xian, Topaz, Adams, Ziegelmeier) *Capturing dynamics of time-varying data via topology*

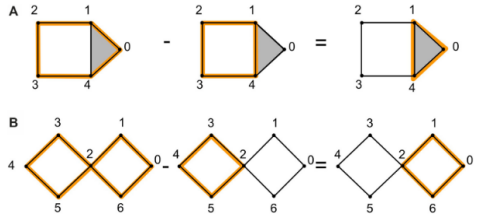
Developing dynamics



introduction of zigzag PH

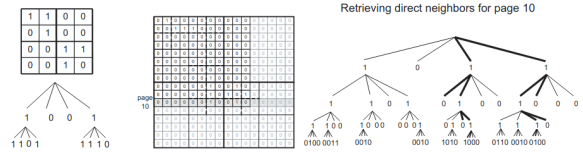
Dey-Hou on computing zigzag PH in all dynamic scenarios

Luo-Nelson on updating PH in all dynamic scenarios



optimal cycles for compact representation

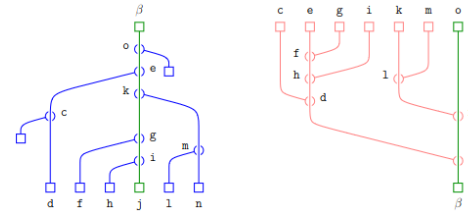
minimal data struc.



k2-trees as compact data structures

clever data struc.

algorithm for dynamic updates

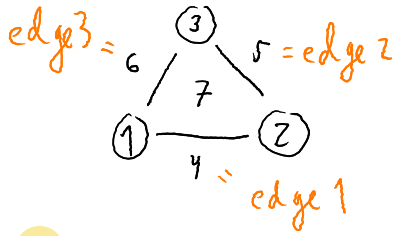


efficient data structures for updating PH in all dynamic scenarios on 1D spaces

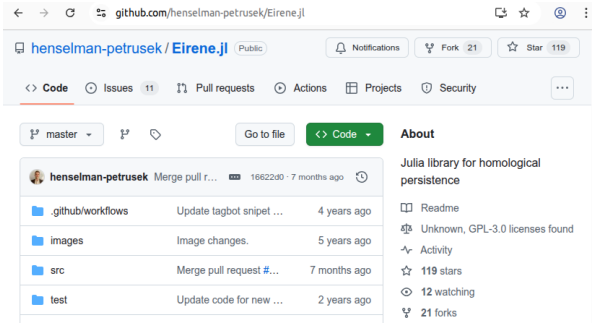
Zigzags, computation, data structures.

zigzag persistent homology:	(2009 Carlsson, de Silva, Morozov) *Zigzag persistent homology and real-valued functions*
swaps using zigzags:	(2022 Dey, Hou) *Updating Barcodes and Representatives for Zigzag Persistence*
computational details:	(2022 Dey, Hou) *Fast Computation of Zigzag Persistence*
removal and addition using zigzags:	(2024 Dey, Hou) *Computing Zigzag Vineyard Efficiently Including Expansions and Contractions*
universal barcode updates:	(2023 Luo, Nelson) *Accelerating Iterated Persistent Homology Computations with Warm Starts*
efficient data structures for updates:	(2024 Cultrera di Montesano, Edelsbrunner, Henzinger, Ost) *Dynamically Maintaining the Persistent Homology of Time Series*
tracking PH efficiently in 1D spaces:	(2023 Biswas, Cultrera di Montesano, Edelsbrunner, Saghafian) *Geometric characterization of the persistence of 1D maps*
compact k^2-trees:	(2009 Brisaboa, Ladra, Navarro) *k^2-Trees for Compact Web Graph Representation*
minimal cycle overview:	(2021 Li, Thompson, Henselman-Petrusek, Giusti, Ziegelmeier) *Minimal Cycle Representatives in Persistent Homology Using Linea*

Computing dynamics: retrieving cycles



Eirene, C. Henselman-Petrusek 2016



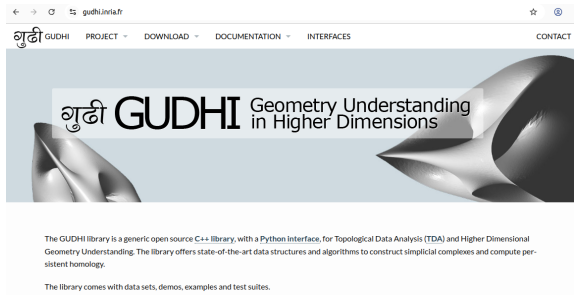
simplex_eirene.txt

```
0, 0.1
0, 0.2
0, 0.3
1, 0.4, 1, 2
1, 0.5, 2, 3
1, 0.6, 1, 3
2, 0.7, 4, 5, 6
```

```
$ juliaup add 1.7.0
$ julia +1.7.0
julia> using Pkg
julia> Pkg.add("Eirene")
julia> using Eirene
julia> C = eirene("simplex_eirene.txt", model="complex", entryformat="sp")
julia> barcode(C,dim=0)
3×2 Matrix{Float64}:
 0.2   0.4
 0.3   0.5
 0.1   Inf
julia> barcode(C,dim=1)
1×2 Matrix{Float64}:
 0.6  0.7
julia> C["cyclerep"]
3-element Vector{Vector{Vector{Int64}}}:
 []
 [[1, 2], [1, 3], [1]]
 [[1, 2, 3]]
```

* input formats
* indexation

GUDHI, ... 2013



simplex_gudhi.cpp

```cpp
template <class Matrix>
void print_representative_cycles_example()
{
  Matrix mp({ { },
              { },
              { },
              { 0, 1 },
              { 1, 2 },
              { 0, 2 },
              { 0, 1, 2 }
            });
```

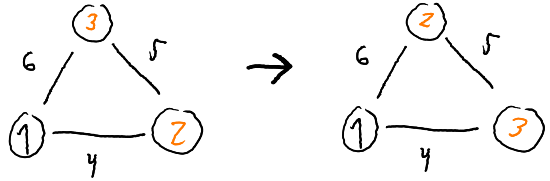
```
$ g++ -o simplex_gudhi -I gudhi.3.11.0/include simplex_gudhi.cpp
$ ./simplex_gudhi
RU_matrix:
0-bar: (0,6)
0-bar: (1,3)
0-bar: (2,4)
1-bar: (5,4294967295)
0-cycle: 0,
0-cycle: 1,
0-cycle: 2,
1-cycle: 3, 4, 5,
```

persistence_matrix_options.h

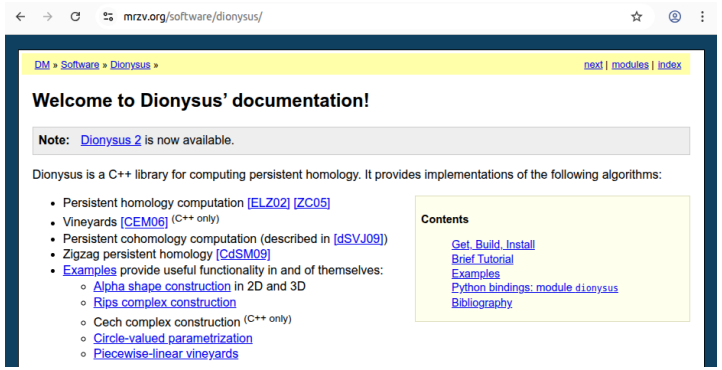
```cpp
  static const bool has_column_compression = false;
  static const bool has_column_and_row_swaps = false;

  static const bool has_map_column_container = false;
  static const bool has_removable_columns = false;

  static const bool has_row_access = false;
  static const bool has_intrusive_rows = true;
  static const bool has_removable_rows = false;

  static const bool is_of_boundary_type = true;

  static const bool has_matrix_maximal_dimension_access = false;
  static const bool has_column_pairings = true; // false
  static const bool has_vine_update = true; // false
  static const bool can_retrieve_representative_cycles = false;
};
```

* C++ experience
* Python modules

Computing dynamics: cycle modification



Dionysus 1, D. Morozov, 2009



simplex_dionysus.txt

```
0
1
2
0 1
1 2
0 2
```

values_dionysus.txt

```
0.1 0.2 0.3
0.1 0.3 0.2
```

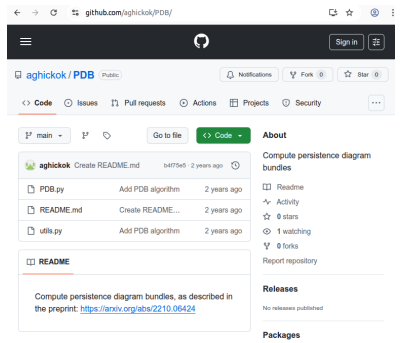
```
$ g++ -o pl-vineyard pl-vineyard.cpp -I ../../include/ -lboost_program_options
$ ./pl-vineyard simplex_dionysus.txt values_dionysus.txt vineyard
Simplices read:
<0>
<1>
<2>
<0, 1>
<1, 2>
<0, 2>
Complex read, size: 6
Vertex values read:
0.1 0.2 0.3
0.1 0.3 0.2

0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************
Pairing computed
Processed frame: 1
Vineyard computed
```

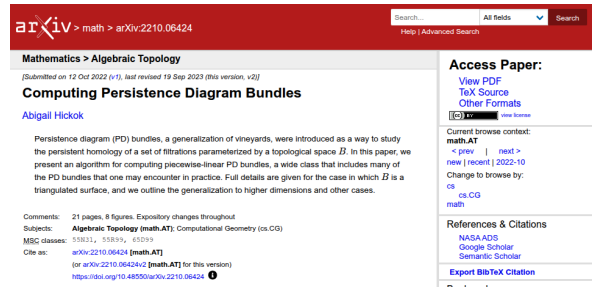
vineyard1.edg

```
0.3 inf 0
0.25 inf 0.5
0.25 inf 0.5
0.3 inf 1
```

* only vertices get values

PDB, A. Hickok, 2023



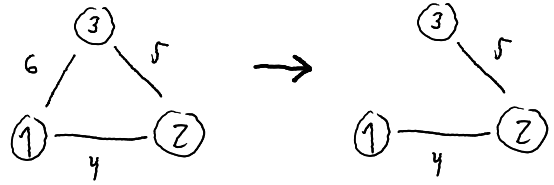



```
>>> exec(open('PDB.py').read())
>>> v1 = Simplex(nodes=np.array([1]), val=np.expand_dims([.1,.1],axis=0)); v1.curr_idx = 0
>>> v2 = Simplex(nodes=np.array([2]), val=np.expand_dims([.2,.3],axis=0)); v2.curr_idx = 1
>>> v3 = Simplex(nodes=np.array([3]), val=np.expand_dims([.3,.2],axis=0)); v3.curr_idx = 2
>>> e1 = Simplex(nodes=np.array([1,2]), val=np.expand_dims([.4,.4],axis=0)); e1.curr_idx = 3
>>> e2 = Simplex(nodes=np.array([2,3]), val=np.expand_dims([.5,.5],axis=0)); e2.curr_idx = 4
>>> e3 = Simplex(nodes=np.array([1,2]), val=np.expand_dims([.6,.6],axis=0)); e3.curr_idx = 5
>>> simplices = [v1,v2,v3,e1,e2,e3]
>>> x = RU(simplices)
>>> x.swap(v2,v3)
False
```

?

* under development

Computing dynamics: cycle modification



GUDHI, ... 2013

vine_swap() [1/2]

template<class PersistenceMatrixOptions >

Matrix< PersistenceMatrixOptions >::index
Gudhi::persistence_matrix::Matrix< PersistenceMatrixOptions
>::vine_swap (index columnIndex1,
index columnIndex2
)

Only available if **PersistenceMatrixOptions::has_vine_update** is true and if it is either a **chain matrix** or **PersistenceMatrixOptions::column_indexation_type** is set to Column_indexation_types::IDENTIFIER. Does a vine swap between two faces which are consecutives in the filtration. Roughly, if F is the current filtration represented by the matrix, the method modifies the matrix such that the new state corresponds to a valid state for the filtration F' equal to F but with the two given faces at swapped positions. Of course, the two faces should not have a face/coface relation which each other ; F' has to be a valid filtration. See **[21]** for more information about vine and vineyards.

Parameters

columnIndex1	**MatIdx** index of the first face.
columnIndex2	**MatIdx** index of the second face. It is assumed that the **PosIdx** of both only differs by one.

Returns

Let $pos1$ be the **PosIdx** index of `columnIndex1` and $pos2$ be the **PosIdx** index of `columnIndex2`. The method returns the **MatIdx** of the column which has now, after the swap, the **PosIdx** $max(pos1, pos2)$.

CONTACT

Search

operator=
remove_column
remove_last
remove_maximal_face
remove_maximal_face
set_characteristic
swap_columns
swap_rows
update_representative_cycles
vine_swap
vine_swap
vine_swap_with_z_eq_1_case
vine_swap_with_z_eq_1_case
zero_cell
zero_cell
zero_column
zero_column
swap
Base_matrix
Base_matrix_with_column_compression
Base_pairing

remove_column()

template<class PersistenceMatrixOptions >

void Gudhi::persistence_matrix::Matrix< PersistenceMatrixOptions
>::remove_column (index columnIndex)

Only available for **base matrices** without column compression and if **PersistenceMatrixOptions::has_map_column_container** is true. Otherwise, see **remove_last**. Erases the given column from the matrix. If **PersistenceMatrixOptions::has_row_access** is also true, the deleted column cells are also automatically removed from their respective rows.

Parameters

columnIndex	**MatIdx** index of the column to remove.

* removal as swap+drop
* different structure than for get_barcode

PHAT-SiRUP, B. Giunti, J.L. 2025

bitbucket.org/jlazovskis/phat-sirup/src/master/

Pull requests Repositories Projects Search

arXiv > math > arXiv:2312.03925

Mathematics > Algebraic Topology

[Submitted on 6 Dec 2023 (v1), last revised 22 May 2025 (this version, v2)]

Pruning vineyards: updating barcodes and representative cycles by removing simplices

Barbara Giunti, Jānis Lazovskis

The barcode of a filtration and its representative cycles encode rich information often useful in data analysis. However, obtaining them can be computationally expensive. Therefore, it is useful to have methods that update them if the associated filtration undergoes small changes. There are already efficient algorithms updating a barcode if simplices exchange entrance order or are added, but not if simplices are removed. We provide an implementation to update a reduced boundary matrix when simplices in the filtration are removed. Our algorithm, the Simplicial Removal Update Procedure (SiRUP), intrinsically updates also the representative cycles, and is compatible with the twist optimizations. We show that the complexity of our algorithm is lower than recomputing the barcode from scratch and that the number of executed matrix column additions is minimal, with both theoretical and experimental methods.

README.md

PHAT-sirup

A fork of the Persistent Homology Algorithm Toolbox for when simplices are removed from the filtrations, using the SiRUP algorithm to compute updates to the barcode and representative cucles. This code is developed and maintained by Barbara Giunti and Jānis Lazovskis.

- (source software) Phat – Persistent Homology Algorithms Toolbox, 2017 (Bauer, Kerber, Reininghaus, Wagner)
- (updates to persistent homology) Vines and vineyards by updating persistence in linear time, 2006 (Cohen-Steiner, Edelsbrunner, Morozov)
- (SiRUP algorithm) Pruning vineyards: updating barcodes and representative cycles by removing simplices, 2025 (Giunti, Lazovskis)

simplex_sirup.bmat

```
0
0
0
1 0 1
1 1 2
1 0 2
```

simplex_sirup.updates

```
remove 5
```

```
$ g++ -o phat_sirup phat.cpp -std=c++20 -I ../include/

$ ./phat_sirup --ascii \
--bmat_in simplex_sirup.bmat \
--bmat_out simplex_sirup_reduced.bmat \
--opmat_out simplex_sirup.opmat \
--pairs_out simplex_sirup.pairs

$ ./phat_sirup --ascii \
--bmat_in simplex_sirup_reduced.bmat \
--opmat_in simplex_sirup.opmat \
--updates simplex_sirup.updates \
--pairs_out simplex_removed_sirup.pairs
```

simplex_sirup_reduced.bmat

```
0
0
0
1 0 1
1 1 2
1
```

simplex_sirup.opmat

```
0 0
0 1
0 2
0 3
0 4
0 3 4 5
```

simplex_sirup.pairs

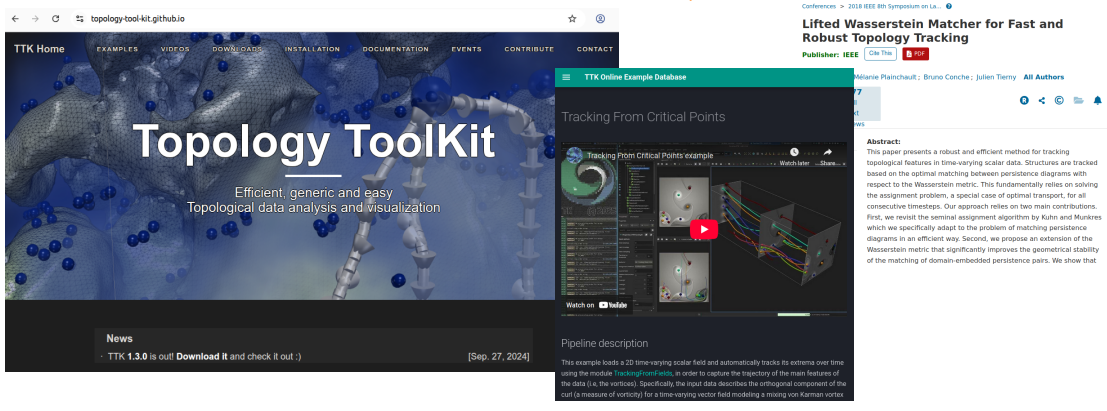
```
2
1 3
2 4
```

* next: add + swap

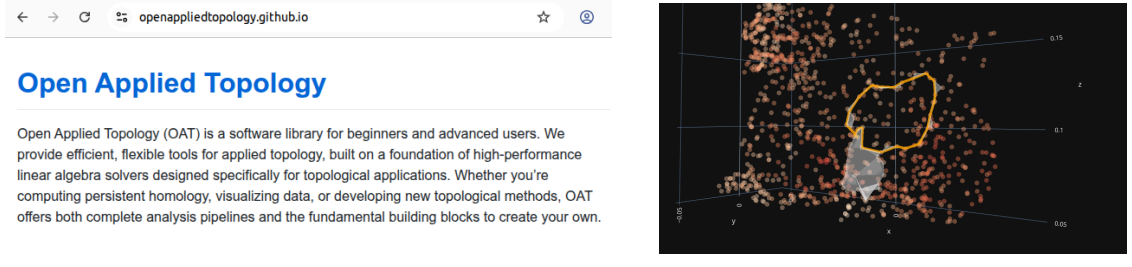
Computing dynamics: more software

HomCloud, I. Obayashi, 2021: minimize cycles

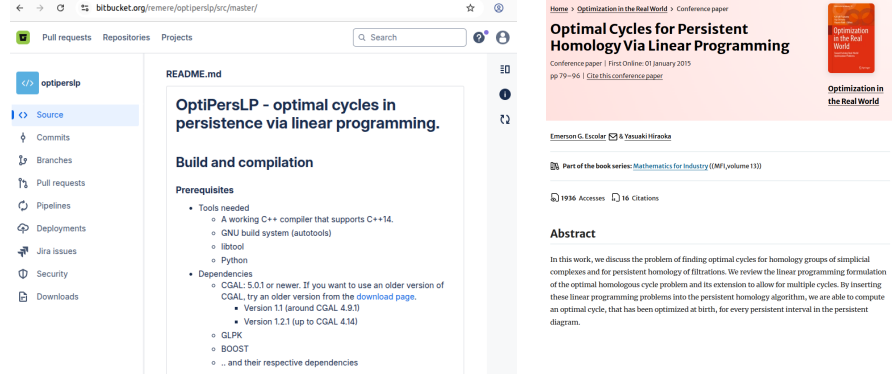
TTK, J. Tierny, ... 2020: tracking critical pts



OAT, G. Henselman-Petrusek Roak, ..., 2024: access, minimize cycles



OptPersLP, E. Escolar, Y. Hiraoka 2015: minimize cycles



Open questions / tasks:

1. *(task)* Implement a graphical user interface (point and click) to visualize and modify a dataset and its PH information.
2. *(exploratory + task)* What properties (oracle answers) should a dynamic PH data structure have, to accomocate all possible dynamic changes? Implement it.
3. *(inverse problem)* Given a choice of cycle representatives, does there exist a barcode algorithm that produces them?
   (a) *(exploratory)* What is the set / space of barcode algorithms? Can they be listed / sampled from?

Thanks for your attention!

Banff International Research Station
for Mathematical Innovation and Discovery