

Comparative Analysis of Topological Structures

A THESIS
SUBMITTED FOR THE DEGREE OF
Doctor of Philosophy
IN THE
Faculty of Engineering

BY
Raghavendra G S



Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

November, 2022

Declaration of Originality

I, **Raghavendra G S**, with SR No. **04-04-00-10-12-16-1-13915** hereby declare that the material presented in the thesis titled

Comparative Analysis of Topological Structures

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2016-2022**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: 25/11/2022



Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Vijay Natarajan



Advisor Signature

© Raghavendra G S
November, 2022
All rights reserved

DEDICATED TO

My Family

Acknowledgements

ऊर्ध्वोर्ध्वमारुह्य यदर्थतत्त्वं धीः पश्यति श्रान्तिमवेदयन्ती ।
फलं तदाद्यैः परिकल्पितानां विवेकसोपानपरम्परानाम् ॥

-Abhinavabhāraṭī ¹

We climb higher and higher, without exhaustion, gaining clarity of the ultimate truth, this is indeed the fruit of the stairs of discretion built by our predecessors.

As I look back at the past years I've spent during my PhD, I remember the import of the verse quoted above - nothing can happen in isolation. So on this note I'd like to express my gratitude towards everyone who has helped me during this journey.

I'd like to thank my advisor Prof. Vijay Natarajan. He has done way beyond what an advisor would typically do for an advisee. He not only introduced me to the fields of Scientific Visualization and Computational Topology, but also has helped me to develop my research capabilities, presentation and writing skills, with his guidance, support, and pertinent feedback. He has also handled pesky logistics, funding and many other things with equanimity. He always encouraged collaborations which have been extremely fruitful for me in every possible way. Thanks Vijay for everything you have done for me!

I'd like to thank Profs. Rajesh Sudaresan, Ramesh Hariharan, Subhojay Gupta, Pooja Singla, Anand Louis, Satish Govindarajan and Vijay Natarajan for teaching some wonderful courses which formed my RTP. Though my performances weren't the best, I've learnt a lot from these courses.

I'd like to thank the Dept. heads during my PhD, Profs. Jayanth Haritsa, Shalabh Bhatnagar and Chiranjib Bhattacharya for their support. I'd also like to thank Profs. Narahari, Matthew Jacob, Satish Govindarajan, Deepak D'souza for their constant encouragement and Vinod Ganapathy, Siddharth Barman, Rahul Saladi, Chandan Saha, Gugan Thoppe, Sunil Chandran for cordial interactions whenever I've met them.

¹Commentary of Nāṭyaśāstra by Mahāmāheśvara Abhinavagupta (950-1016 CE), philosopher, aesthetician and polymath.

Acknowledgements

I'd like to acknowledge Prof. Satish Vadhyar, the PI of the project which supported me for the previous year.

The CSA office staff, Ms. Padmavathi, Ms. Nishitha, Ms. Kushael, Ms. Meenakshi have helped me in all office related work and the admins, Mr. Pushparaj and Mr. Akshaynath have been helpful in things related to systems and the lab.

I've been fortunate enough to work with wonderful colleagues. In no particular order, I'd like to thank, Talha bin Masood, Adhitya Kamakshidasan, Somenath Das, Lin Yan, Farhan Rasheed, Ingrid Hotz and Bei Wang for fruitful collaborations. Thanks to Talha, Harish, Nithin, Karran, Abhijath, Somenath and Adhitya for either providing their implementations/scripts or helping me in writing some. I'd like to thank Dr. Srisha Rao from Aerospace dept. for his help in interpreting some of the results.

VGL has never been a dull place thanks to the wonderful folks who have been part of the lab. I'd like to thank Akash, Talha, Nagarjun, Karran, Varshini, Mohit, Somenath, Adhitya, Giri, Dhruv, Dinesh, Dhurjati, Meenaly, Abhishek, Amritendu, Upkar, Harish, and Nithin. Thank you all for the innumerable trips to Prakruti/Sarvam for coffee/tea, sharing the TA load, various trips, movies, and discussions.

The VIS community has been extremely welcoming and friendly, to a newbie like me. I'd like to thank them for all the encouragement. I'd like to thank Profs. Julien Tierny, Ingrid Hotz and Bei Wang in particular for their help. I'd also like to thank Prof. Tino Weinkauff for providing some of the data used in our experiments. During an extremely uncertain phase before my PhD, Prof. Jaya Nair hosted me for a year, I'd like to thank her for everything she has done for me.

Outside research, I've been fortunate to have a lot of good friends, \mathbb{R}^4 and *The Kumaars* have been with me through thick and thin.

I'd like to acknowledge various study groups where we regularly read classical literature in Sanskrit, Kannada, and English, which I've been part of. These sessions have been the ultimate stress-buster. I'd also like to thank Dr. R Ganesh for his insightful expositions of various literary works and extempore compositions of refreshingly original poetry. I'd like to thank Shri Ramachandra K B S and his family for hosting us and for being of great help to me. The two great epics of India, the Ramayana and the Mahabharata along with the works of Kalidasa, have been dear to my heart and have been instrumental in shaping my perspective about life. They have been the support in hard times and also have kept me firmly on the ground whenever there is the danger of getting carried away and taking myself more seriously than required.

I'd like to thank my relatives Shri. H C Nagendra, Shri. H C Shashidhar, Shri. Chandrakanth,

Acknowledgements

Shri. G S Ramakrishna, Shri. Ramashastry and their families, and Karthik Muralidharan for their unwavering support in difficult times. I'd like to thank my in-laws Prof. Jayadeva Bhat, Dr. Vani and Harsha for the trust they have placed in me and for all their support. I'd also like to thank Shri. Ramakrishna Hirisave for his encouragement.

In our culture, the debt we owe to our parents can never be transcended, it can only be acknowledged. I'd like to thank my parents, Usha and Sridharamurthy for never questioning my choices and giving me absolute freedom to do whatever I want, even when some of my choices didn't always result in favourable outcomes.

It takes one PhD student to understand another! I've been lucky that I've Chandralekha as my wife. Her constant and unwavering support has been crucial to me during my tenure as a PhD student. Her support has made this look like a walk in the park. Thank you for everything!

Finally, I'd like to acknowledge the support of various funding agencies,

1. Scholarship from MHRD.
2. Swarnajayanti Fellowship from the Department of Science and Technology, India (DST/SJF/ETA-02/2015-16), Govt. of India.
3. Mindtree Chair research grant.
4. Joint Advanced Technology Programme, Indian Institute of Science (JATP/RG/PROJ/2015/16).
5. Robert Bosch Centre for Cyber Physical Systems.
6. Tata Trust travel grant.
7. ACM IARCS travel grant.

Abstract

Measuring scientific processes result in a set of scalar functions (scalar fields) which may be related temporally, be part of an ensemble, or unrelated. Overall understanding and visualization of scientific processes require the study of individual fields and, more importantly, the development of methods to compare them meaningfully. In this thesis, we focus on the problem of designing meaningful measures to compare scalar fields by comparing their abstract representations called topological structures. We emphasize on intuitive and practical measures with useful properties and applications.

The first part of the thesis deals with comparing a topological structure called the merge tree. We propose two global comparison measures, both based on tree edit distances. The first measure OTED is based on the assumption that merge trees are ordered rooted trees. Upon finding that there is no meaningful way of imposing such an order, we propose a second measure called MTED for comparing unordered rooted trees. We propose intuitive cost models and prove that MTED is a metric. We also provide various applications such as shape comparison, periodicity detection, symmetry detection, temporal summarization, and an analysis of the effects of sub-sampling /smoothing on the topology of the scalar field.

The second part deals with a local comparison measure LMTED for merge trees that supports the comparison of substructures of scalar fields, thus facilitating hierarchical or multi-scale analysis and alleviating some drawbacks of MTED. We propose a dynamic programming algorithm, prove that LMTED is a metric and also provide applications such as symmetry detection in multiple scales, a finer level analysis of sub-sampling effects, an analysis of the effects of topological compression, and feature tracking in time-varying fields.

The third part of the thesis deals with comparison of a topological structure called the extremum graph. We provide two comparison measures for extremum graphs based on persistence distortion (PDEG) and Gromov-Wasserstein distance (GWEG). Both persistence distortion and Wasserstein distance are known metrics. We analyze how the underlying

Abstract

metric affects these comparison measures and present various applications such as periodicity detection to facilitate scientific data analysis and visualization.

The final part of the thesis introduces a time-varying version of extremum graphs (TVEG) with a simple comparison criterion to identify correspondences between features in successive time steps. We provide applications to tracking features in time-varying scalar fields from computational fluid dynamics.

Publications based on this Thesis

1. Raghavendra Sridharamurthy, Adhitya Kamakshidasan and Vijay Natarajan. “Edit Distances for Comparing Merge Trees”, In *Proc. IEEE Conference on Visualization (Posters)*, 2017.
Received Best Poster Award.
2. Raghavendra Sridharamurthy, Talha Bin Masood, Adhitya Kamakshidasan and Vijay Natarajan. “Edit Distance between Merge Trees.”, *IEEE Transactions in Visualization and Computer Graphics*, March 2020, 26(3), 1518–1531, DOI: <https://doi.org/10.1109/tvcg.2018.2873612>.
3. Lin Yan, Talha Bin Masood, Raghavendra Sridharamurthy, Farhan Rasheed, Vijay Natarajan, Ingrid Hotz and Bei Wang. “Scalar Field Comparison with Topological Descriptors: Properties and Applications for Scientific Visualization”, *Computer Graphics Forum (EuroVis STAR 2021)*, 2021, 40(3), 599–633, DOI: <https://doi.org/10.1111/cgf.14331>.
4. Raghavendra Sridharamurthy and Vijay Natarajan. “Comparative Analysis of Merge Trees using Local Tree Edit Distance”, *IEEE Transactions in Visualization and Computer Graphics*, 2022, *Preprint*, 1–13, DOI: <https://doi.org/10.1109/tvcg.2021.3122176>.
5. Somenath Das, Raghavendra Sridharamurthy and Vijay Natarajan. “Time-varying Extremum Graphs”, *Computer Graphics Forum*, 2022, *Under review*.
6. Raghavendra Sridharamurthy and Vijay Natarajan. “Comparing Extremum Graphs”, *In Preparation*.

Contents

- Acknowledgements i
- Abstract iv
- Publications based on this Thesis vi
- Contents vii
- List of Figures xii
- List of Tables xv

- 1 Introduction 1**
 - 1.1 Comparison pipeline 2
 - 1.2 Global comparison of merge trees 3
 - 1.2.1 Ordered merge tree edit distance OTED 3
 - 1.2.2 Unordered merge tree edit distance MTED 4
 - 1.3 Local comparison of merge trees LMTED 4
 - 1.4 Comparison of extremum graphs 5
 - 1.5 Time-varying extremum graphs TVEG 6
 - 1.6 Organization 6

- 2 Related Work 7**
 - 2.1 Topological structures and their construction 7
 - 2.2 Comparative measures for scalar fields 8
 - 2.2.1 Comparative measures for Isosurfaces 8
 - 2.2.2 Comparative measures for Set-based structures 9
 - 2.2.3 Comparative measures for Graph-based structures 9

2.2.4	Comparative measures for Complex-based structures	10
2.2.5	Local comparison measures	11
2.3	Feature tracking and time-varying topological structures	11
2.3.1	Tracking based on topology of isosurfaces	11
2.3.2	Tracking based on topological structures	11
2.3.3	Time-varying topological structures	12
2.4	Tree edit distance	13
2.5	Comparison of metric graphs	13
2.5.1	Persistence Distortion for metric graphs	13
2.5.2	Gromov-Hausdorff and Gromov-Wasserstein distances	14
2.6	Summary	14
3	Background	16
3.1	Scalar field and Morse theory	16
3.1.1	Morse function	17
3.1.2	Morse theory	18
3.2	Topological structures	18
3.2.1	Set-based topological structures	18
3.2.1.1	Persistence diagram and barcode	19
3.2.2	Graph-based topological structures	20
3.2.2.1	Merge tree	21
3.2.2.2	Reeb graph and contour tree	22
3.2.3	Complex-based topological structures	24
3.2.3.1	Morse and Morse-Smale complex	24
3.2.3.2	Extremum graph	24
3.3	Topological simplification	25
3.4	Metric graphs	26
3.5	Properties of comparison measures	26
3.6	Comparison measures	28
3.6.1	Bottleneck and Wasserstein distances	28
3.6.2	Persistence distortion distance	29
3.6.3	Gromov-Hausdorff and Gromov-Wasserstein distances	29
3.7	Tree edit distance	30
3.7.1	Edit distance mappings	32
3.7.1.1	Unconstrained edit distance mappings	32

3.7.1.2	Constrained and restricted mappings	33
3.7.1.3	Illustrations of the mappings	34
3.7.2	Constrained edit distance	34
3.7.3	Algorithm	36
3.8	Tree edit distance with general gap model	36
3.8.1	Distance measure	37
3.8.2	Algorithm	37
3.9	Summary	38
4	Global edit distance for merge trees (OTED) and (MTED)	39
4.1	Introduction	39
4.2	Contributions	40
4.2.1	OTED	40
4.2.2	MTED	40
4.3	Global merge tree edit distances	41
4.3.1	Ordered merge tree edit distance OTED	41
4.3.1.1	Definition	41
4.3.1.2	Algorithm	42
4.3.2	Unordered merge tree edit distance MTED	42
4.3.2.1	Comparing merge trees	43
4.3.2.2	Cost models	44
4.3.2.3	Properties	46
4.3.2.4	Handling instabilities	49
4.3.2.5	Algorithm	50
4.4	Applications	50
4.4.1	Applications of OTED	52
4.4.1.1	Periodicity in time-varying data	52
4.4.1.2	Symmetry detection	52
4.4.2	Applications of MTED	53
4.4.2.1	Understanding the distance measure	53
4.4.2.2	Comparison with other distance measures	55
4.4.2.3	Periodicity in time-varying data	57
4.4.2.4	Topological effects of subsampling and smoothing	60
4.4.2.5	Detecting symmetry / asymmetry	60
4.4.2.6	Shape matching	61

4.4.2.7	Data Summarization	64
4.5	Further developments	65
4.6	Summary	67
5	Local edit distance for merge trees (LMTED)	70
5.1	Introduction	70
5.2	Contributions	70
5.3	Local merge tree edit distance LMTED	72
5.3.1	Truncated persistence and truncated costs	72
5.3.2	Definition	73
5.3.3	Cost models and properties	76
5.3.4	Algorithm and computation	78
5.3.4.1	Dynamic Programming tables	78
5.3.4.2	Pseudocode and analysis	80
5.3.5	Refinement and optimization	84
5.3.5.1	Ordering subtrees	84
5.3.5.2	Comparison refinement	84
5.3.5.3	Subtree refinement	85
5.4	Applications	86
5.4.1	Understanding the local tree edit distance	86
5.4.2	Symmetry Detection	87
5.4.3	Analysis of subsampling, smoothing, and topology based compression	95
5.4.4	Spatio-temporal exploration and feature tracking	99
5.5	Summary	102
6	Comparing extremum graphs (PDEG) and (GWEG)	103
6.1	Introduction	103
6.2	Contributions and challenges	104
6.2.1	Contributions	104
6.2.2	Challenges	104
6.2.2.1	Common challenges	104
6.2.2.2	Challenges specific to persistence distortion distance	106
6.2.2.3	Challenges specific to Gromov-Wasserstein distance	106
6.3	The persistence distortion (PDEG) and Gromov-Wasserstein (GWEG) distances for extremum graphs	107

CONTENTS

6.3.1	Properties	108
6.3.1.1	Properties of PDEG	109
6.3.1.2	Properties of GWEG	109
6.3.2	Algorithm and computation	110
6.3.2.1	Computation of PDEG	110
6.3.2.2	Computation of GWEG	110
6.3.3	Discussion	111
6.4	Applications	111
6.4.1	Understanding the comparison measures	113
6.4.2	Periodicity detection	115
6.4.3	Comparing pore networks	116
6.5	Summary	119
7	Time Varying Extremum Graphs (TVEG)	121
7.1	Introduction	121
7.2	Contributions	121
7.3	Time varying extremum graphs TVEG	122
7.3.1	Temporal correspondences	123
7.3.2	Algorithm	125
7.4	Applications	128
7.4.1	Moving Gaussians	128
7.4.2	Vortex Street	135
7.5	Summary	138
8	Conclusions and future work	139
8.1	Impact	139
8.2	Future work	139
8.2.1	Merge tree measures	139
8.2.2	Extremum graph measures	140
8.2.3	General directions	140
	Bibliography	141

List of Figures

1.1	The comparison pipeline	3
3.1	Morse functions	17
3.2	Persistence diagram	19
3.3	Merge tree	21
3.4	Persistence pairs in merge trees	22
3.5	Discriminative power of merge trees	22
3.6	Reeb graph	23
3.7	Contour tree	23
3.8	Morse-Smale complex and extremum graph	25
3.9	Tree edit operations	31
3.10	One-to-one mapping	34
3.11	Ancestor preserving mapping	34
3.12	Disjoint subtree preserving mapping	35
4.1	OTED cost model	42
4.2	Valid edit operations	44
4.3	Cost models for MTED	46
4.4	Edit cost bipartite graph	47
4.5	Triangle inequality proof for the cost model	48
4.6	Illustrating instabilities	49
4.7	Periodicity detection using OTED	52
4.8	Symmetry detection using OTED	53
4.9	Understanding MTED	54
4.10	Periodicity detection and stability analysis of MTED	56
4.11	Periodicity detection in 2D vortex street	57
4.12	Periodicity matrix in full resolution	58

LIST OF FIGURES

4.13	Measuring the effects of subsampling and smoothing using MTED	59
4.14	Symmetry detection in toy data using MTED	62
4.15	Symmetry detection in EMDB data using MTED	63
4.16	Shapes from the TOSCA non-rigid world dataset	64
4.17	Detecting shape similarities using MTED	65
4.18	Top-4 queries in shape similarities	66
4.19	The 3D Bénard-von Kármán vortex street dataset	67
4.20	Data summarization of 3D vortex street using MTED	68
5.1	Illustrating globally dissimilar but locally similar scalar fields	71
5.2	Split trees corresponding to Figure 5.1	72
5.3	Illustrating LMTED calculation	75
5.4	Case analysis of LMTED	79
5.5	Dynamic programming table for MTED	80
5.6	The modified DP table contributing to LMTED	80
5.7	Highlighted entries required to calculate LMTED in original table	81
5.8	Highlighted entries required to calculate LMTED in modified table	81
5.9	Understanding LMTED	86
5.10	Distance matrix for EMDB 1654	87
5.11	Symmetric regions in EMDB 1654	89
5.12	Symmetric regions in EMDB 1654 in smaller scales	90
5.13	Distance matrix for EMDB 1603	91
5.14	Symmetric regions in EMDB 1603	92
5.15	Distance matrix of EMDB 1897 with reordering	93
5.16	Symmetric regions in EMDB 1897	94
5.17	Effect of subsampling and smoothing	96
5.18	Measuring the effect of subsampling using LMTED	97
5.19	Distance matrix to analyse topological effects of compression	98
5.20	Visualizing the effect of different compression thresholds	100
5.21	Visualizing the top tracks in the 3D vortex street using LMTED	100
5.22	Query based tracking using LMTED	101
6.1	Choice of extremum graphs	105
6.2	Computing PDEG	107
6.3	Computing GWEG	108
6.4	PDEG for 2D vortex street time steps 0, 8	112

LIST OF FIGURES

6.5	GWEG for 2D vortex street time steps 0,8	114
6.6	Distance matrices for GWEG	115
6.7	Distance matrices for PDEG	116
6.8	Periodicity detection using GWEG and PDEG	117
6.9	Similarity in zeolite structures	120
7.1	TVEG of a 2D time-varying scalar field	122
7.2	Illustrating TVEG	123
7.3	<i>Gauss8</i> dataset	133
7.4	Temporal tracks from the TVEG of <i>Gauss8</i>	133
7.5	Comparison of tracks obtained by TVEG and LWM	135
7.6	Visualizing the top 20% tracks in the 3D von Kármán vortex street data	135
7.7	Query based tracking using TVEG	135
7.8	Illustrating challenges in tracking vortices in the 3D vortex street	136

List of Tables

- 2.1 Summary of related work 15
- 2.2 Summary of contributions 15
- 7.1 List of different attributes of the critical points. 127

Chapter 1

Introduction

This thesis deals with comparative analysis of topological structures to facilitate meaningful comparison of the underlying scalar functions. Scalar functions or fields are omnipresent when it comes to measuring and simulating scientific phenomena. Analysis and visualization of these scalar fields lead to a better understanding of the processes involved. Direct analysis and visualization of such a scalar function using standard visualization techniques like isosurface construction or volume rendering provides a good overview but is limited by two factors. First, increasing size of data makes storage and retrieval inefficient. Second, the analysis often requires a sweep over a large subset of the domain or range of the function even when the features of interest may be contained within a small region. These limitations are amplified when we consider time-varying scalar functions. Thus, these techniques are not well suited for feature directed analysis and visualization. This has necessitated the need for a summarized representation. This is and has been addressed by topological data structures. Topological structures provide us with an abstract representation of the scalar field, thus reducing the complexity while retaining the salient features. These structures have been successfully used in the recent years to represent scalar fields in a meaningful and succinct manner. Topological structures have facilitated a feature-aware, interactive and exploratory analysis and visualization resulting in better understanding of these phenomena. Various topological data structures like persistence diagrams [42], merge trees [22], contour trees [22], Reeb graphs [95], extremum graphs [30], Morse-Smale complexes [44, 43] have been defined in the literature which provide meaningful combinatorial abstractions of the scalar field while capturing different aspects of the field. The properties of these structures have also been well-studied along with efficient algorithms to compute them.

Many of these scientific phenomena may vary with time resulting in a set of scalar fields that are related temporally. They may vary based on different parameters result-

ing in ensembles. They can also be just a set of scalar fields which are neither related temporally nor dependent on any parameters but still needs to be compared to understand the phenomena. Overall understanding may mandate not just studying the individual fields but also methods to study them together in a global context and compare them meaningfully. Such methods have been found to be useful for tracking features in time-varying phenomena [101], topological shape matching [62], detecting symmetry/asymmetry in scalar fields [124, 125, 76, 102, 126], or clustering [89], computing temporal summaries of large datasets, to identify features that are preserved in ensemble simulations or multi-field data, or to compare simulated data against measured data [104, 85, 33, 38]. This thesis deals with such methods facilitating meaningful representations of time-varying fields and also measures comparing topological structures which represent the scalar fields.

1.1 Comparison pipeline

The comparison pipeline (Figure 1.1) starts with scalar fields - individual, time-varying or ensembles - constructs topological structures as their abstract representations, compares them using a comparison measure customised for such comparison or the application. Finally the pipeline provides an output that captures the similarity/dissimilarity of the fields. This can be used further to gain insights about the underlying scalar fields and drive applications mentioned in the previous section.

We illustrate the comparison pipeline with an example. Symmetry detection in bio-molecules is an important application which helps the biologists to understand the structure and function of the molecules. The measurements are typically done by Cryo-electron microscopy (Cryo-EM) which provides electron density - a scalar field - corresponding to the bio-molecule. In this case since the biologists are interested in similar sub-structures present within the same molecule, the field should be compared to itself. The field will contain noise and also point-wise comparison won't lead to meaningful comparisons of the high level structures which the biologists would be interested in. To remove noise and also get an abstraction which captures such structures we build a topological structure like merge tree out of the scalar field. We then compare the merge tree to itself using some local comparison measure to build a distance matrix. The sub-matrices provides us the symmetric structures in multiple scales as shown in the Figure 1.1.

In the sections that follow, we give an overview of our contribution to facilitate comparison - local and global - of various topological structures.

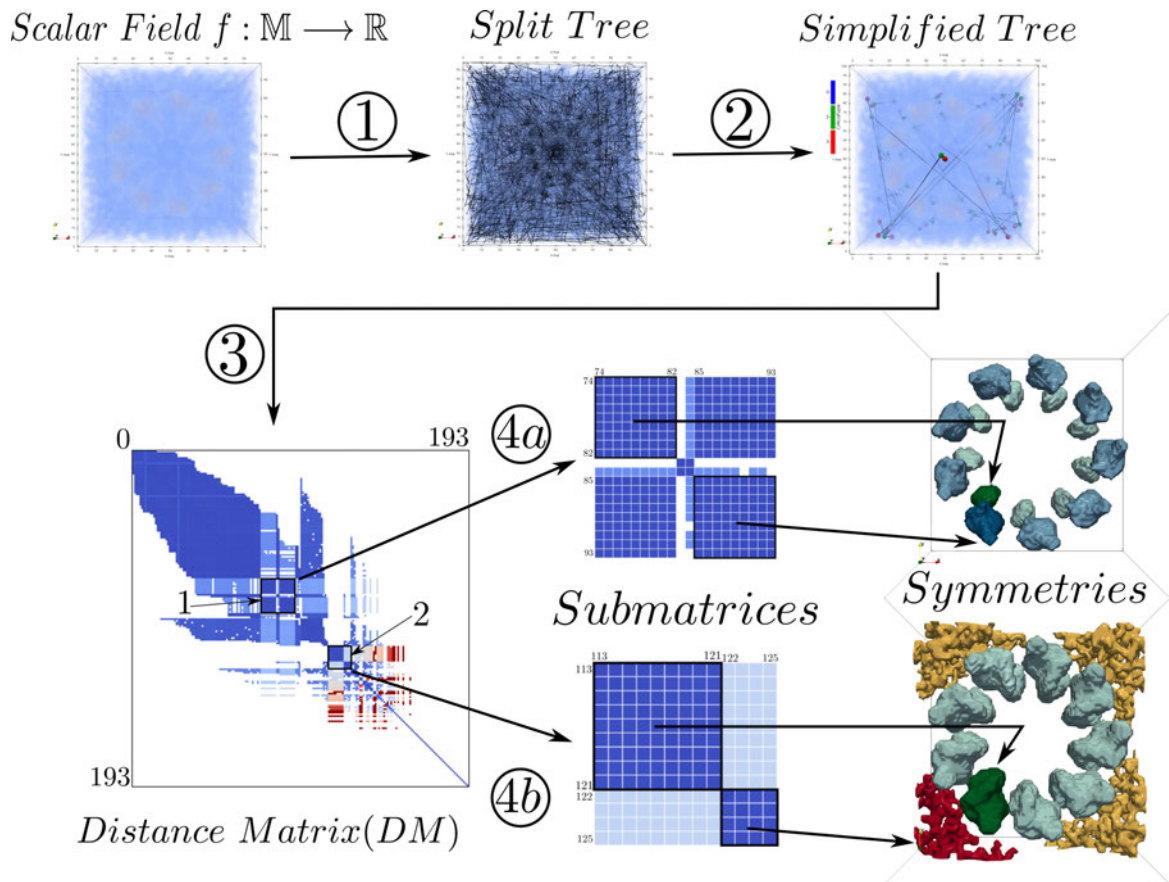


Figure 1.1: Illustrating a typical comparison pipeline. The pipeline takes a scalar field as the input, in step 1 it computes the topological structure, a split tree, in step 2 the tree is simplified by removing topological noise, in step 3 the simplified tree is compared to itself to generate a distance matrix DM, in step 4 the sub-matrices correspond to the symmetric regions which can be used to extract them as shown in the bottom right.

1.2 Global comparison of merge trees

We start with global comparison of merge trees. Merge trees capture the topology of the sublevel/superlevel sets to provide a succinct representation of the scalar fields.

1.2.1 Ordered merge tree edit distance OTED

We adapt the ordered tree edit distance from Xu [139] assuming merge trees to be ordered rooted binary trees to design OTED [119]. The important contributions include,

- A simple comparison measure for merge trees.
- A simple cost model based on topological persistence.

- Applications to periodicity detection and symmetry detection in 2D scalar fields.

1.2.2 Unordered merge tree edit distance MTED

Continuing the work on merge trees, we observed that merge trees are unordered rooted trees and there is no meaningful ordering which can be imposed upon them. The merge trees are also not necessarily binary trees. The algorithm used in OTED [119] has a slow running time of $O(n^5)$ where n is the size of the trees. To alleviate these shortcomings we adapt the constrained tree edit distance for unordered rooted trees by Zhang [144] to merge trees with significant modifications and propose MTED [120]. The important contributions include,

- An intuitive and mathematically sound cost model for the individual edit operations.
- A proof that the comparison measure is a metric under the proposed cost model.
- A computational solution to handle instabilities.
- Experiments to demonstrate the utility of the distance measure
 - Periodicity detection in 2D time-varying data
 - Temporal summarization to support visualization of 3D time-varying data
 - Symmetry detection in scalar fields
 - Study of topological effects of subsampling and smoothing
 - Shape matching.

We also describe a comprehensive set of validation experiments that are designed to help understand the properties of the measure.

1.3 Local comparison of merge trees LMTED

While global comparison measures like OTED and MTED help to address variety of interesting problems, they do not support fine-grained and hierarchical analysis of local structures or substructures of scalar fields. We adapt MTED [120] with significant modifications to propose a local comparison measure called LMTED [117]. The important contributions include,

- A novel local tree edit distance (LMTED) to compare substructures in scalar fields.
- A proof that it satisfies metric properties.

- A dynamic programming algorithm to compute the L_{MTED} efficiently.
- A notion of truncated persistence to compute costs of matching / correspondences, which brings in the additional benefit of saving computation time by reducing the number of comparisons.
- Experiments to demonstrate the practical value of the distance
 - Symmetry detection at multiple scales
 - Analysis of the effects of smoothing and subsampling
 - Fine grained analysis of topological compression [115]
 - Applications to feature tracking

Note that the $MTED$ supports only a few of the above-mentioned applications in a more restricted level with higher level of user intervention. Feature tracking is not possible with $MTED$ without significant modifications.

1.4 Comparison of extremum graphs

While merge trees provide a good representation of the topology, they need to be explicitly augmented if there is a need to incorporate geometric information. Morse-Smale complex while capturing everything takes a lot of space, storing higher dimensional cells and it also lacks stability. Extremum graph serves as a via media by storing some geometric information but takes less space as it only stores nodes and arcs.

We present Persistence distortion distance (PDEG) and Gromov-Wasserstein distance (GWEG) for extremum graphs by adapting persistence distortion distance defined by Dey et.al [34] for metric graphs and Gromov-Wasserstein distance defined by Memoli [81] to the case of extremum graphs [118]. The measures are metrics. Important contributions include

- Adaptation of the distances to extremum graphs.
- Guidelines for the choice of metrics through experimental analysis.
- Experiments to showcase the utility of the distances.
 - Intuitive interpretation.
 - Periodicity detection.
 - Comparison of pore networks.

1.5 Time-varying extremum graphs TVEG

We also propose a time-varying extremum graphs TVEG which is an extension of extremum graph [30] to facilitate analysis and visualization of scalar fields [31]. The important contributions include,

- A definition of a novel topological structure, the time-varying extremum graph (TVEG).
- An algorithm for constructing the TVEG based on a formulation as an optimization problem.
- Application in feature tracking.
 - 3D vortex street data.

1.6 Organization

This thesis is organized as follows. The Chapter 2 surveys previous work done in this area. The Chapter 3 deals with the necessary mathematical background. The Chapters 4-7 comprises of the contributions of the thesis. Specifically, in Chapter 4 provides two methods to compute global merge tree edit distances OTED and MTED. Chapter 5 covers the local comparison of merge trees LMTED. Chapter 6 provides a method to compare extremum graphs using persistence distortion PDEG and Gromov-Wasserstein distance GWEG. Chapter 7 discusses a time-varying version of extremum graph TVEG. Finally, Chapter 8 concludes the thesis.

Chapter 2

Related Work

Topological structures in the past two or three decades have been found to be extremely useful in data analysis and visualization and has resulted in a lot of exciting work both in theory and practice. Various topological descriptors have been defined, their properties have been thoroughly studied, multiple applications have been showcased. Various strategies to compute them efficiently both in terms of space and time complexity have been developed including parallel and distributed algorithms. There have also been many comparison measures defined for these structures and have been applied to solve various problems. In this chapter we discuss all the related work and position our work w.r.t these previous results. Some of the material covered in this chapter is based on the *State of The Art Report (STAR)* on Comparative measures by Yan et.al. [141]. Relevant portions also come from the related work sections of [119, 120, 117, 118, 31]

Topological structures have been classified [141] into set-based structures (persistence diagrams), graph-based structures (Reeb graphs, merge/contour trees) and complex-based structures (Morse/Morse-Smale complexes).

2.1 Topological structures and their construction

All topological descriptors are defined based on critical points of the scalar field and their relation. Banchoff [5] provides a strategy to compute critical points for piece-wise linear (PL) functions. Edelsbrunner et.al. [39] first defined persistence diagrams. Reeb [95] first defined Reeb graph as the quotient space based on the equivalence relation between points belonging to the same component of level sets. Contour tree is the same as the Reeb graph for domains that are simply connected. A related topological structure is the merge tree which track the connectivity of sublevel/superlevel sets. Efficient algorithms

exist to compute Reeb graphs, contour/merge trees [1, 22, 37, 84]. Morse and Morse-Smale complexes [44, 43] on the other hand are defined based on the gradient of the scalar field. There are efficient algorithms to compute them [58, 112, 110, 61]. Correa et.al. [30] defined the extremum graph which is a substructure of Morse-Smale complex and also provided an algorithm to compute it.

There are a number of tools available for computing these topological structures, some as standalone softwares, others as web-based interactive applications (see [141]). Some of the tools for computing persistence diagrams and/or persistence barcodes, together with their bottleneck and Wasserstein distances, include GUDHI [54], PHAT [8], Dionysus [83], R-TDA [46], HERA [63], persim [92], Perseus [86], and Ripser [6]. They also support multiple use-cases which aid in machine learning tasks. Many of these tools are implemented in C/C++ or Python, while a few provide Python/R wrappers. Tools to compute merge trees, contour trees and Reeb graphs along with support for related tasks like computing branch decomposition representations, symmetry detection, and feature tracking are available in software such as Recon [36], contour-tree [36], mtlib [100], AMT [142], and SymmetryViewer [123]. These software are implemented in C/C++, Python, or Java. Tools for computing Morse-Smale complexes alike include mscomplex3d [111], MSCEER [55], CompExtGraph [87]. Topology ToolKit (TTK) [129] has been a popular toolkit designed to work together with the visualization software ParaView [3], with support for the computation and visualization of persistence diagrams, merge trees, contour trees, Reeb graphs, and Morse-Smale complexes, together with persistence based simplifications of these descriptors. It also allows computation of bottleneck/Wasserstein distances between persistence diagrams and feature tracking via nested tracking graphs.

2.2 Comparative measures for scalar fields

Comparison of scalar fields can be facilitated by methods which compare the fields directly, compare isosurfaces or topological structures. In this section we review previous results which deal with comparison of scalar fields.

2.2.1 Comparative measures for Isosurfaces

Assuming identical domains, RMS distance, Chebyshev distance, and other norms such as $L_p, 1 \leq p \leq \infty$ can be used for point-to-point comparisons. However, a direct comparison of the two scalar functions may not be appropriate because of its sensitivity to noise, transformations, and minor perturbations.

Scalar fields can be compared based on their isosurfaces which are surfaces with the

same scalar value. Since theoretically, the number of isosurfaces is infinite, the comparison requires a selection of a finite number of isosurfaces of interest. Tao et al. [121] extend the notion of isosurface similarity maps, first conceptualized by Bruckner and Möller [18], to construct matrices of isosurface similarity maps (MISM), and use it to explore multivariate time-varying data. This involves construction of self-similarity maps, temporal similarity maps, and variable similarity maps followed by temporal clustering and variable grouping. Finally, paths spanning across these maps are used to guide the visual comparison. The choice of the isovalues used is crucial but the inclusion hierarchy followed by isosurfaces is not utilised.

2.2.2 Comparative measures for Set-based structures

Comparison measures between various topological structures have been studied in the literature, beginning with the bottleneck distance between persistence diagrams [27]. A persistence diagram depicts the persistence or “lifetime” of all topological features by plotting their corresponding time of creation (birth) and destruction (death) as points in \mathbb{R}^2 . The bottleneck distance (d_B) between two persistence diagrams is equal to the weight of the minimum weight mapping between points of the two diagrams. The weight of a mapping is equal to the largest L_∞ distance between a point and its image under the mapping. The p -Wasserstein distance d_p which is a generalization of d_B is an extended pseudometric; it is a metric [40, Page 184] when the persistence diagrams are locally finite.

We say that the persistence diagram is *stable* with respect to a distance measure if it is bounded above by the L_∞ distance between the two scalar functions. Intuitively, we require that small perturbations to the scalar functions translate to small changes in the distance between the respective persistence diagrams. The persistence diagram is stable with respect to d_B and also with respect to d_p for a reasonably large class of functions [28]. But, the persistence diagram is only a multiset. It does not capture the spatial configuration of critical points, which reduces its discriminative capability.

For other comparison measures for persistence diagram and its variants, refer the Survey by Yan et.al. [141]

2.2.3 Comparative measures for Graph-based structures

Comparative measures for graph-based structures constitute the bulk of measures defined in the recent years.

We start with describing some of the measures related to the merge tree. Morozov et al. [84] proposed the interleaving distance between merge trees. This distance is defined

by a continuous map that shifts points of one merge tree onto the other and vice-versa. The distance is equal to the smallest value of the shift such that the map satisfies certain compatibility conditions. The merge tree is stable under this distance and the distance measure is more discriminative compared to the bottleneck distance but computing it is not a tractable problem. Beketayev et al. [10] define a distance measure between merge trees that can be computed by considering all possible branch decompositions. This measure can be computed in polynomial time but provide no guarantees on stability.

Various comparison measures for Reeb graph have been defined. Bauer et al. [7] imposed a metric on Reeb graphs called the functional distortion distance. They proved its stability and connections with other distances such as the bottleneck and interleaving distance. The computation depends on the Gromov-Hausdorff distance, which is proven to be NP-hard [2] to even approximate up to a constant factor for general metric graphs. Di Fabio and Landi [35] defined an edit distance for Reeb graphs on surfaces. They also proved its stability and showed connections with interleaving distance and function distortion distance but there is no polynomial time algorithm to compute the distance.

In contrast to the rigorous definitions of comparison measures introduced in the above-mentioned works, simpler but practical similarity measures have also been studied. Saikia et al. [102] introduced the extended branch decomposition graph (eBDG) that describes a hierarchical representation of all subtrees of a join/split tree and designed an efficient algorithm to compare them. They also present experimental results on time-varying data. Saikia et al. [103] studied a measure that compared histograms that are constructed together with the merge trees. As in the case of bottleneck distance, this measure ignores the structure but it can be computed efficiently and is therefore useful in practice. Saikia and Weinkauff [101] later extended this measure and demonstrated applications to feature tracking in time-varying data.

2.2.4 Comparative measures for Complex-based structures

Comparison measures for Complex-based structures are few in number, mainly because of the complexity of the structures and their vulnerability to noise. A distance measure to compare *extremum graphs* is defined by Narayanan et al. [88]. The distance measure is based on the maximum weight common sub-graph and they use pruning techniques to speedup the computation. While stability is not guaranteed, they present experimental results on time-varying data to demonstrate its application to time-varying data analysis and visualization.

2.2.5 Local comparison measures

Most of the comparison measures are global in the sense that they do not compare sub-structures of the descriptor. Some of the above mentioned methods like Saikia et al. [103] do involve local comparison but they are limited. Saikia et al. [102] compare all subtrees of a merge tree, they demonstrate its use in applications by explicitly choosing or selecting region(s) of interest rather than considering the collection of all pairs of subtrees. Lukasczyk et al. [72, 73] do facilitate tracking features in all scales and across time, but does not support generic comparison between local features which are not part of time-varying scalar fields. Symmetry detection has been well studied by Thomas and Natarajan [124, 125, 126], but the methods have not been applied to detect local similarities between different scalar fields in general. In other words, the symmetry detection problem is a special case of the more general local similarity detection problem.

2.3 Feature tracking and time-varying topological structures

Feature tracking methods and time-varying topological structures as areas of research have a non-trivial overlap with comparison measures, comparison measures often become the foundation upon which these methods are constructed. But there are some differences too. In this section we review these methods, some based on comparing topological structures while others with a different flavour.

2.3.1 Tracking based on topology of isosurfaces

Many methods use isosurfaces as representatives for features of interest and track them. Shamir et.al. [109] describe a method to progressively track isosurfaces in time-varying scalar fields by combining spatial and temporal propagation followed by a step for tracking changes in topology. Sohn and Bajaj [114] address the problem of finding correspondences between isosurfaces at a fixed isovalue across time. They do so by defining spatial overlaps between sublevel and superlevel sets and constructing a topology change graph (TCG). Both papers [109, 114] showcase the utility of the proposed method by tracking vortices in turbulent vortex data. More details regarding isosurface topology based tracking can also be found in the comprehensive survey by Mascarenhas and Snoeyink [75].

2.3.2 Tracking based on topological structures

Various methods have been proposed in the literature to track features captured by topological structures. Laney et al. [68] use the MS complex to define and represent bubbles in the mixing envelope of hydrodynamic instabilities and track them over time. Bremer et.

al. [16] use the Morse complex to define features, followed by a hierarchical representation and construction of tracking graphs to track the evolution of combustion in lean premixed hydrogen flames. Bremer et.al. [17] facilitate exploration and analysis of burning cells from turbulent combustion simulation. Weber et al. [134] track burning regions by extracting iso-volumes in a 4D space-time temperature field, followed by construction of Reeb graphs of time defined on the 4D domain. They convert a 4D tracking problem into the computation of the Reeb graph. Widanagamaachchi et al. [137] use correspondences between branches of merge trees followed by progressive construction of tracking graphs to track features in combustion simulations. They also provide a linked view for concise and effective visualization of the features. In a follow up work [138], they present methods to handle temporal artifacts, temporal simplification, and a parameter independent approach to track embedded features and apply it to track extinction holes in turbulent combustion simulations. Saikia and Weinkauff [101] use merge trees to represent multi-scale features and use a global shortest path formulation together with dynamic time warping to identify similar spatio-temporal structures. Lukasczyk et al. [72, 73] use a nesting tree, a variant of the merge tree, to capture hierarchical features and track them in dynamic nested tracking graphs. Soler et.al. [116] use lifted Wasserstein matcher to find temporal correspondences between critical points and track features. Many other works propose different forms of tracking graphs to support the study of evolving topological features [66, 107, 106]. Tracking features in vector fields is also an active area of research [122, 96].

2.3.3 Time-varying topological structures

Many time-varying counterparts of topological structures have been proposed to facilitate analysis of feature rich data. Cohen-Steiner et al. [26] describe an algorithm to update persistence diagrams and use it to study protein folding trajectories. Edelsbrunner et al. [45] apply Jacobi curves to track the temporal evolution of Reeb graphs by describing a complete characterization of the combinatorial changes that occur in the Reeb graph of a time-varying scalar field. Oesterling et al. [89] introduced time-varying merge trees which provides a topological summary of time-varying scalar fields by tracking features for all scalar thresholds. They represent the time-varying merge trees as a sequence of landscape profiles and showcase the utility of the method by applying it towards the analysis of time-varying high dimensional point clouds.

2.4 Tree edit distance

In this section we review tree edit distance (TED) as it forms the foundation for some of the comparison measures we later define for topological structures. Tree edit distance has various applications, such as comparing neuronal trees [52], comparing shapes [65], comparing music genre taxonomy [79], analysis of glycan structures [50], comparing RNA structures [105], and comparing plant architectures [47]. Edit distances and alignment distances for trees are inspired by edit distances defined on strings. Given two strings, one is transformed into the other via a sequence of operations where each operation has a non-negative associated cost. The distance is defined as the minimum cost over all such transformations. Similar distance measure may be defined for labeled trees with edit operations like relabeling, addition, and deletion of nodes. Zhang and Shasha [145] described an algorithm to compute the tree edit distance for ordered labeled trees. Later, Zhang [143] proposed a new algorithm for constrained tree edit distance for ordered labeled trees. The computation of tree edit distance for unordered labeled trees is NP-complete [146]. However, the constrained version of the problem can be solved in polynomial time using a dynamic programming based algorithm [144]. A gap corresponds to a collection of nodes that are inserted / deleted during a sequence of edit operations. Edit distance with arbitrary gap costs were first proposed by Touzet [132], who showed that the distance computation is NP-hard. But, the distance between ordered labeled binary trees can be computed in polynomial time [139]. While tree edit distance based algorithms have been employed in many applications, they have not been well studied for comparing topological structures like merge trees except in very recent work. Riecke et al. [97] defined a hierarchy of persistence pairs and a tree edit distance based dissimilarity measure to compare hierarchies.

2.5 Comparison of metric graphs

In this section we review methods used to compare metric graphs which can be adapted to compare some of the graph-based and complex-based topological structures.

2.5.1 Persistence Distortion for metric graphs

The persistence distortion distance between metric graphs is discussed separately as its setting is more general, i.e. the objects are metric graphs which can be thought of as super-sets of many of the topological descriptors discussed till now.

Dey et al. [34] defined the persistence distortion distance to compare metric graphs, proved its stability (w.r.t graphs not scalar fields), and described a polynomial time algo-

rithm with asymptotic running time $O(m^{12} \log n)$ (continuous version) and $O(n^2 m^{1.5} \log m)$ (discrete version), where m is the number of edges and n is the number of vertices in the larger graph. They also reported applications to shape matching.

2.5.2 Gromov-Hausdorff and Gromov-Wasserstein distances

Gromov [53] introduced Gromov-Hausdorff (GH) distance between metric spaces, though NP-hard to compute for general cases, it has been used in shape matching applications by treating shapes as metric spaces [80]. Later Memoli [81] introduced Gromov-Wasserstein distance framework, relaxing GH and making it amenable to practical applications. Since then GW and its variants (fused GW [130], scalable GW [140], sliced GW [131], sampled GW [64], quantized GW [25]) have been utilized in many applications, like comparing graphs [130], networks [24], computing barycenters of point clouds [93], graph partitioning [140], sketching merge trees [71], segment transfer [25], alignment of word embeddings [4], cross domain alignment [23] and various machine learning applications.

2.6 Summary

From the discussion presented in the sections above, we summarize the key aspects from all the previous work described. Table 2.6 provides the properties of some of the measures discussed in this chapter.

We observe from Table 2.6 that there have been many comparison measures defined for various topological descriptors, but they either suffer from high computational complexity making them unusable in practice or they lack theoretical properties which would provide guarantees. Interpretation of these measures are also desirable since that would help the user to understand how the measure changes w.r.t changes in the input. The choice of measures for different structures also vary a lot. In particular set and graph based structures have a lot to choose from while the complex based structures have hardly a few measures. Local comparison measures are not well-explored, which can be crucial for many applications. We also note that from Yan et.al. [141, Table 2] these observations are not limited to the measures which we have discussed in the previous sections but they hold true in general for many other measures which exist in the literature barring few exceptions.

With these observations, we aim to design comparison measures that address some of these gaps to whatever extent it is possible. Table 2.6 provides the properties of the measures defined in this thesis.

Measures	Metric	Stability	Discrimination	Complexity	Global/Local
Comparing persistence diagrams					
Bottleneck distance [27]	extended pseudometric	Yes	baseline n/a	$O(n^{1.5} \log(n))$	global
p -Wasserstein distance [28]	extended pseudometric	Yes	Yes	$O(n^3)$	global
Lifted Wasserstein [116]	conj. Metric	unknown	unknown	$O(n^3)$	global
Comparing merge/contour trees, Reeb graphs					
Functional distortion distance [7]	extended pseudometric	Yes	Yes	NP-hard	global
Edit distance for Reeb graphs [35]	extended pseudometric	Yes	unknown	We conj. NP-hard	global
Interleaving distance between merge trees [84]	metric	Yes	Yes	NP-hard	global
Distance based on branch decompositions [10]	We conj. Yes	We conj. No	We conj. Yes	$O(n^3 \log(I_\epsilon))$	global
Distance based on histograms for merge trees [103]	metric	unknown	unknown	$O(n^2 B)$	global
Comparing extremum graphs					
Distance between extremum graphs [88]	metric	unknown	unknown	NP-hard	global

Table 2.1: We summarize the pros and cons of some of the measures discussed in the related work in this table. n is the number of critical points in a topological descriptor; I_ϵ is the search range for ϵ_{min} ; B is the number of bins in a histogram; “conj” means conjecture. This table borrows from the STAR report by Yan et.al [141, Table 2].

Measures	Metric	Stability	Discrimination	Complexity	Global/Local
Comparing merge trees					
OTED	conj. Yes	conj. No	conj. Yes	$O(n^5)$	global
MTED	Yes	conj. No	conj. Yes	$O(n^2)$	global
LMTED	Yes	conj. No	conj. Yes	$O(n^2)$	local
Comparing extremum graphs					
PDEG	Yes	conj. Yes	Yes	$O(n^2 m^{1.5} \log m)$	global
GWEG	Yes	unknown	conj. Yes	$O(n^3)$ [93]	global

Table 2.2: We summarize the pros and cons of the measures we present in this thesis. n is the number of critical points in a topological descriptor; m denotes the edges in the descriptor; “conj” means conjecture.

Chapter 3

Background

In this chapter we present all the relevant background required to understand subsequent chapters. The content of this chapter is mainly based on the *State of The Art Report* (STAR) on Comparative measures by Yan et.al. [141]. Relevant portions also come from the background sections of [119, 120, 117, 118, 31]. While the STAR by Yan et.al. [141] is more comprehensive, we restrict ourselves to the topological structures which we have used in this thesis. For more computation-oriented details see [148, 40] and for visualization-oriented details see [127]. We review set-based (persistence diagrams, barcodes), graph-based (merge trees, contour trees, Reeb graphs), and complex-based (Morse and Morse-Smale complexes) topological structures and their variants. The graph-based structures are largely based on level sets of a scalar function, whereas complex-based ones are primarily based on its gradient.

3.1 Scalar field and Morse theory

Scalar fields are real valued functions. They can be classified [141] as follows

A *single field* f is a scalar-valued field defined on a 2D, 3D, or higher-dimensional domain \mathbb{X} , $f : \mathbb{X} \rightarrow \mathbb{R}$.

A *time-varying field* F is a dynamic field varying over time, and is defined over the Cartesian product of a spatial domain \mathbb{X} and a time axis \mathbb{R} , $F : \mathbb{X} \times \mathbb{R} \rightarrow \mathbb{R}$. Time-varying data is typically available as a discrete set of temporal snapshots.

An *ensemble* is a collection \mathcal{F} of scalar fields that are indexed by a collection of parameters, $\mathcal{F} = \{f_i : i \in I\}$ (where I is an index set).

The next section defines properties which make real valued functions Morse. The topological structures which we further describe find their origin in Morse theory [82], see [78] for more details.

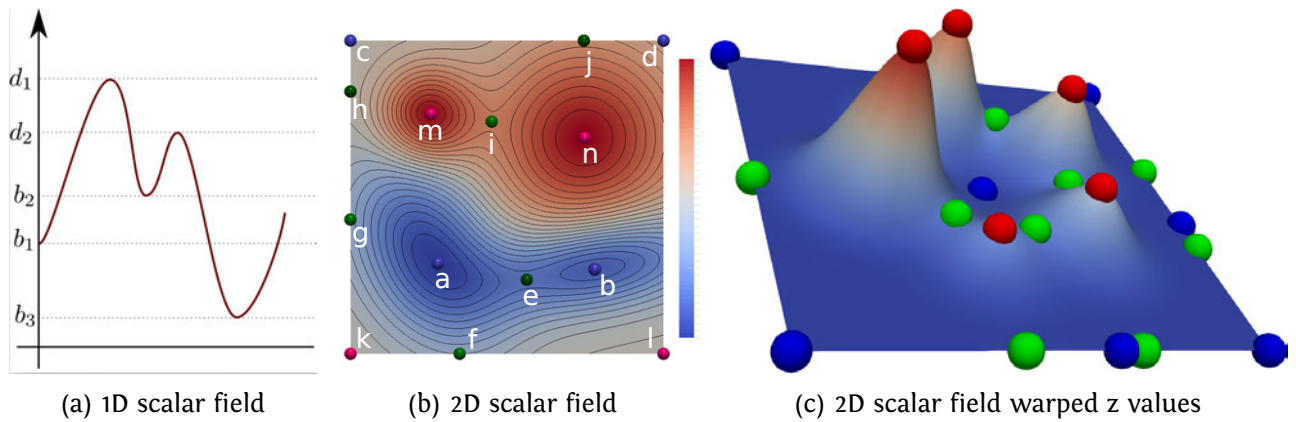


Figure 3.1: Morse functions with (a) a 1-dimensional and (b) a 2-dimensional domain (c) 2-dimensional domain where z coordinates are warped by the scalar value. We consistently use the (—) color map for the scalar field. Depending on the variety of critical points (—) color map for representing critical points based on their Morse index (0: minimum, 1: 1-saddle, 2: 2-saddle) or (—) where Morse Index is as follows (0: minimum, 1: 1-saddle, 2: 2-saddle, 3: maximum).

3.1.1 Morse function

Let \mathbb{M} be a smooth manifold and $f : \mathbb{M} \rightarrow \mathbb{R}$ a smooth function on \mathbb{M} . A point $x \in \mathbb{M}$ is a *critical point* of f if and only if the partial derivatives at x are zero; otherwise, it is a *regular point*. The image of a critical point is a *critical value* of f . A critical point x is *non-degenerate* if the Hessian (the matrix of second derivatives) at x is non-singular. f is a *Morse function* if all its critical points are non-degenerate and have distinct function values. Given the function f and a value c in the range of f (called *isovalue*) all points in \mathbb{M} which map to c i.e. $\{x \in \mathbb{M} \mid f(x) = c\}$ form a *level set*. It is also known as *isocontour* or an *isosurface* depending on the dimension of the domain. *Morse index* which is defined as the number of negative eigenvalues of the Hessian matrix, which alternatively counts the number of independent directions along which the function f decreases, is used to classify the critical points of f . The Morse indices of the *minimum* and *maximum* are 0 and n , respectively, rest of the critical points which are referred to as *k-saddles*, have indices k , $1 \leq k \leq (n-1)$. Figure 3.1 gives examples of Morse functions with a 1- and a 2-dimensional domain, respectively. Critical points are always displayed as red (for local maxima), blue (for local minima), and green (for saddles) spheres, level sets are curves displayed in black.

3.1.2 Morse theory

For a Morse function $f : \mathbb{M} \rightarrow \mathbb{R}$, we define sublevel sets as the preimage of $f^{-1}(-\infty, t]$ i.e. $\mathbb{M}_t := f^{-1}(-\infty, t] = \{x \in \mathbb{M} \mid f(x) \leq t\}$ and the superlevel sets as the preimage of $f^{-1}[t, \infty)$ i.e. $\mathbb{M}_t := f^{-1}[t, \infty) = \{x \in \mathbb{M} \mid f(x) \geq t\}$. Morse theory states that almost all functions are Morse and a non-Morse function can be made into a Morse function by resolving degenerate conditions via the simulation of simplicity [41] in practice. All scalar functions discussed in this thesis are assumed to be Morse.

The Morse lemma states that a function looks extremely simple near a non-degenerate critical point. Two fundamental theorems of Morse theory state that the topology of sublevel/superlevel sets \mathbb{M}_t change as t varies, in particular, when t passes a critical value. See [82, Theorems 3.1 and 3.2]. Topological descriptors are related with one another via these theorems of Morse theory as they are defined over the sublevel/superlevel sets of a function or over the behaviour of the gradients of the function.

Smooth functions don't exist in practice, instead, we always encounter sampled version of such functions, represented as a function on a point cloud sample of \mathbb{M} . To make things more amenable to computation, a combinatorial structure (i.e., a simplicial complex K) is built on the sampled space as an approximation of \mathbb{M} . Let K be such a simplicial complex with real values specified on its vertices; $|K|$ represents its *underlying space*. We obtain a piecewise linear (PL) function $f : |K| \rightarrow \mathbb{R}$ using linear extension over the simplices, where $f(x) = \sum_i b_i(x)f(u_i)$ (u_i are vertices of K and $b_i(x)$ are the barycentric coordinates of x) [40, page 135]. We can then apply Morse-theoretical ideas to this PL approximation. This application is justifiable according to the *Simplicial Approximation Theorem* [40, page 56], which states that every continuous function on a triangulable topological space can be approximated by a PL function.

3.2 Topological structures

Using Morse-theoretical ideas we can build various topological structures based on the way the critical points are related, they can be without connections (set-based structures), connected to form trees and graphs (graphs-based structures), and augmented with geometric information (complex-based structures). We describe them in subsequent sections.

3.2.1 Set-based topological structures

We describe set-based topological descriptors like persistence diagrams in this section. For a comprehensive treatment with different perspectives we refer the reader to [40, 21, 11].

3.2.1.1 Persistence diagram and barcode

For a Morse function defined on compact \mathbb{M} i.e. $f : \mathbb{M} \rightarrow \mathbb{R}$ with sublevel sets $\mathbb{M}_t := f^{-1}((-\infty, t])$ would have finitely many critical points (as a consequence of the Morse lemma). Let n be the (finite) number of critical values of f . Let $a_0 < \dots < a_n$ be a sequence of regular values of f such that each interval (a_i, a_{i+1}) contains exactly one critical value of f . A sublevel set filtration of f is a sequence of sublevel sets connected by inclusions,

$$\mathbb{M}_{a_0} \rightarrow \mathbb{M}_{a_1} \rightarrow \dots \rightarrow \mathbb{M}_{a_n}.$$

Persistent homology studies the topological changes of sublevel sets by applying k -dimensional homology ($k \geq 0$) to this sequence,

$$H_k(\mathbb{M}_{a_0}) \rightarrow H_k(\mathbb{M}_{a_1}) \rightarrow \dots \rightarrow H_k(\mathbb{M}_{a_n}).$$

Given a topological space \mathbb{X} , the 0-, 1-, and 2-dimensional homology groups, denoted as $H_0(\mathbb{X})$, $H_1(\mathbb{X})$, and $H_2(\mathbb{X})$, respectively, capture the connected components, tunnels, and voids of \mathbb{X} . We give an example of 0-dimensional persistence homology based on the sublevel set filtration of a 1-dimensional Morse function in Figure 3.2.

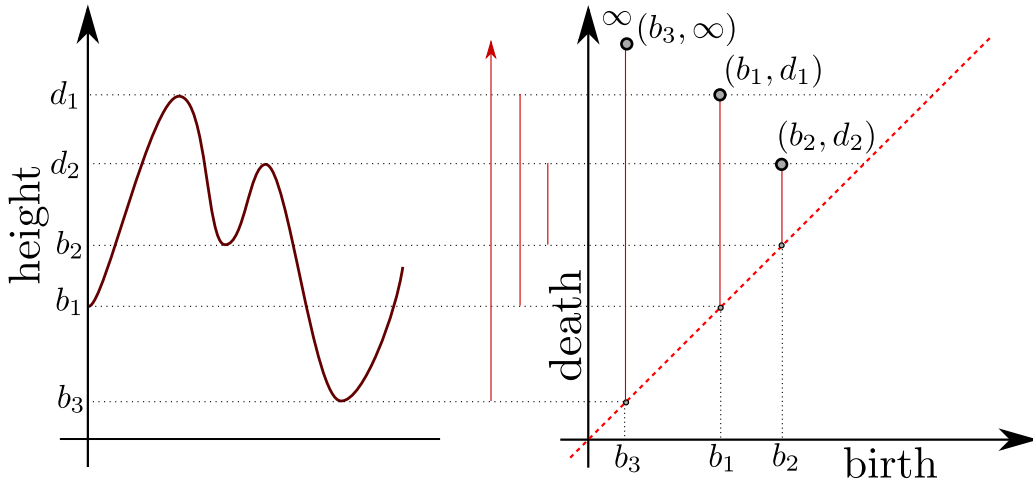


Figure 3.2: The graph of $f : \mathbb{M} \rightarrow \mathbb{R}$ (left); together with the 0-dimensional barcode (middle) and 0-dimensional persistence diagram of f (right) Each birth-death pair (b_i, d_i) is a feature of the scalar function and its persistence is defined as $d_i - b_i$. Each pair is represented as a point in \mathbb{R}^2 .

Formally, a k -dimensional persistence diagram \mathcal{D} is the disjoint union of a multi-set of off-diagonal points $\{(b, d) \mid b \neq d, b, d \in \mathbb{R}_{\geq 0}\}$ on the Euclidean plane $\bar{\mathbb{R}}^2$ (where $\bar{\mathbb{R}} =$

$\mathbb{R} \cup \{-\infty, +\infty\}$) and the diagonal $\Delta = \{(b, b) \mid b \in \mathbb{R}_{\geq 0}\}$ counted with infinite multiplicity.

Let c_i denote the critical values of a Morse function f restricted to an interval $\mathbb{M} \subset \mathbb{R}$, $f : \mathbb{M} \rightarrow \mathbb{R}$, where $c_0 < c_1 < \dots < c_{n-1}$. Let x_i denote the critical points of f . Assume f is Morse, then $c_i = f(x_i)$. For simplicity, we set $c_0 = 0$, $c_1 = 1$, and $c_i = i$, etc. Let $a_0 < a_1 < \dots < a_n$ be a sequence of regular values of f such that each interval (a_i, a_{i+1}) contains exactly one critical value c_i .

The 0-dimensional persistent homology captures how connected components in the sublevel sets \mathbb{M}_t changes as t varies from a_0 to a_n .

From Figure 3.2(left), at $t < b_3$, $\mathbb{M}_t = \emptyset$. At $t = b_3$, a single (connected) component appears in the sublevel set \mathbb{M}_t containing the global minimum, we call this a *birth* event at \mathbb{M}_0 . At $t = b_1, b_2$ 2nd and 3rd component appears in \mathbb{M}_t containing local minima. At $t = d_2$, the component merges with the component born at b_3 as per the *Elder Rule* [40, Page. 150], referred to as a *death* event: the component disappears (dies) while the component created b_3 remains. At $t = d_1$, the component containing born at b_1 merges with the component born b_3 and dies.

Persistent homology pairs the birth and death events either as a set of intervals (called *barcode*), or a multi-set of points in the plane (called *persistence diagram*).

Barcodes A barcode is shown in Figure 3.2(middle). The component born at b_3 never dies, giving rise to a bar $[b_3, \infty)$ in the barcode that begins at b_3 and goes to ∞ . The component born at b_1 dies at d_1 , which corresponds to the bar in the middle, similarly, the birth and death events b_2, d_2 gives rise to an additional bar on the right. A persistence diagram is shown in Figure 3.2(right), where each bar $[b, d)$ is mapped to a point (b, d) on the plane.

The *topological persistence* which captures the life span of a component in the filtration quantifies the importance of a pair of critical points in the diagram. It is defined as the absolute difference between the function values i.e. $|d - b|$

For other variants of persistence diagram like persistence landscapes, betti curves etc refer to survey by Yan.et.al [141].

3.2.2 Graph-based topological structures

Graph-based topological structures such as merge trees, contour trees, and Reeb graphs capture topological changes of (sub)level sets of scalar fields. While the same holds for superlevel sets in the current discussion we mention sublevel sets.

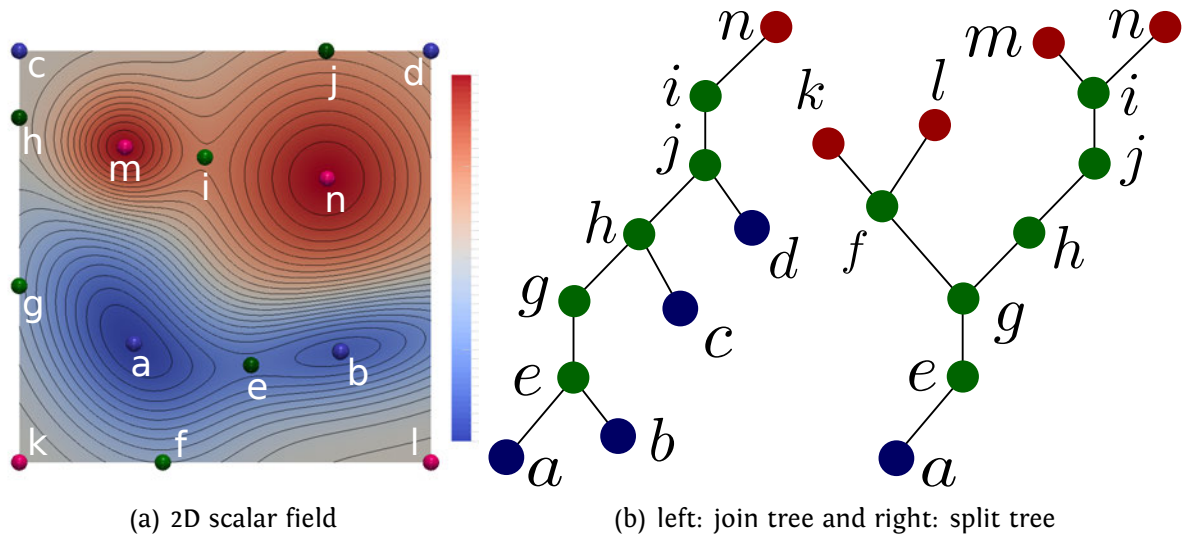


Figure 3.3: Merge trees. (a) A 2D scalar field (b) A merge tree tracks the connectivity of sublevel sets (preimage of $f^{-1}(-\infty, c]$) or the superlevel sets (preimage of $f^{-1}[c, \infty)$).

3.2.2.1 Merge tree

Given a Morse function $f : \mathbb{M} \rightarrow \mathbb{R}$ defined on a connected domain \mathbb{M} , a merge tree records the connectivity of its sublevel sets. Two points $x, y \in \mathbb{M}$ are *equivalent* (w.r.t f), $x \sim y$, if they have the same function value, that is, $f(x) = f(y) = t$, and if they belong to the same connected component of the sublevel set \mathbb{M}_t , for some $t \in \mathbb{R}$. A *merge tree* is the quotient space \mathbb{M}/\sim obtained by gluing together points in \mathbb{M} that are equivalent under the relation \sim . It keeps track of the evolution of connected components in \mathbb{M}_t as t increases; see Figure 3.3 for an example.

In the abstract view of a merge tree in Figure 3.3(b), each leaf corresponds to a local minimum of f that represents the birth of a connected component; each internal node corresponds to the merging of components; and the root represents the entire space as a single component. Note that the notions of join and split trees [22] are the two forms of merge trees; a join tree is the merge tree of f and a split tree is the merge tree of $-f$.

Nodes of join trees consist of minima $M = \{m_i\}$, saddles $S = \{s_j\}$, and the global maximum. In theory, the structure of a join tree is simple. Excluding the global maximum, which is the root of the tree, every node has either 0 (minimum) or 2 children (saddle). All minima are paired with saddles based on the notion of topological persistence [42] except for one which is paired to the lone global maximum. Each such pair (m, s) represents a topological feature and its persistence is defined as $\text{pers}(m) = \text{pers}(s) = f(s) - f(m)$. In practice, saddles may have more than two children. A split tree is defined likewise. It

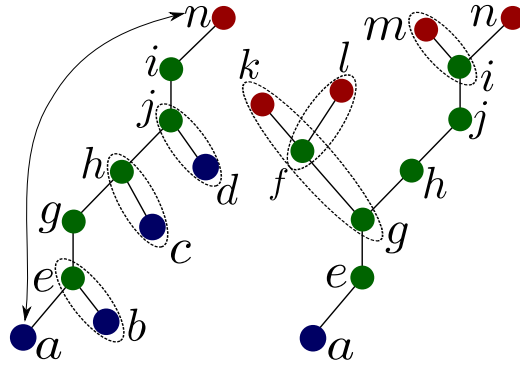


Figure 3.4: Persistence pairs in the join (left) and split (right) trees.

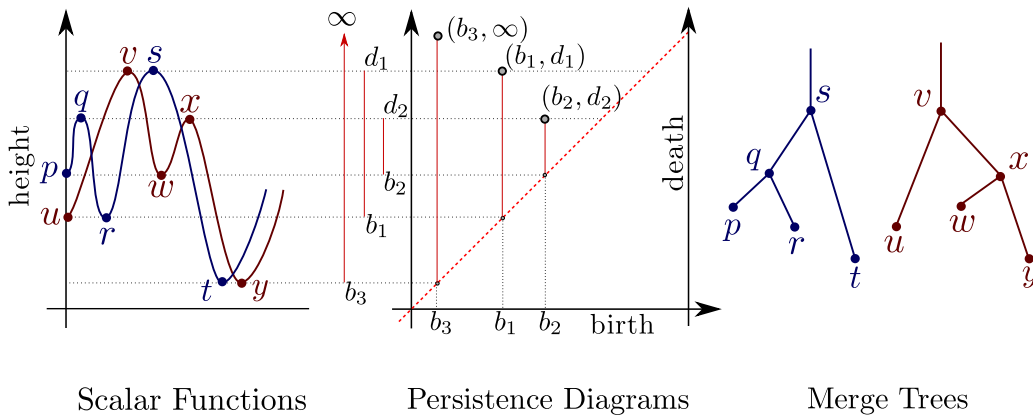


Figure 3.5: Two scalar functions (blue and red) and the corresponding persistence diagrams and merge trees. Even though the scalar functions and the corresponding trees are different, the persistence diagram is the same.

contains a set of maxima and saddles together with the global minimum. Figure 3.4 shows the persistence pairing for the trees from Figure 3.3.

As merge trees contain more information than persistence diagrams, they can be used generate the later. The reverse however is not always possible since there can be different trees which would result in the same persistence diagram and barcodes. See Figure 3.5.

3.2.2.2 Reeb graph and contour tree

A Reeb graph, on the other hand, relies on equivalence relations among points in the level sets of a Morse function $f : \mathbb{M} \rightarrow \mathbb{R}$. Two points $x, y \in \mathbb{M}$ are *equivalent*, $x \sim y$, if $f(x) = f(y) = t$, and if they belong to the same connected component of the level set $f^{-1}(t)$, for some $t \in \mathbb{R}$. The *Reeb graph* $\mathcal{G}_f := \mathbb{M}/\sim$ is the quotient space obtained by identifying equivalent points; see Figure 3.6. Nodes in the Reeb graph have a one-to-one correspondence with the critical points of f , while arcs connect the nodes. A point on an arc represents

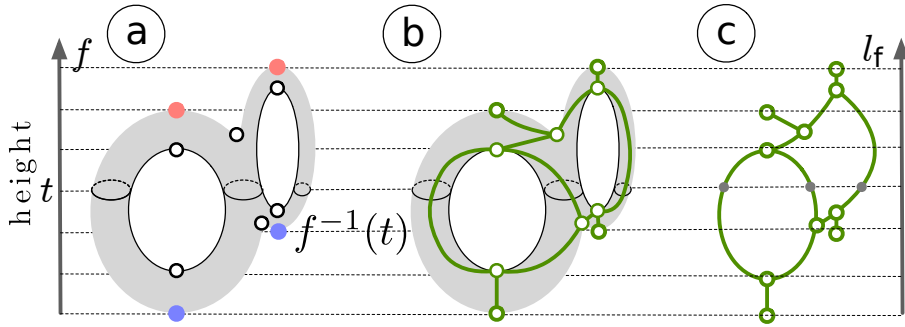


Figure 3.6: (a) A height function $f : \mathbb{M} \rightarrow \mathbb{R}$ defined on a double torus, (b) its Reeb graph embedded in the domain \mathbb{M} , and (c) its Reeb graph shown in an abstract view. If the Reeb graph in (c) is further equipped with a function l_f defined on its vertices, where l_f is the restriction of f to V , then we obtain a labeled Reeb graph. Image from [141]

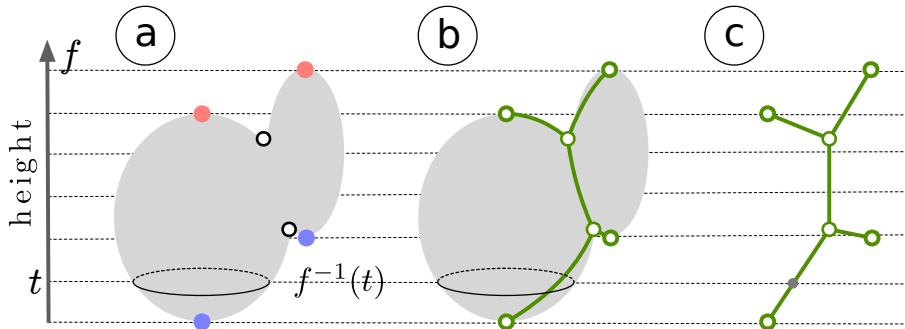


Figure 3.7: (a) A height function $f : \mathbb{M} \rightarrow \mathbb{R}$ defined on the surface of two (solid) balls glued together; (b) its contour tree embedded in the domain \mathbb{M} ; and (c) its contour tree shown in an abstract view. Image from [141]

a connected component of a level set in \mathbb{M} . Intuitively, as t increases within the range of f , a Reeb graph captures the topological changes in the level sets of f , in particular, the appearances, disappearances, splitting, and merging among the connected components (contours) of $f^{-1}(t)$; see [40, section VI.4] for a formal treatment.

A contour tree is a special type of Reeb graph when the domain \mathbb{M} is simply connected. Then, \mathbb{M}/\sim gives rise to a tree; see Figure 3.7 for an example involving a “deformed” spherical domain. The main difference between a contour tree and a merge tree is that the former captures the connectivity among level sets, while the latter encodes the connectivity among sublevel sets of a Morse function.

A related structure is the *branch decomposition tree* (BDT) which can be derived from a contour tree [91] or a merge tree [102]. A BDT represents the branch decomposition of a tree, with the nodes representing the branches and the edges representing their hierarchy.

Saikia et.al. [102] further introduced an *extended branch decomposition graph* (eBDG), which represents a forest of BDTs, where each of the BDTs is computed from a subtree of the merge tree.

3.2.3 Complex-based topological structures

We review complex-based structures like the Morse and Morse-Smale complex, extremum graphs which are subgraphs of these complexes. These structures are derived from the gradient of the function f rather than the level sets.

3.2.3.1 Morse and Morse-Smale complex

A curve $l(r)$ is defined as an *integral line* of f if $\frac{\partial}{\partial r}l(r) = \nabla f(l(r))$ for all $r \in \mathbb{M}$. The critical points of f are exactly the limit points of an integral line. The collection of all integral lines that begin and terminate at a critical point p is referred to as the *ascending manifold* and the *descending manifold* of p respectively. The dimensions of the ascending and descending manifolds of an index k critical point are $n - k, k$ respectively. These ascending and descending manifolds partition \mathbb{M} into valley-like and mountain-like regions respectively called the *Morse decomposition*. These represent a collection of monotonic regions when overlaid upon \mathbb{M} . The *Morse-Smale (MS) complex* is the partition of \mathbb{M} into cells formed by collection of integral lines having common source and destination. We refer to the 0-dimensional cells as *nodes* and 1-dimensional cells as *arcs*. Figure 3.8(b) shows the extremum graph of a 2D scalar field.

While the MS complex provides a comprehensive description of the topology of the scalar field, it has some disadvantages. The complexes are often large which results in clutter while visualizing and sensitive to noise which results in lots of spurious saddles which increases the size dramatically hindering its utility to aid meaningful visualizations.

3.2.3.2 Extremum graph

Correa et al. [30], observe that extrema are associated with interesting and meaningful topological features. They introduce a topological abstraction called the extremum graph. The *extremum graph* is a substructure of MS complex that captures the connectivity between maxima and saddles (maximum graph) or between minima and saddles (minimum graph). In this paper, we focus on the maximum graph while referring to it as the extremum graph to simplify terminology. The extremum graph $G(V, E)$ consists of the node set V consisting of the maxima and $(n - 1)$ -saddles of f and the arc set E consisting of the connecting arcs (m_i, s_j) between maxima and $(n - 1)$ -saddles. If a pair of maxima $m_1, m_2 \in V$ are adjacent to a common saddle s_1 , namely both (m_1, s_1) and (m_2, s_1) belong to E , then the segments

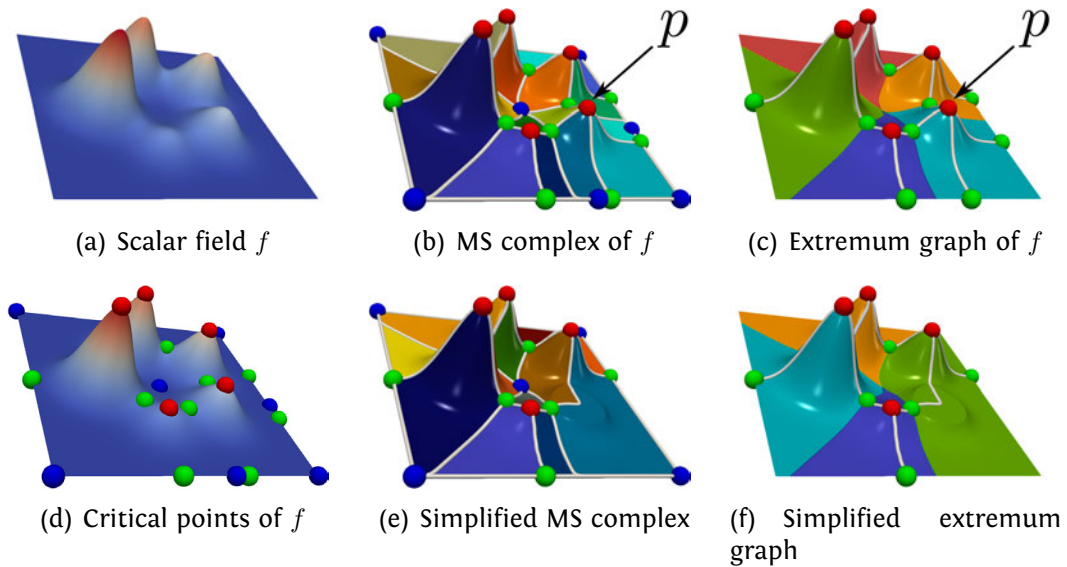


Figure 3.8: Extremum graph of a 2D scalar field. (a) Scalar field f defined on a 2D domain, shown using both a color map and surface height. (d) Critical points of f . (b) MS complex and the segmentation of the domain into monotonic regions. (c) Extremum graph of f embedded within the domain represents the peaks and adjacency relationship between their corresponding segments. (e) Simplified MS complex obtained by canceling critical point pairs, including a max-saddle pair. The peak p is canceled together with an adjacent saddle thereby flattening one of the mountains. (f) Extremum graph corresponding to the simplified MS complex.

associated to m_1 and m_2 are adjacent to each other. Figure 3.8(c) shows the extremum graph of a 2D scalar field.

3.3 Topological simplification

All the topological structures can be simplified by removing topological noise using a sequence of critical point pair cancellations [43]. The ordering of such cancellations are based on the notion of topological persistence [42]. A sequence of repeated cancellations in the increasing order of persistence is used to simplify topological structures. In case of graph based structures, cancellation would result in the removal of an edge. In case of an MS complex, an individual cancellation operation removes the two critical points, the arc connecting them, arcs incident on the two critical points, and reconnects of the surviving critical points in the neighborhood [43, 56]. Note that such simplification would also result in changes in the scalar field and updating the original field to keep it consistent is crucial for many applications. The MS complex in Figure 3.8(b) is simplified by canceling

a maximum-saddle pair, resulting in the MS complex in Figure 3.8(e). The simplification removes the peak p and merges its associated segment with the adjacent segment. The extremum graph can also be simplified by simplifying the corresponding MS complex. Figure 3.8(f) shows the simplified extremum graph obtained by canceling a maximum-saddle pair.

3.4 Metric graphs

In this section we describe metric graphs which are used in the formulation of the comparison measure for extremum graphs. We follow the definition given by Dey et.al. [34]. We also describe a particular metric of interest, the *function metric*.

Given a graph $G(V, E)$ and a weight function $w : E \rightarrow \mathbb{R}^+$ on the set of edges E , we can define a metric graph $(|G|, d_G)$ by considering $|G|$ to be the geometric realization of G and $|e|$ as the image of some edge $e \in E$. the metric d_G can be defined considering the arc length parametrization $e : [0, w(e)] \rightarrow |e|$ for every $e \in E$ and for any two points $x, y \in |e|$ define $d_G(x, y) = |e^{-1}(x) - e^{-1}(y)|$. For a path $\pi(u, v)$ where $u, v \in |G|$ lie on different edges, the length of the path can be obtained by summing up the lengths of the restrictions of this path to the edges in G . Given any two points $u, v \in |G|$ the distance $d_G(u, v)$ is given by the minimum length path connecting u to v in $|G|$. To simplify the notation, (G, d_G) is used to denote the metric graph.

Given two graphs G_1, G_2 with functions $f : G_1 \rightarrow \mathbb{R}$ and $g : G_2 \rightarrow \mathbb{R}$ respectively. Define (pseudo)metrics on the input graphs as induced by f and g .

Function metric. Given $x_1, x_2 \in G_1$, Bauer et.al. [7] define the *function metric* d_f as

$$d_f(x_1, x_2) = \min_{\pi: x_1 \rightsquigarrow x_2} \text{height}(\pi) \quad (3.1)$$

here π ranges over all paths connecting $x_1, x_2 \in G_1$ and $\text{height}(\pi) = \max_{x \in \pi} f(x) - \min_{x \in \pi} f(x)$ is the maximum f -function value difference for points from the path π .

3.5 Properties of comparison measures

This section is the same as Section 2.2 from Yan et.al. [141]. We discuss desirable properties of a comparative measure $d = d(\mathcal{A}_1, \mathcal{A}_2)$ between a pair of topological descriptors (of the same type), \mathcal{A}_1 and \mathcal{A}_2 . We focus on four types of properties surrounding *metricity*, *stability*, *discriminativity*, and *computational complexity*. These properties have been studied across scattered literature in TDA and visualization.

Metricity and Pseudometricity. Requiring d to be a metric is desirable. That is, d satisfies the following metric properties:

1. Non-negativity: $d(\mathcal{A}_1, \mathcal{A}_2) \geq 0$;
2. Identity: $d(\mathcal{A}_1, \mathcal{A}_2) = 0$ iff $\mathcal{A}_1 = \mathcal{A}_2$;
3. Symmetry: $d(\mathcal{A}_1, \mathcal{A}_2) = d(\mathcal{A}_2, \mathcal{A}_1)$;
4. Triangle inequality: $d(\mathcal{A}_1, \mathcal{A}_2) \leq d(\mathcal{A}_1, \mathcal{A}_3) + d(\mathcal{A}_2, \mathcal{A}_3)$.

If the triangle inequality (item 4) above is not required, d becomes a *dissimilarity measure* instead. If the identity is not required, d becomes a *pseudometric*, replacing item 2 above by:

- $d(\mathcal{A}_1, \mathcal{A}_1) = 0$ (but possibly $d(\mathcal{A}_1, \mathcal{A}_2) = 0$ for some distinct $\mathcal{A}_1 \neq \mathcal{A}_2$).

Stability. Many definitions of stability for a distance metric d with respect to the underlying scalar field have been proposed. Stability can refer to whether d is stable with respect to simplification or perturbation of the underlying function. For example, given two scalar fields f_1 and $f_2 : \mathbb{X} \rightarrow \mathbb{R}$ that give rise to a pair of topological descriptors \mathcal{A}_1 and \mathcal{A}_2 , d is L^∞ -stable if for some constant $C > 0$,

$$d(\mathcal{A}_1, \mathcal{A}_2) \leq C \cdot \|f_1 - f_2\|_\infty.$$

Intuitively, stability requires that if the functions are not too “different” in terms of the L^∞ norm of the difference between the functions then the comparison measure between the topological structures representing the scalar functions should also be small.

Discriminativity. Discriminativity also has various definitions. For instance, using a comparative measure d_0 as a baseline, d is considered to be more discriminative than d_0 if for some constant $c > 0$,

$$d_0(\mathcal{A}_1, \mathcal{A}_2) \leq c \cdot d(\mathcal{A}_1, \mathcal{A}_2)$$

and there exists no constant $c' > 0$ such that $d_0 = c' \cdot d$ (that is, d is not a scaled version of d_0).

Discrimination, on the other hand, requires that however “small” the difference between two functions, it should be captured by the comparison measure. Specifically, the distance measure equals 0 should imply that the functions are equal.

Since the bottleneck distance d_B between persistence diagrams of f and g was among the first measures defined between topological structures, we typically state this property in terms of how a newly proposed distance d is related to bottleneck distance.

Figure 3.5 shows an example where $d_B = 0$ for a pair of functions that are not equal, which implies that d_B is not discriminative enough. One reason for the low discriminative power is that the persistence diagram, and hence d_B , does not incorporate the connectivity between critical points as in the merge tree.

In general comparison measures are designed so that they are more discriminative compared to the bottleneck distance.

$$d_B(f, g) \leq d(f, g) \leq \|f - g\|_\infty \quad (3.2)$$

Computational complexity. We investigate the computational complexity of d in terms of the time and space complexity. We investigate whether d is *easily implementable*, referring to whether an algorithmic solution has been proposed which affects its practicality.

The above properties are particularly desirable for analysis and visualization tasks that are supported by a comparative measure. They lead to theoretically sound, interpretable, robust, reliable, and practical methods for comparative visualization.

3.6 Comparison measures

In this section we discuss some of the comparison measures which are either used as the baseline to compare other measures (like bottleneck/Wasserstein distances) or foundational material for other measures we define later (like persistence distortion and Gromov-Wasserstein distance).

3.6.1 Bottleneck and Wasserstein distances

Bottleneck distance [27]. Given two persistence diagrams $\mathcal{D}_f, \mathcal{D}_g$ corresponding to scalar fields $f : \mathbb{M}_1 \rightarrow \mathbb{R}, g : \mathbb{M}_2 \rightarrow \mathbb{R}$ and a bijection $\eta : \mathcal{D}_f \rightarrow \mathcal{D}_g$, the *bottleneck distance* between \mathcal{D}_f and \mathcal{D}_g is defined as

$$d_B(\mathcal{D}_f, \mathcal{D}_g) = \inf_{\eta: \mathcal{D}_f \rightarrow \mathcal{D}_g} \sup_{x \in \mathcal{D}_f} \|x - \eta(x)\|_\infty. \quad (3.3)$$

Cohen-Steiner et.al.[27] provide an important stability result for d_B .

$$d_B(\mathcal{D}_f, \mathcal{D}_g) \leq \|f - g\|_\infty \quad (3.4)$$

Wasserstein distance [28]. The p -Wasserstein distance is defined as

$$d_p(\mathcal{D}_1, \mathcal{D}_2) = \left[\inf_{\eta: \mathcal{D}_1 \rightarrow \mathcal{D}_2} \sum_{x \in \mathcal{D}_1} \|x - \eta(x)\|_\infty^p \right]^{\frac{1}{p}} \quad (3.5)$$

3.6.2 Persistence distortion distance

Given metric graphs $\mathcal{G}_1 = (G_1, d_{G_1})$ and $\mathcal{G}_2 = (G_2, d_{G_2})$ where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let $n = \max\{|V_1|, |V_2|\}$ and $m = \max\{|E_1|, |E_2|\}$.

Considering a point $\mathbf{s} \in G_1$ as a base point and for any other point $x \in G_1$ consider shortest path distance from \mathbf{s} , $d_{G_1, \mathbf{s}} : G_1 \rightarrow \mathbb{R}$, $d_{G_1, \mathbf{s}}(x) = d_{G_1}(\mathbf{s}, x)$. Let $P_{\mathbf{s}}$ denote 0-dimensional persistence diagram ${}_0(d_{G_1, \mathbf{s}})$ induced by $d_{G_1, \mathbf{s}}$.

Consider base points $\mathbf{s} \in G_1$, $d_{G_1, \mathbf{s}} : G_1 \rightarrow \mathbb{R}$ and $\mathbf{t} \in G_2$, $d_{G_2, \mathbf{t}} : G_2 \rightarrow \mathbb{R}$ and corresponding persistence diagrams as $P_{\mathbf{s}}$ and $Q_{\mathbf{t}}$. Map \mathcal{G}_1 and \mathcal{G}_2 to the set of (infinite number of) points in the space of persistence diagrams \mathbb{D} given by $\mathcal{C} := \{P_{\mathbf{s}} | \mathbf{s} \in G_1\}$ and $\mathcal{F} := \{Q_{\mathbf{t}} | \mathbf{t} \in G_2\}$.

Persistence Distortion Distance [34]. The persistence-distortion distance between \mathcal{G}_1 and \mathcal{G}_2 , denoted by $d_{PD}(\mathcal{G}_1, \mathcal{G}_2)$, is the Hausdorff distance $d_H(\mathcal{C}, \mathcal{F})$ between the two sets \mathcal{C} and \mathcal{F} where the distance between two persistence diagrams is measured by the bottleneck distance. In other words,

$$d_{PD}(\mathcal{G}_1, \mathcal{G}_2) = d_H(\mathcal{C}, \mathcal{F}) = \max\left\{ \max_{P \in \mathcal{C}} \min_{Q \in \mathcal{F}} d_B(P, Q), \max_{Q \in \mathcal{F}} \min_{P \in \mathcal{C}} d_B(P, Q) \right\}. \quad (3.6)$$

3.6.3 Gromov-Hausdorff and Gromov-Wasserstein distances

Metric distortion in metric spaces (hence graphs) is given by the Gromov-Hausdorff distance [53]. We state the equivalent definition given in [20] and [80]. Given two metric spaces $\mathcal{X} = (X, d_X)$, $\mathcal{Y} = (Y, d_Y)$, the *correspondence* between \mathcal{X}, \mathcal{Y} is the relation $\mathcal{M} : X \times Y$ such that for any $x \in X$ there exists $(x, y) \in \mathcal{M}$ and for any $y' \in Y$ there exists $(x', y') \in \mathcal{M}$. The Gromov-Hausdorff distance is defined as

$$d_{GH}(\mathcal{X}, \mathcal{Y}) = \frac{1}{2} \inf_{\mathcal{M}} \max_{(x, y), (x', y') \in \mathcal{M}} |d_X(x, x') - d_Y(y, y')| \quad (3.7)$$

Where \mathcal{M} ranges over all correspondences. If \mathcal{X}, \mathcal{Y} are compact metric spaces and the diameter is defined as $\mathbf{diam}(X) = \max_{(x, x') \in X} d_X(x, x')$, a useful property of d_{GH} is as follows:

$$\frac{1}{2} |\mathbf{diam}(X) - \mathbf{diam}(Y)| \leq d_{GH}(X, Y) \leq \frac{1}{2} \max(\mathbf{diam}(X)), \quad (3.8)$$

Given a metric graph $\mathcal{G} = (G, d_G)$, let W be either a weighted adjacency matrix or the matrix containing the all-pairs shortest path distances w.r.t d_G . Since \mathcal{G} is finite graph both (G, d_G) and (V, W) provide the same information. \mathcal{G} can be represented as a measure network with additional information p which corresponds to probability measure supported on the nodes of G . In most cases p is taken to be uniform, i.e. $p = \frac{1}{n} \mathbf{1}_n, \mathbf{1}_n = \{1, 1, 1, \dots, 1\}^T \in \mathbb{R}^n$.

Let $G_1(V_1, W_1, p_1)$ and $G_2(V_2, W_2, p_2)$ be two graphs $|V_1| = n_1, |V_2| = n_2$. A *coupling* between p_1 and p_2 is defined as a joint probability measure on $V_1 \times V_2$ whose marginals agree with p_1 and p_2 , i.e. it is a non negative matrix C of size $n_1 \times n_2$ such that $C \mathbf{1}_{n_2} = p_1, C^T \mathbf{1}_{n_1} = p_2$. The *distortion* of a coupling C w.r.t a *loss function* L is defined as

$$\mathcal{E}(C) = \sum_{i, k \in [n_1], j, l \in [n_2]} L(W_1(i, k), W_2(j, l)) C_{i, j} C_{k, l}$$

If $\mathcal{C} = \mathcal{C}(p_1, p_2)$ denotes the collection of all couplings between p_1 and p_2 . The *Gromov-Wasserstein discrepancy* [93] is defined as

$$\mathcal{D}(C) = \min_{C \in \mathcal{C}} \mathcal{E}(C).$$

If L is chosen to be the quadratic loss function $L(a, b) = \frac{1}{2} |a - b|^2$, the *Gromov-Wasserstein distance* d_{GW} given by [81, 93] between G_1 and G_2 is defined as

$$d_{GW}(G_1, G_2) = \frac{1}{2} \min_{C \in \mathcal{C}} \sum_{i, k \in [n_1], j, l \in [n_2]} |W_1(i, k) - W_2(j, l)|^2 C_{i, j} C_{k, l}.$$

3.7 Tree edit distance

In this section we provide background on tree edit distance (TED). Further details can be found in the survey by [15].

Let T be a rooted tree with node set V and edge set E . For a node $v \in V$, $deg(v)$ is the number of children of v , and $parent(v)$ is its parent in the tree. The maximum degree of a node in the tree is denoted as $deg(T)$. We denote an empty tree by θ . Since we are

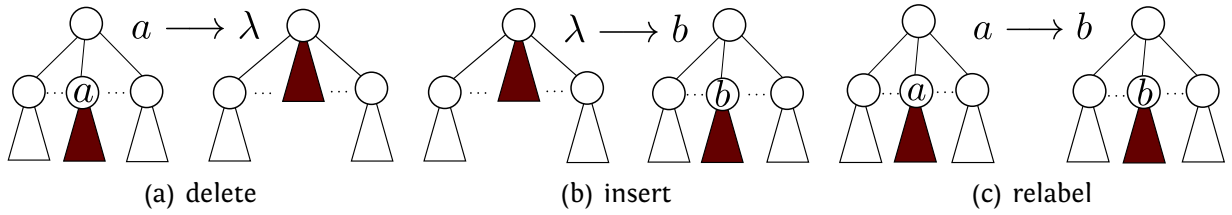


Figure 3.9: Three different tree edit operations. Each edit affects only one node in the tree. The null character λ corresponds to a gap.

interested in labeled trees, let Σ be the set of labels, and $\lambda \notin \Sigma$ denote the null or empty character, which corresponds to a gap. In the following discussion, we use notations and definitions from Zhang [144].

Edit operations. The edit operations differ based on the gap model. For this discussion we consider edit operations that modify the tree, one node at a time. Xu [139] gives a detailed discussion of general gaps where edits modify multiple nodes. We consider a total of three edit operations as shown in Figure 3.9.

1. **relabel:** A relabel $a \longrightarrow b$ corresponds to an operation where the label $a \in \Sigma$ of a node is changed to a label $b \in \Sigma$.
2. **delete:** A delete operation $a \longrightarrow \lambda$ removes a node n with label $a \in \Sigma$ and all the children of n are made the children of $parent(n)$.
3. **insert:** An insert operation $\lambda \longrightarrow b$ inserts a node n with label $b \in \Sigma$ as a child of another node m by moving all the children of m to children of n .

We define a cost function γ that assigns a non-negative real number to each edit operation of the form $a \longrightarrow b$. It is useful if the cost function γ satisfies metric properties i.e. $\forall a, b, c \in \Sigma \cup \{\lambda\}$

1. $\gamma(a \longrightarrow b) \geq 0, \gamma(a \longrightarrow a) = 0$
2. $\gamma(a \longrightarrow b) = \gamma(b \longrightarrow a)$
3. $\gamma(a \longrightarrow c) \leq \gamma(a \longrightarrow b) + \gamma(b \longrightarrow c)$

In particular, Zhang [144] proved that if γ is a metric then the edit distance is also a metric, else it will be merely a dissimilarity measure. Given a tree T_1 , we can apply a sequence of edit operations to transform it into another tree T_2 . If $S = s_1, s_2, \dots, s_k$ is a sequence of

edit operations, where each s_i is an edit, we can extend the cost function to S by defining $\gamma(S) = \sum_{i=1}^{|S|} \gamma(s_i)$.

Edit distance. Formally, the distance between two trees T_1, T_2 is defined as

$$D_e(T_1, T_2) = \min_S \{\gamma(S)\} \quad (3.9)$$

where S is an edit operation sequence from T_1 to T_2 .

3.7.1 Edit distance mappings

Computing the edit distance between merge trees is a minimization problem with a huge search space. In order to understand this search space and how it affects the computation, we first define some edit distance mappings – unconstrained, constrained, and restricted – and their properties as described by Zhang [144].

3.7.1.1 Unconstrained edit distance mappings

The sequence of edit operations performed to transform T_1 into T_2 determines a mapping between the two trees. For convenience, we order the nodes of both the trees. This ordering does not affect the distance. Let t_1 and t_2 denote the ordering of nodes in T_1 and T_2 , respectively, and $t_1[i]$ represents the i th node in the ordering. Let M_e denote a collection of ordered integer pairs (i, j) . A triple (M_e, T_1, T_2) defines the *edit distance mapping* from T_1 to T_2 , where each pair $(i_1, j_1), (i_2, j_2) \in M_e$ satisfies the following properties:

- $i_1 = i_2$ iff $j_1 = j_2$ (one-to-one)
- $t_1[i_1]$ is an ancestor of $t_1[i_2]$ iff $t_2[j_1]$ is an ancestor of $t_2[j_2]$ (ancestor ordering).

The cost of transforming T_1 into T_2 can be expressed through the mapping as

$$\begin{aligned} \gamma(M_e) &= \sum_{(i,j) \in M_e} \gamma(t_1[i] \longrightarrow t_2[j]) \\ &+ \sum_{\{i \nmid j, (i,j) \in M_e\}} \gamma(t_1[i] \longrightarrow \lambda) \\ &+ \sum_{\{j \nmid i, (i,j) \in M_e\}} \gamma(\lambda \longrightarrow t_2[j]) \end{aligned} \quad (3.10)$$

Given a sequence of edit operations S that transforms T_1 into T_2 , there exists a mapping M_e such that $\gamma(M_e) \leq \gamma(S)$. Conversely, given an edit distance mapping M_e , there exists a sequence of edit operations S such that $\gamma(S) = \gamma(M_e)$. Using the above, it can be shown

that

$$D_e(T_1, T_2) = \min_{M_e} \{\gamma(M_e)\} \quad (3.11)$$

where (M_e, T_1, T_2) defines the *edit distance mapping* from T_1 to T_2 . Zhang et al. [146] showed that computing $D_e(T_1, T_2)$ is NP-complete even when the trees are binary and $|\Sigma| = 2$.

3.7.1.2 Constrained and restricted mappings

Adding constraints to the edit distance mapping brings it within the computationally tractable realm. The main constraint imposed is that disjoint subtrees are mapped to disjoint subtrees. Let $T[i]$ denote the subtree rooted at the node with label i and $F[i]$ denote the unordered forest obtained by deleting the node $t[i]$ from $T[i]$. A node $t_1[i]$ is a *proper ancestor* of $t_1[j]$ if $t_1[i]$ lies on the path from the root to $t_1[j]$ and $t_1[i] \neq t_1[j]$. The triple (M_c, T_1, T_2) is called a *constrained edit distance mapping* if,

- (M_c, T_1, T_2) is an edit distance mapping, and
- Given three pairs $(i_1, j_1), (i_2, j_2), (i_3, j_3) \in M_c$, the *least common ancestor* $\text{lca}(t_1[i_1], t_1[i_2])$ is a proper ancestor of $t_1[i_3]$ iff $\text{lca}(t_2[j_1], t_2[j_2])$ is a proper ancestor of $t_2[j_3]$.

The constrained edit distance mappings can be composed. Given two constrained edit distance mappings M_{c_1} from T_1 to T_2 and M_{c_2} from T_2 to T_3 , $M_{c_2} \circ M_{c_1}$ is a constrained edit distance mapping between T_1 and T_3 . Also,

$$\gamma(M_{c_2} \circ M_{c_1}) \leq \gamma(M_{c_1}) + \gamma(M_{c_2}) \quad (3.12)$$

which can be proven using the triangle inequality imposed on the edit operation costs. This leads to the definition of constrained edit distance

$$D_c(T_1, T_2) = \min_{M_c} \{\gamma(M_c)\} \quad (3.13)$$

D_c also satisfies metric properties. Both M_e and M_c deal with mapping between unordered trees. Similar mappings work for forests. We define a *restricted mapping* $M_r(i, j)$ between $F_1[i]$ and $F_2[j]$ as follows:

- $M_r(i, j)$ corresponds to a constrained edit distance mapping between $F_1[i]$ and $F_2[j]$.
- Given two pairs $(i_1, j_1), (i_2, j_2) \in M_c$, $t_1[l_1]$ and $t_1[l_2]$ belong to a common tree in $F_1[i]$ if and only if $t_2[j_1]$ and $t_2[j_2]$ belong to a common tree in $F_2[j]$.

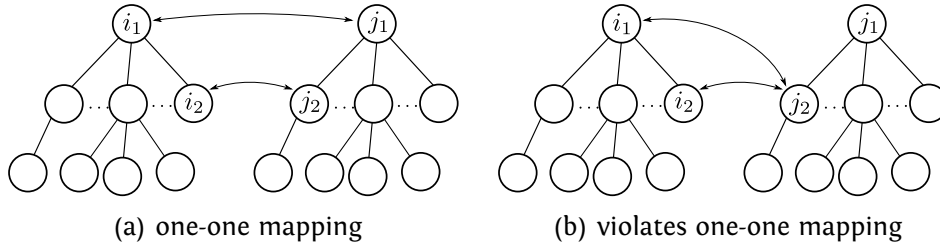


Figure 3.10: Unconstrained TED mappings satisfy the one-to-one mapping. (a) A mapping that satisfies the property. (b) A mapping that violates the property.

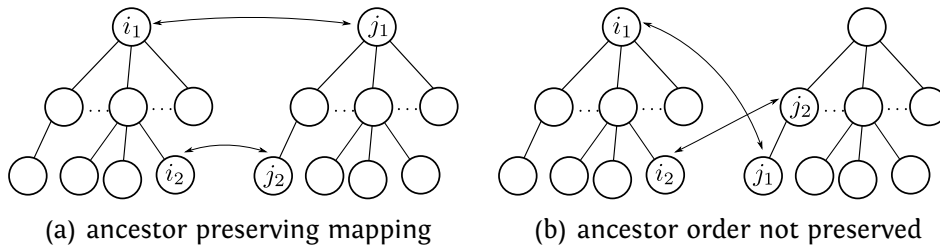


Figure 3.11: Unconstrained TED mappings satisfying the ancestor preservation. (a) A mapping that satisfies the property. (b) A mapping that violates the property.

Essentially, nodes within different trees of F_1 are mapped to nodes lying in different trees of F_2 .

3.7.1.3 Illustrations of the mappings

Figures 3.10 and 3.11 illustrate these mappings using a small example. The mapping in Figure 3.11(b) is one-to-one but does not satisfy the ancestor preservation property, i_1 is ancestor of i_2 but j_1 is child of j_2 . Figure 3.12 illustrates an important property required for a mapping to be constrained, namely disjoint subtrees map to disjoint subtrees. Figure 3.12(b) illustrates a mapping that satisfies the properties of unconstrained tree edit distance mapping but is not a constrained tree edit distance mapping. The node i_3 is a descendant (immediate descendant in this case) of the $lca(i_1, i_2) = I$ but j_3 is not a descendant of the $lca(j_1, j_2) = J$.

3.7.2 Constrained edit distance

We recall the properties of D_c as described by Zhang [144].

Let $t_1[i_1], t_1[i_2], \dots, t_1[i_{n_i}]$ be the children of $t_1[i]$ and $t_2[j_1], t_2[j_2], \dots, t_2[j_{n_j}]$ be the children of $t_2[j]$. Further, let θ denote the empty tree. Then,

$$D_c(\theta, \theta) = 0, \tag{3.14}$$

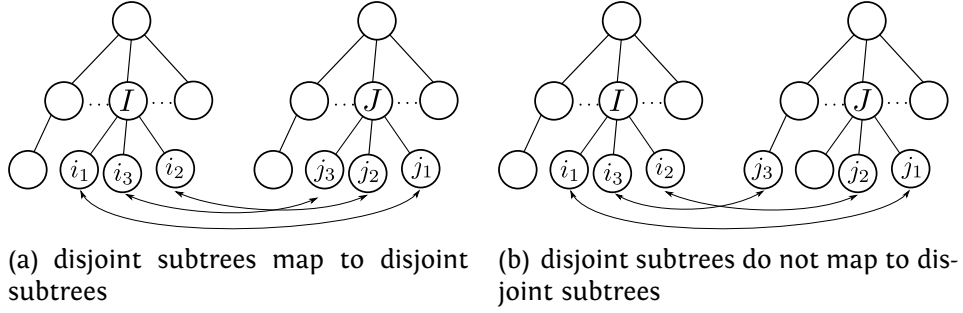


Figure 3.12: Constrained TED mappings satisfying the disjoint subtree mapping. (a) A mapping that satisfies the property. (b) A mapping that violates the property.

$$D_c(F_1[i], \theta) = \sum_{k=1}^{n_i} D_c(T_1[i_k], \theta), \quad (3.15)$$

$$D_c(T_1[i], \theta) = D_c(F_1[i], \theta) + \gamma(t_1[i] \longrightarrow \lambda), \quad (3.16)$$

$$D_c(\theta, F_2[j]) = \sum_{k=1}^{n_j} D_c(\theta, T_2[j_k]), \quad (3.17)$$

$$D_c(\theta, T_2[j]) = D_c(\theta, F_2[j]) + \gamma(\lambda \longrightarrow t_2[j]), \quad (3.18)$$

$$D_c(T_1[i], T_2[j]) = \min \begin{cases} D_c(\theta, T_2[j]) + \min_{1 \leq t \leq n_j} \{D_c(T_1[i], T_2[j_t]) - D_c(\theta, T_2[j_t])\}, \\ D_c(T_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D_c(T_1[i_s], T_2[j]) - D_c(T_1[i_s], \theta)\}, \\ D_c(F_1[i], F_2[j]) + \gamma(t_1[i] \longrightarrow t_2[j]). \end{cases} \quad (3.19)$$

If the cost is not a metric, we need to include one additional case, namely $D_c(F_1[i], F_2[j]) + \gamma(t_1[i] \longrightarrow \lambda) + \gamma(\lambda \longrightarrow t_2[j])$. The distance between two forests is given by

$$D_c(F_1[i], F_2[j]) = \min \begin{cases} D_c(\theta, F_2[j]) + \min_{1 \leq t \leq n_j} \{D_c(F_1[i], F_2[j_t]) - D_c(\theta, F_2[j_t])\}, \\ D_c(F_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D_c(F_1[i_s], F_2[j]) - D_c(F_1[i_s], \theta)\}, \\ \min_{M_r(i,j)} \gamma(M_r(i, j)). \end{cases} \quad (3.20)$$

The minimum restricted mapping may be computed by constructing a weighted bipartite graph in such a way that the cost of the minimum weight maximum matching $MM(i, j)$

is exactly the same as the cost of the minimum restricted mapping $M_r(i, j)$,

$$\min_{M_r(i,j)} \gamma(M_r(i, j)) = \min_{MM(i,j)} \gamma(MM(i, j)) \quad (3.21)$$

3.7.3 Algorithm

Zhang described an algorithm for computing the tree edit distance for labeled unordered trees [144]. It is a dynamic programming based algorithm that follows from the properties discussed in Section 3.7.2. The entry $D(T_1[m], T_2[n])$ in the table with $m = |T_1|$ and $n = |T_2|$ corresponds to the final result. The algorithm computes the distance in $O(|T_1| \times |T_2| \times (deg(T_1) + deg(T_2)) \times \log_2(deg(T_1) + deg(T_2)))$ time in the worst case.

3.8 Tree edit distance with general gap model

Tree edit distance with general gaps is extensively discussed by Xu [139], the material in this section is borrowed from the same.

Consider two join trees T_1 and T_2 . Let their vertex sets be V_1 and V_2 , with $|V_1| = m$, $|V_2| = n$. Let $p(i)$ denote parent of a node i . The distance measure is defined on the preorder traversal of the trees. A relabel cost $r(i, j)$ is included if a node i from T_1 is relabeled to node j in T_2 and a gap cost $g(i)$ is included when a node either starts a gap or extends a previously started gap. Optionally, the costs could differ in the two cases.

Given $1 \leq i' \leq i \leq m$ and $1 \leq j' \leq j \leq n$, the edit distance is defined as,

$$Q[i' \dots i, j' \dots j] = \min \begin{cases} Q[i' \dots i - 1, j' \dots j - 1] + r(i, j), & \text{relabel} \\ Q_{\perp*}[i' \dots i, j' \dots j], & i \text{ is gap node} \\ Q_{*\perp}[i' \dots i, j' \dots j], & j \text{ is gap node} \end{cases}$$

If both i and j exist, then first expression gives the relabel cost, else depending on whether i or j is a gap node, $Q_{\perp*}$ or $Q_{*\perp}$ are used. $Q_{\perp*}$ and $Q_{*\perp}$ are defined based on the gap model as follows.

$$Q_{\perp*}[i' \dots i, j' \dots j] = \min \begin{cases} Q[i' \dots i - 1, j' \dots j] + g(i), \\ Q_{\perp*}[i' \dots i - 1, j' \dots j] + g(i), \\ \min_{j_1 \leq k \leq j} \{Q_{\perp*}[i' \dots p(i), j'_1 \dots k] + \\ Q[p(i) + 1' \dots i - 1, k + 1' \dots j] + g(i)\} \end{cases}$$

$$Q_{*\perp}[i' \dots i, j' \dots j] = \min \begin{cases} Q[i' \dots i, j' \dots j - 1] + g(j), \\ Q_{*\perp}[i' \dots i, j' \dots j - 1] + g(j), \\ \min_{i_1 \leq k \leq i} \{Q_{*\perp}[i'_1 \dots k, j'_1 \dots p(j)] + \\ Q[k + 1' \dots i, p(j) + 1' \dots j - 1] + g(j)\} \end{cases}$$

The cases here are based on whether for the node i , (a) if $p(i)$ is not a gap node, (b) $p(i)$ is a gap node and i is its left child, (c) $p(i)$ is a gap node and i is its right child. Refer Xu [139] for case by case analysis and proof of correctness for these recurrences.

3.8.1 Distance measure

The overall cost C is defined as

$$C = \sum_{(i,j) \in R} r(i, j) + \sum_{n \in G} g(n)$$

The distance measure γ between the trees T_1 and T_2 is defined as,

$$\gamma[T_1, T_2] = \min\{C\}$$

3.8.2 Algorithm

The overall cost is given by the minimum edit distance cost $\gamma = d_e = Q[1 \dots m, 1 \dots n]$ from the algorithm. We compute d_e by incorporating the above-mentioned costs into the recurrences Q , $Q_{\perp*}$ and $Q_{*\perp}$.

Here are the initial conditions.

$$Q[\emptyset, \emptyset] = 0$$

$$Q[1 \dots i, \emptyset] = \infty, 1 \leq i \leq m$$

$$Q[\emptyset, 1 \dots j] = \infty, 1 \leq j \leq n$$

As it is minimization problem, these initial values make sense. Now set

$$Q_{\perp*}[1 \dots i, \emptyset] = \sum_i g(i), 1 \leq i \leq m$$

$$Q_{\perp*}[\emptyset, 1 \dots j] = \infty, 1 \leq j \leq n$$

as it is impossible to match an empty tree with $T_2[1 \dots j]$ such that T_1 ends in a gap

node but $T_1[1 \dots i]$ has a unique matching with the empty tree given by i number of gap nodes. By symmetry we have.

$$Q_{*\perp}[1 \dots i, \emptyset] = \infty, 1 \leq i \leq m$$

$$Q_{*\perp}[\emptyset, 1 \dots j] = \sum_j g(j), 1 \leq j \leq n$$

These expressions directly lead to a dynamic programming algorithm which can be computed in a bottom up fashion. The computation is slow as the algorithm even though polynomial takes $O(m^3n^2 + m^2n^3)$ time.

3.9 Summary

We provide a summary of the background. Section 3.1 provides the theoretical basis for scalar field topology. Section 3.2 describes the topological structures which are used in this thesis as an abstraction of scalar fields. In particular, the persistence diagram described in Section 3.2.1.1 is used directly or indirectly in all the subsequent chapters, i.e. Chapters 4,5,6,7, the merge trees described in Section 3.2.2.1 is the topological structure compared in Chapters 4,5, the extremum graphs described in Section 3.2.3.2 are compared in Chapter 6 and a time-varying extension is provided in Chapter 7. Topological simplification described in Section 3.3 is extensively used to remove topological noise from all the structures before comparing them. Metric graphs described in Section 3.4 form the basis for comparing extremum graphs described in Chapter 6. Properties of comparison measures discussed in Section 3.5 provides us a set of criteria to aim for while defining new comparison measures. Comparison measures discussed in Section 3.6 either form baseline cases to compare our new measures or form the building blocks of the new measures. Tree edit distance described in Section 3.7 forms the basis MTED and LMTED described in Chapters 4,5. The tree edit distance with general gap model described in Section 3.8 forms the basis of OTED described in Chapter 4.

Chapter 4

Global edit distance for merge trees (OTED) and (MTED)

In this chapter we define and describe two comparison measures for merge trees. The ordered merge tree edit distance OTED [119] and the unordered merge tree edit distance MTED [120]. We also discuss various properties and applications of these two comparison measures.

4.1 Introduction

Defining and computing a tree edit distance that allows or disallows for gaps is a well-studied problem. In case gaps are allowed, the computation has been shown to be NP-hard for arbitrary trees. However, for labeled binary trees a polynomial time, dynamic programming-based algorithm exists [139]. We focus on merge trees, which we assume to constitute a special subset of labeled binary trees. Further, we are interested in scenarios where the scalar functions being compared are not significantly different from each other. For example, functions from consecutive time-steps of time-varying data or a function compared against another obtained via a minor perturbation. Given the above assumptions, our aim is to design an effective distance measure using topological persistence. This leads to the ordered merge tree edit distance OTED. OTED is an adaptation of Xu's algorithm [139] for computing distance between ordered labeled binary trees to the case of the general subtree gap model that preserve the merge tree structure. The general subtree gap model allows for interior nodes to be inserted / deleted while retaining the child nodes.

However, this method has multiple shortcomings:

1. The gap model is too general. In the case of merge trees, we require a constrained

version that considers gaps as persistence pairs in order to preserve the structural integrity of the tree. These pairs depend on function values. So, the constraints are ad hoc, difficult to express directly and to incorporate into the dynamic programming based algorithm that is used to compute the measure.

2. The above-mentioned pairs are not stable under perturbations to the scalar function.
3. Merge trees constructed on real world data are not necessarily binary trees.
4. Absence of a natural left-to-right ordering of children of a node in the merge tree. The algorithm requires such an ordering, random or canonical orderings lead to instabilities.
5. The running time of the algorithm is approximately $O(n^5)$, where n is the number of nodes in the tree. This is very slow for practical applications.

To alleviate these, we propose a different approach by adapting of the constrained unordered tree edit distance [144], but is a significant modification that caters to merge trees. Individual edits correspond to topological features. The edit operations may be subsequently studied for a fine grained analysis.

4.2 Contributions

4.2.1 OTED

We adapt the ordered tree edit distance from Xu [139] assuming merge trees to be ordered rooted binary trees to design OTED [119]. The important contributions include,

1. A simple comparison measure for merge trees.
2. A simple cost model based on topological persistence.
3. Applications to periodicity detection and symmetry detection in 2D scalar fields.

4.2.2 MTED

We adapt the constrained unordered tree edit distance [144] to design MTED [120]. The important contributions include,

1. An intuitive and mathematically sound cost model for the individual edit operations.
2. A proof that the distance measure is a metric under the proposed cost model.

3. A computational solution to handle instabilities.
4. Experiments to demonstrate the practical value of the distance measure using various applications – 2D time-varying data analysis by detecting periodicity, summarization to support visualization of 3D time-varying data, detection of symmetry and asymmetry in scalar fields, study of topological effects of subsampling and smoothing, and shape matching.

In addition, we describe a comprehensive set of validation experiments that are designed to help understand the properties of the measure.

4.3 Global merge tree edit distances

In this section we describe the formulation of the two global distances, followed by the algorithms, properties, and applications.

4.3.1 Ordered merge tree edit distance OTED

Xu [139] describes a distance measure with a focus on correctness and worst case runtime analysis for computing the measure. We describe a distance measure that is also based on edit distances with a general subtree gap similar to Xu. However, our focus is on applicability to merge trees. The edit operations are (a) Relabel nodes, (b) Insert a subtree or gap, and (c) Delete a subtree or gap. A gap is defined as collection of nodes that are present in one tree but not in the other. We do not distinguish between starting gaps and continuing gaps.

A key property of the join tree is that it supports only a restricted set of insert/delete operations. Consider a minimum-saddle pair (m, s) . Let l denote the parent of s and m' denote the child of s that is neither equal to m nor ancestor of m . If s is deleted, then m should also be deleted, and vice-versa, also m' is paired with l . But deletion of s does not result in the deletion of the entire subtree rooted at s .

4.3.1.1 Definition

Consider two join trees T_1 and T_2 representing scalar functions f_1 and f_2 , whose ranges are normalized to lie within $[0,1]$. Let their vertex sets be V_1 and V_2 , with $|V_1| = m, |V_2| = n$. The distance measure is defined on the preorder traversal of the trees. A relabel cost $r(i, j)$ is included if a node i from T_1 is relabeled to node j in T_2 and a gap cost $g(i)$ is included when a node part of a gap.

Given $1 \leq i' \leq i \leq m$ and $1 \leq j' \leq j \leq n$, the edit distance is defined as

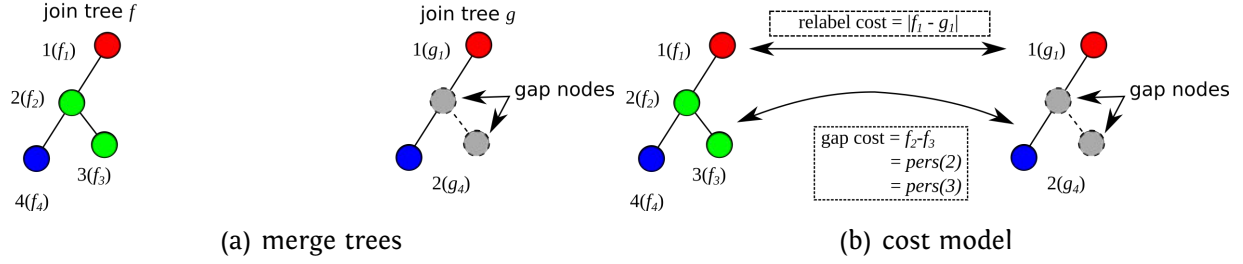


Figure 4.1: Two merge trees with the cost model.

$$\mathcal{D}[i'..i, j'..j] = \min \begin{cases} \mathcal{D}[i'..i-1, j'..j-1] + r(i, j), \\ \mathcal{D}_{\perp*}[i'..i, j'..j], \\ \mathcal{D}_{*\perp}[i'..i, j'..j], \end{cases}$$

If nodes corresponding to both i and j exist, then first expression gives the relabel cost, else depending on whether i or j is a gap node, $\mathcal{D}_{\perp*}$ or $\mathcal{D}_{*\perp}$ are used. $\mathcal{D}_{\perp*}$ and $\mathcal{D}_{*\perp}$ are defined based on the gap model and details are provided in Xu [139].

We propose to utilise well-known properties of merge trees to define appropriate costs for the edit operations (see Figure 4.1).

Relabel cost. If the node $i \in V_1$ is matched with node $j \in V_2$, define the cost of relabelling i to j as $r(i, j) = |f_1(i) - f_2(j)|$

Add/delete gap cost. We define the cost of a gap node irrespective of whether it is a minimum or a saddle, as the persistence of the feature to which the node belongs to: $g(m) = g(s) = \text{pers}(m) = \text{pers}(s)$.

The overall cost is given by the minimum edit distance cost $d_e = \mathcal{D}[1..m, 1..n]$ from the algorithm. We compute d_e by incorporating the above-mentioned costs into \mathcal{D} and computing it in a bottom up fashion.

4.3.1.2 Algorithm

The algorithm and computation follows [139] with the customised cost model. The worst case running time is $O(m^3n^2 + m^2n^3)$ where n, m are the sizes of the trees.

4.3.2 Unordered merge tree edit distance MTED

OTED is based on the assumption that the trees are binary and can be ordered. Absence of a natural left-to-right ordering of children of a node in the merge tree, high run time of $O(n^5)$, and existence of more than two children limits the utility of OTED.

To handle such scenarios we describe a new tree edit distance that is appropriate for comparing merge trees, discuss its properties, and an algorithm for computing the distance measure.

4.3.2.1 Comparing merge trees

Our proposed measure is based on a variant of tree edit distance that applies to unordered general trees as opposed to ordered binary trees. This variant is appropriate because

- Merge trees are unordered trees.
- Merge trees are not binary in general.
- Persistence pairs represent topological features. So, it is natural that the edit operations are defined in terms of persistent pairs.
- The pairs do not fit into any subtree gap model that has been studied in the literature.

Consider the properties of edit distance mapping mentioned in Section 3.7.1.1, but now in the context of merge trees. The one-to-one property is applicable but ancestor ordering might not hold in all cases. Small perturbations in the function value may result in swaps similar to rotations in AVL or red-black trees [29, Chapter 13], which violate the ancestor ordering. Such violations also result in instabilities *i.e.*, cause significant fluctuations in the distance (see Section 4.3.2.4). Computing the edit distance with the ancestor order preserving mappings is already infeasible. Removing that constraint will make the computation more difficult. We introduce a stability parameter to ensure that ancestor order preserving mappings are identified in practically all cases. More details on this computational solution to handling instabilities can be found in Section 4.3.2.4. This solution does discard some mappings and may lead us away from the optimum solution. But, the stabilization ensures that the mapping remains meaningful and helps reduce the search space thereby making the problem tractable.

To summarize, D_c between unordered trees with suitable modifications seems to be a good candidate for comparing merge trees. In this section, we describe one such distance measure and demonstrate its use in the following section. The additional constraint of mapping disjoint subtrees to disjoint subtrees may seem limiting. Also, $D_e(T_1, T_2) \leq D_c(T_1, T_2)$, which implies that the constrained edit distance may not be optimal in many cases. But, we observe that, in practice, it is not as limiting and gives good results in many applications.

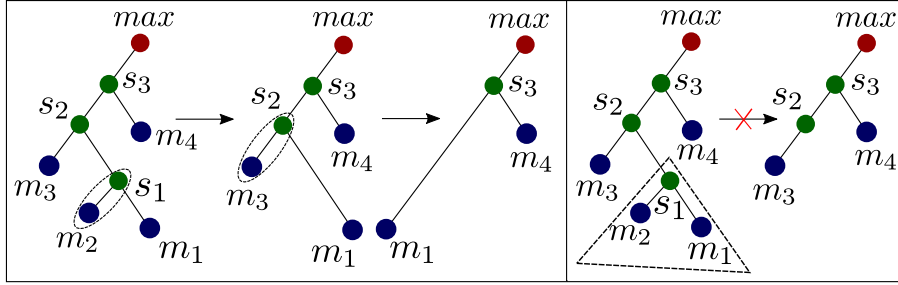


Figure 4.2: Permitted and forbidden edit operations. (left) A gap is introduced by removing a persistence pair. (right) An edit operation that is permitted for generic trees but is invalid for a join tree. Nodes and arcs are repositioned to improve the tree layout.

4.3.2.2 Cost models

The edit distance mapping M_e and the constrained edit distance mapping M_c need to be suitably modified so that they are applicable for comparing merge trees. We begin by considering the edit operations as applicable to merge trees together with appropriate cost models. The literature on tree edit distances study generic trees and hence do not describe particular cost models. The following discussions focus on join trees but all results hold for split trees also.

Tree edit operations on the join tree need to preserve the structural integrity of the join tree. This reduces the number of operations, say insertions and deletions. Consider a min-saddle pair (m_2, s_1) in Figure 4.2. If s_1 is deleted, then its pair m_2 should also be deleted, and vice-versa. After deletion, m_1 is adjacent to s_2 . But deletion of s_1 does not necessarily require that the entire subtree rooted at s_1 be deleted. In fact, deleting the entire subtree may not result in a valid join tree as illustrated in Figure 4.2. In this particular illustration, we consider the pairing imposed by the persistence. But, in general, we may consider other pairings based on say volume, hyper-volume, etc.

Gaps in the join tree can be represented as a collection of min-saddle pairs. In Figure 4.2, we can transform the first tree into the last tree by deleting the pairs $\{(m_2, s_1), (m_3, s_2)\}$. We propose two cost models that capture the preservation of topological features and are applicable for join trees. Consider nodes $p \in T_1$ and $q \in T_2$. Then p and q are creators or destroyers of topological features in T_1 and T_2 , respectively. Let the birth and death times of these features be (b_p, d_p) and (b_q, d_q) , respectively. These birth-death pairs correspond to points in the persistence diagrams. Alternatively, they are represented as closed intervals $[b_p, d_p]$ and $[b_q, d_q]$ in a persistence barcode.

L_∞ cost C_W

$$\gamma(p \longrightarrow q) = \min \left\{ \begin{array}{l} \max(|b_q - b_p|, |d_q - d_p|), \\ \frac{(|d_p - b_p| + |d_q - b_q|)}{2} \end{array} \right. \quad (4.1)$$

$$\gamma(p \longrightarrow \lambda) = \frac{|d_p - b_p|}{2} \quad (4.2)$$

$$\gamma(\lambda \longrightarrow q) = \frac{|d_q - b_q|}{2} \quad (4.3)$$

This cost model is based on the bottleneck and Wasserstein distances. Note that the insert / delete cost is based on the L_∞ -distance of the points p (or q) from the diagonal in the persistence diagram. The relabel cost is the minimum of the L_∞ -distance between the points p and q and the sum of the L_∞ -distance from the points p (or q) to the diagonal. This corresponds to the scenario where transforming p to q ($p \longrightarrow q$) by deleting p and inserting q ($p \longrightarrow \lambda$ and $\lambda \longrightarrow q$) has a lower cost in some cases. Figure 4.3 shows how these costs can be derived from the persistence diagram when there is no overlap in the barcodes and when there is overlap between the barcodes.

Overhang cost C_O

$$\gamma(p \longrightarrow q) = \min \left\{ \begin{array}{l} |b_q - b_p| + |d_q - d_p|, \\ |d_p - b_p| + |d_q - b_q| \end{array} \right. \quad (4.4)$$

$$\gamma(s \longrightarrow \lambda) = |d_p - b_p| \quad (4.5)$$

$$\gamma(\lambda \longrightarrow t) = |d_q - b_q| \quad (4.6)$$

This cost model is based on the overlap of the barcodes or the intervals. We consider the lengths of the overhang or the non-overlapping section to determine the costs. Consider $p \longrightarrow \lambda$, the interval corresponding to p is given by $[b_p, d_p]$ with length $|d_p - b_p|$ and the interval corresponding to λ is \emptyset with length 0. Since there is no overlap, the cost is $|d_p - b_p| + 0 = |d_p - b_p|$. The cost of $\lambda \longrightarrow q$ can be derived similarly. Let us now consider the cost of $p \longrightarrow q$. If there is an overlap, we discard the overlap and obtain $|b_q - b_p| + |d_q - d_p|$. If there is no overlap then the cost is equal to $|d_p - b_p| + |d_q - b_q|$. The minimum of the two expressions is the relabel cost. The barcodes are shown in the fourth quadrant of the persistence diagrams in Figure 4.3.

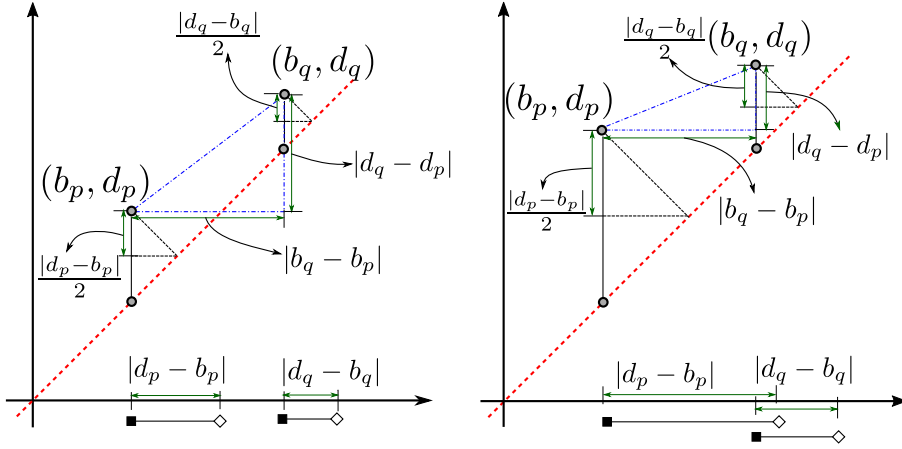


Figure 4.3: Illustration of the cost models when there is no overlap in the barcodes (left) and when there is overlap (right). We distinguish between birth and death events in the barcode by using different glyphs for the start and the end of the intervals.

4.3.2.3 Properties

Metric property enables us to study the space of all the trees, compute the mean, and also compose transformations between merge trees. From Sections 3.7 and 3.7.1.2, we know that if the cost model satisfies the metric property then the distance measure is also a metric. We now prove the metric properties for our cost model.

The overhang cost is similar to symmetric difference, which is a well-known metric [70]. We now prove that the L_∞ cost C_W is a metric.

C_W is a metric

We show that the cost C_W is equal to the Wasserstein distance between two corresponding persistence diagrams. Let N denote the set of all nodes in the merge trees and λ denote a node corresponding to the null character. We define a mapping $\mathcal{M} : N \cup \{\lambda\} \rightarrow Dgm$, where Dgm is set of all persistence diagrams, as follows:

1. $\forall p \in N, \mathcal{M}(p) = \{(b_p, d_p)\} \cup \{(x, x), x \geq 0\}$,
2. $\mathcal{M}(\lambda) = \{(x, x), x \geq 0\}$.

Define the distance on the set $N \cup \{\lambda\}$ as the Wasserstein distance of the first order *i.e.*, given $p, q \in N \cup \{\lambda\}$

$$d(p, q) = W_1(\mathcal{M}(p), \mathcal{M}(q))$$

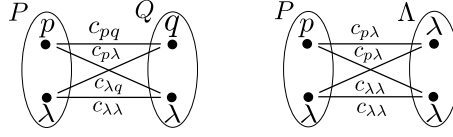


Figure 4.4: The cost of edit operations can be reformulated as the weight of a minimum weight maximum matching in a bipartite graph. Bipartite graph for a relabel operation $p \rightarrow q$ (left) and a delete operation (right).

Now, the cost C_W can be rewritten as

$$\gamma(p \rightarrow q) = W_1(\mathcal{M}(p), \mathcal{M}(q)) \quad (4.7)$$

$$\gamma(p \rightarrow \lambda) = W_1(\mathcal{M}(p), \mathcal{M}(\lambda)) \quad (4.8)$$

$$\gamma(\lambda \rightarrow q) = W_1(\mathcal{M}(\lambda), \mathcal{M}(q)). \quad (4.9)$$

Since the Wasserstein distance $W_1(\cdot, \cdot)$ between persistence diagrams is known to be a metric [133, Chapter 6], C_W is also a metric. However, this proof of the metric property is for general distributions. We have an alternative proof for merge trees from first principles with the aim to better understand the cost.

C_W is a metric : proof from first principles

Non-negativity and symmetry follows by definition because C_W is based on sum, max, min of absolute values. To prove the triangle inequality, we first reformulate the cost of the edit operations as the weight of a minimum weight maximum matching. The matching is defined in a bipartite graph. Nodes of the bipartite graph consists of the merge tree nodes together with an equal number of copies of λ . We collect the nodes of the graph to construct sets of the form $P = \{p, \lambda\}$ and a special multiset $\Lambda = \{\lambda, \lambda\}$, see Figure 4.4. All pairs of nodes from different multisets are connected by an edge. The edge weight c is given by the L_∞ distance between the corresponding points in the persistence diagram:

$$c_{pq} = L_\infty(p, q) = \max(|b_q - b_p|, |d_q - d_p|) \quad (4.10)$$

$$c_{p\lambda} = L_\infty(p, \lambda) = \frac{|d_p - b_p|}{2} \quad (4.11)$$

$$c_{\lambda q} = L_\infty(\lambda, q) = \frac{|d_q - b_q|}{2} \quad (4.12)$$

$$c_{\lambda\lambda} = L_\infty(\lambda, \lambda) = 0 \quad (4.13)$$

The cost of the edit operations is equal to the cost of the minimum weight maximum

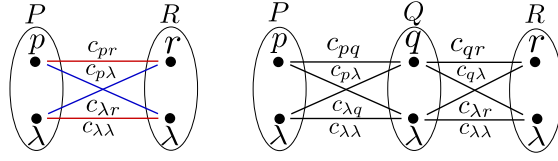


Figure 4.5: The cost function satisfies triangle inequality. (left) The blue or the red matching may be the minimum weight maximum matching that corresponds to the cost of the edit operation. (right) The matching between P and R is a composition of matching between P, Q and Q, R . The inequality can be proved via case analysis by considering all possible compositions.

matching MM in this bipartite graph. In Figure 4.4, one of the two matchings will determine the cost of the edit operation.

$$\gamma(p \longrightarrow q) = MM(P, Q) = \min \begin{cases} c_{pq} + c_{\lambda\lambda}, \\ c_{p\lambda} + c_{\lambda q} \end{cases} \quad (4.14)$$

$$\gamma(p \longrightarrow \lambda) = MM(P, \Lambda) = \min \begin{cases} c_{p\lambda} + c_{\lambda\lambda}, \\ c_{\lambda\lambda} + c_{p\lambda} \end{cases} \quad (4.15)$$

$$\gamma(\lambda \longrightarrow q) = MM(\Lambda, Q) = \min \begin{cases} c_{\lambda\lambda} + c_{\lambda q}, \\ c_{\lambda q} + c_{\lambda\lambda} \end{cases} \quad (4.16)$$

Consider three multisets as shown in Figure 4.5. Using the above construction, we prove triangle inequality by considering the two cases, namely when the minimum weight matching is equal to either the red or blue matching.

Case RED: $MM(P, R)$ is given by the red matching. The cost of the relabel $\gamma(p \longrightarrow r) = c_{pr} + c_{\lambda\lambda} = c_{pr}$. Two different paths from p lead to r , $p \longrightarrow q \longrightarrow r$ and $p \longrightarrow \lambda \longrightarrow r$. Consider the first path,

$$\gamma(p \longrightarrow r) = c_{pr} = L_\infty(p, r) \leq L_\infty(p, q) + L_\infty(q, r) \quad (4.17)$$

$$= c_{pq} + c_{qr} \quad (4.18)$$

$$= \gamma(p \longrightarrow q) + \gamma(q \longrightarrow r) \quad (4.19)$$

Now, let us consider the second path. If $c_{pr} \leq c_{p\lambda} + c_{\lambda r}$ then $c_{pr} \leq c_{p\lambda} + c_{\lambda q} + c_{q\lambda} + c_{\lambda r}$ and we are done. Else, we have two sub-cases

$$c_{pr} > c_{p\lambda} + c_{\lambda q} + c_{q\lambda} + c_{\lambda r} > c_{p\lambda} + c_{\lambda r}, \text{ or} \quad (4.20)$$

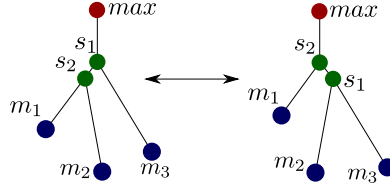


Figure 4.6: Illustrating instabilities. Since the difference in the function values between s_1 and s_2 is small, a slight perturbation leads to a change in the structure of the tree, which affects the distance measure.

$$c_{pr} > c_{p\lambda} + c_{\lambda r} \text{ but } c_{pr} < c_{p\lambda} + c_{\lambda q} + c_{q\lambda} + c_{\lambda r}. \quad (4.21)$$

In both sub-cases, we have a matching with weight $MM'(P, R) = c_{p\lambda} + c_{\lambda r} \leq MM(P, R) = c_{pr} + c_{\lambda\lambda}$, which contradicts our assumption.

The cost $\gamma(\lambda \rightarrow \lambda) = c_{\lambda\lambda} = 0$. Both paths via Q , $\lambda \rightarrow \lambda \rightarrow \lambda$ and $\lambda \rightarrow q \rightarrow \lambda$, should necessarily have a non-zero total cost. So, the inequality holds trivially.

Case BLUE: $MM(P, R)$ is given by the blue matching. The cost of the relabel is equal to the sum $c_{p\lambda} + c_{\lambda r}$. We consider the two weights individually.

Two paths from $p \in P$ lead to $\lambda \in R$ via a node in Q , $p \rightarrow \lambda \rightarrow \lambda$ and $p \rightarrow q \rightarrow \lambda$. Similarly, two paths from $\lambda \in P$ lead to $r \in R$, $\lambda \rightarrow \lambda \rightarrow r$ and $\lambda \rightarrow q \rightarrow r$.

In both cases, triangle inequality holds trivially for the first path via Q , $c_{p\lambda} \leq c_{p\lambda} + c_{\lambda\lambda}$ and $c_{\lambda r} \leq c_{\lambda\lambda} + c_{\lambda r}$. We need to show that the inequality holds for the second paths as well. From the persistence diagram, we observe that $L_\infty(p, \lambda) \leq L_\infty(p, q) + L_\infty(q, \lambda)$ for all q , even when q lies on the perpendicular from p onto the diagonal. So, $c_{p\lambda} \leq c_{pq} + c_{q\lambda}$. A similar argument can be used to show that $c_{\lambda r} \leq c_{\lambda q} + c_{qr}$.

The red and blue cases together imply that the cost C_W satisfies the triangle inequality and is therefore a metric. It follows that the tree edit distance measure D is also a metric.

4.3.2.4 Handling instabilities

Saikia et al. [102] discuss two kinds of instabilities, *vertical* and *horizontal*, that affect branch decompositions and hence the distance measures. Figure 4.6 illustrates how horizontal instability can occur. In our case, the horizontal stability has a more drastic effect on the measure because

- It changes the persistence pairing, which in turn affects the cost.
- It also changes the subtrees thereby affecting the matching found by the algorithm.

We employ a strategy similar to the one used for branch decompositions by Thomas and Natarajan [124] and apply it to merge trees. We introduce a stability parameter ε and use

it to determine how to merge simple saddles into a multi-saddle where instabilities occur. We merge the saddles in a bottom-up manner as follows. Begin from the lower saddle s_l that is further from the root and merge it into a higher saddle s_h that is nearer to the root if the function difference $|f(s_h) - f(s_l)| < \varepsilon$. Repeat this process until none of the saddles satisfy the merging condition. In the implementation, the multi-saddle is represented by the saddle with the highest persistence. We compute the distance between the stabilized trees. Optionally, a fixed value may be added to the final distance to incorporate the cost incurred due to the stabilization. In Section 4.4.2.2, we experimentally analyze how varying the stability parameter ε affects the distance measure.

4.3.2.5 Algorithm

We adapt Zhang’s algorithm [144] with the edit costs discussed in Section 4.3.2.2 to compute the tree edit distance between merge trees. The input to this algorithm is a pair of merge trees that are stabilized using the strategy described in Section 4.3.2.4.

We describe the algorithm 1 in some detail. Line 2 initializes the distance between two empty trees to 0. The loops spanning lines 3–6 and 7–10 fill the table entries corresponding to the distances between the empty tree and all trees and forests. The nested loops spanning lines 11 – 16 fill the entries that correspond to distances between non-empty forests and trees. The entry $D(T_1[m], T_2[n])$ in the table with $m = |T_1|$ and $n = |T_2|$ corresponds to the final result. The runtime analysis is as described by Zhang [144]. This algorithm has a better running time than the one proposed by Xu [139].

The computation proceeds in a bottom up manner. Distances for the subtrees are computed and stored in a table. These are next used for computing distances between subtrees at higher levels of the merge trees. This proof of concept implementation does not include code and memory optimizations for efficiently computing and storing the dynamic programming tables. We use the simple Kuhn-Munkres algorithm [67] for computing $MM(i, j)$. We still observe reasonable running times for most of the datasets as reported in the individual experiments in the following section.

4.4 Applications

We describe various applications which find the two global merge tree edit distances as their underlying foundation in this section.

Algorithm 1: TreeEditDistance [144]

Data: Merge trees T_1, T_2 .

Result: $D(T_1[i], T_2[j])$, where $1 \leq i \leq |T_1|$, $1 \leq j \leq |T_2|$

1 **begin**

2 $D(\theta, \theta) = 0$

3 **for** $i = 1$ **to** $|T_1|$ **do**

4 $D(F_1[i], \theta) = \sum_{k=1}^{n_i} D(T_1[i_k], \theta)$

5 $D(T_1[i], \theta) = D(F_1[i], \theta) + \gamma(t_1[i] \rightarrow \lambda)$

6 **end**

7 **for** $j = 1$ **to** $|T_2|$ **do**

8 $D(\theta, F_2[j]) = \sum_{k=1}^{n_j} D(\theta, T_2[j_k])$

9 $D(\theta, T_2[j]) = D(\theta, F_2[j]) + \gamma(\lambda \rightarrow t_2[j])$

10 **end**

11 **for** $i = 1$ **to** $|T_1|$ **do**

12 **for** $j = 1$ **to** $|T_2|$ **do**

13 $D(F_1[i], F_2[j]) =$

$$\min \begin{cases} D(\theta, F_2[j]) + \min_{1 \leq t \leq n_j} \{D(F_1[i], F_2[j_t]) - D(\theta, F_2[j_t])\}, \\ D(F_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D(F_1[i_s], F_2[j]) - D(F_1[i_s], \theta)\}, \\ \min_{MM(i,j)} \gamma(MM(i,j)). \end{cases}$$

$$D(T_1[i], T_2[j]) =$$

$$\min \begin{cases} D(\theta, T_2[j]) + \min_{1 \leq t \leq n_j} \{D(T_1[i], T_2[j_t]) - D(\theta, T_2[j_t])\}, \\ D(T_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D(T_1[i_s], T_2[j]) - D(T_1[i_s], \theta)\}, \\ D(F_1[i], F_2[j]) + \gamma(t_1[i] \rightarrow t_2[j]). \end{cases}$$

14 **end**

15 **end**

16 **end**

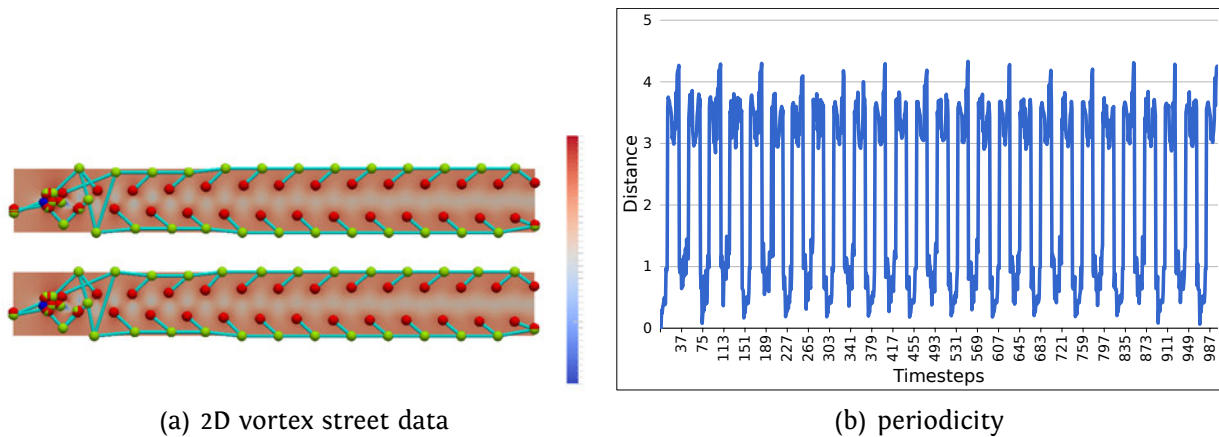


Figure 4.7: Merge trees. (a) Time-step 0 (top) and 74 (bottom) of the flow around a cylinder simulation. The split tree along with the critical points is overlaid. (b) The plot shows distances computed between time step 0 and time steps 0-1000. The timesteps and the distances are indicated on the x-axis and the y-axis respectively. From this plot, a time period of 74-75 can be identified.

4.4.1 Applications of OTED

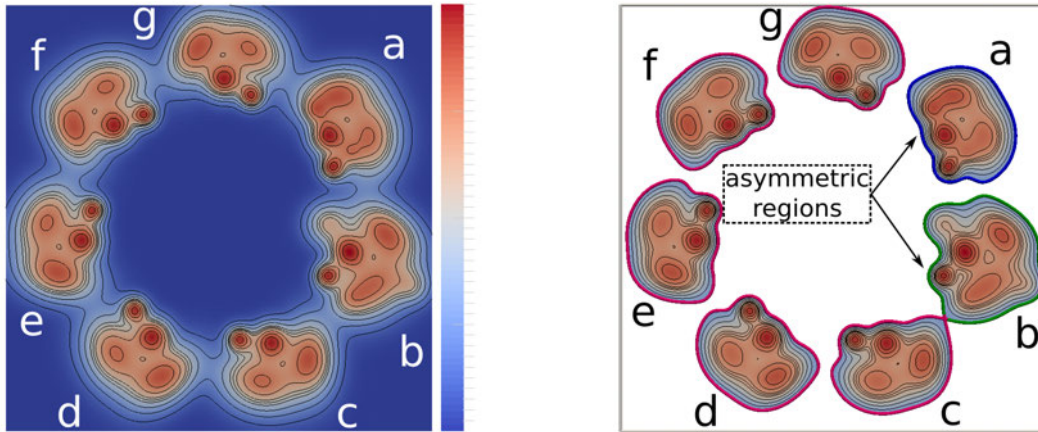
We demonstrate the potential utility of OTED by applying it to analyse time-varying data and to study symmetry in scalar fields.

4.4.1.1 Periodicity in time-varying data

To demonstrate the utility of the measure, we use Bénard von Kármán vortex street data-set formed by flow around a cylinder [135]. Figure 4.7(a) shows few timesteps of the data-set. The data-set is known to exhibit a periodicity of 75 and [88] detects another period of 38 along with 75. To identify periodicity, we compare the split tree of each time step with all 1000 timesteps of the data-set. We plot the distances obtained by our measure in Figure 4.7(b) and observe a periodicity between 74 and 75.

4.4.1.2 Symmetry detection

We use a synthetic data-set (see Figure 4.8) that contains seven regions out of which five are symmetrical and the other two (named as ‘a’ and ‘b’) are slightly perturbed to cause asymmetry. We apply our distance measure to compare each subtree corresponding a region with the other and we are able to distinguish between symmetric and asymmetric regions. Region *d*, for example has a distance close to 0 with regions *c, e, f, g*; 0.53 with region *b* and 0.45 with region *a*. This is consistent with the premise upon which the data is generated.



(a) 2D scalar field

(b) symmetric regions

Figure 4.8: 2D synthetic data-set generated by sum of gaussians, along with the symmetric and asymmetric regions identified. The symmetric regions are highlighted using the same boundary color, magenta and the asymmetric regions are shown using arrows.

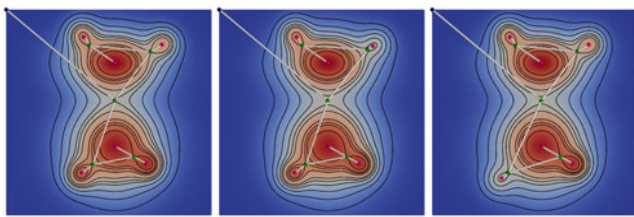
4.4.2 Applications of MTED

We demonstrate the utility of MTED by applying it to analyze time-varying data, to study symmetry in scalar fields, for summarizing data, and for shape matching. We use the Recon library [37] to compute merge trees, the algorithm described in Section 4.3.2.5 to compute the tree edit distance between the merge trees, and Paraview [3] together with the Topology ToolKit TTK [128] to generate renderings of the merge trees together with the scalar fields. We uniformly use the L_∞ cost C_W (4.3.2.2). All experiments were performed on a machine with an Intel Xeon CPU with 8 cores running at 2.0 GHz and 16 GB main memory.

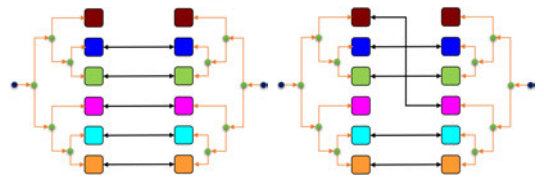
4.4.2.1 Understanding the distance measure

We construct three synthetic datasets to understand the difference between the tree edit distance D and other well known distances between topological structures. The scalar functions f_1, f_2, f_3 are sums of gaussians whose extrema are fixed in space. The scalar values change in a controlled manner for the three functions so that the values at the extrema increase / decrease monotonically as we step from f_1 to f_2 to f_3 . We compute the tree edit distances together with the corresponding mapping for each pair (f_1, f_2) and (f_2, f_3) .

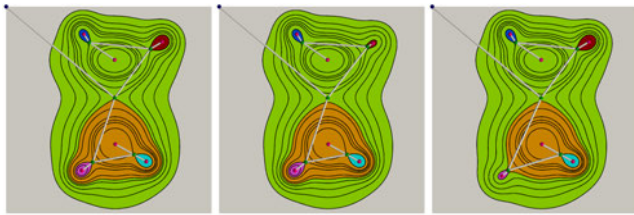
Figure 4.9 shows the three scalar functions f_1, f_2, f_3 . We observe in Figure 4.9(d) that D



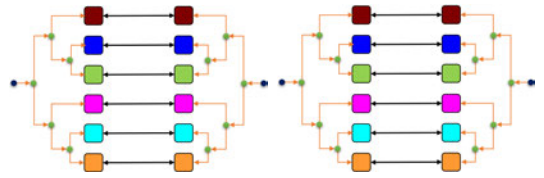
(a) Three scalar field f_1, f_2, f_3



(b) Mapping determined by W_1 for (f_1, f_2) and (f_2, f_3)



(c) Merge tree driven segmentation for each field.



(d) Mapping determined by D for (f_1, f_2) and (f_2, f_3)

Figure 4.9: Comparing mappings established by tree edit distance measure D and Wasserstein distance W_1 . (a) Three scalar functions f_1, f_2, f_3 in the synthetic data set. (c) Regions corresponding to the maxima and arcs incident on them in the merge trees of f_1, f_2, f_3 . Each region is assigned a unique color. (b) Mapping determined by W_1 between (f_1, f_2) and between (f_2, f_3) . (d) Mapping determined by the tree edit distance D between (f_1, f_2) and between (f_2, f_3) . Merge tree nodes and their corresponding spatial regions have the same color.

establishes intuitively correct mappings. The mappings also preserve the tree hierarchy. On the other hand, the Wasserstein distance W_1 (Figure 4.9(b), left) maps the brown regions to the null character. The tree edit distance prefers the relabel over a sequence of delete-insert operations. The reason W_1 does not find the correspondence between the two brown nodes is because their birth-death intervals do not overlap. As a result, these nodes are mapped to the null character *i.e.*, inserted or deleted. The intervals corresponding to the two nodes in question are $[1.17, 1.39]$ in f_1 and $[1.06, 1.08]$ in f_2 . We also see from Figure 4.9(b) that W_1 maps the brown region to the magenta region, thereby mapping nodes that lie within different subtrees. This also causes a pair of nodes being mapped to the null character. The tree edit distance D is constrained to map disjoint subtrees to disjoint subtrees and establishes a better mapping. To summarize, D in general establishes mappings that are better than D_B and W_1 because it is aware of the structure of the merge tree and preserves the hierarchy captured in the tree.

4.4.2.2 Comparison with other distance measures

We compare the proposed tree edit distance measure D with existing measures such as bottleneck distance D_B and Wasserstein distance W_1 via computational experiments on the 2D Bénard-von Kármán vortex street dataset [135]. Figure 4.11(a) shows a few timesteps of the data, which represents flow around a cylinder. The dataset contains the velocity magnitude on a 400×50 grid over 1001 timesteps. Each split tree contains approximately 55 – 65 nodes. We calculate D and plot it together with the bottleneck and Wasserstein distance, see top row of Figure 4.10. The tree edit distance D is always greater than W_1 and D_B . Indeed, D is likely to be more discriminative than W_1 and D_B because it incorporates the structure of the merge tree in addition to the persistence pairs.

We also compute and plot D for increasing values of the stability parameter ε . The values of ε are reported as a percentage of the maximum persistence of the particular dataset. While there are some anomalies for small values of ε , in general we observe in Figure 4.10 that with increase in ε , D tends towards W_1 . For a high enough value of ε , D becomes almost equal to W_1 . The reason for this behavior is that the bottleneck/Wasserstein distance does not consider the structure of the trees. Increasing the stability parameter transforms the tree to become more like a bush. Finally, all the nodes become children of the root thereby simplifying and eliminating the tree structure. Varying ε from 0 – 5% results in a decrease of up to 25 nodes in the split tree. Further increasing ε led to an additional reduction by only 1 – 2 nodes. We observe this trend in the distance plots also.

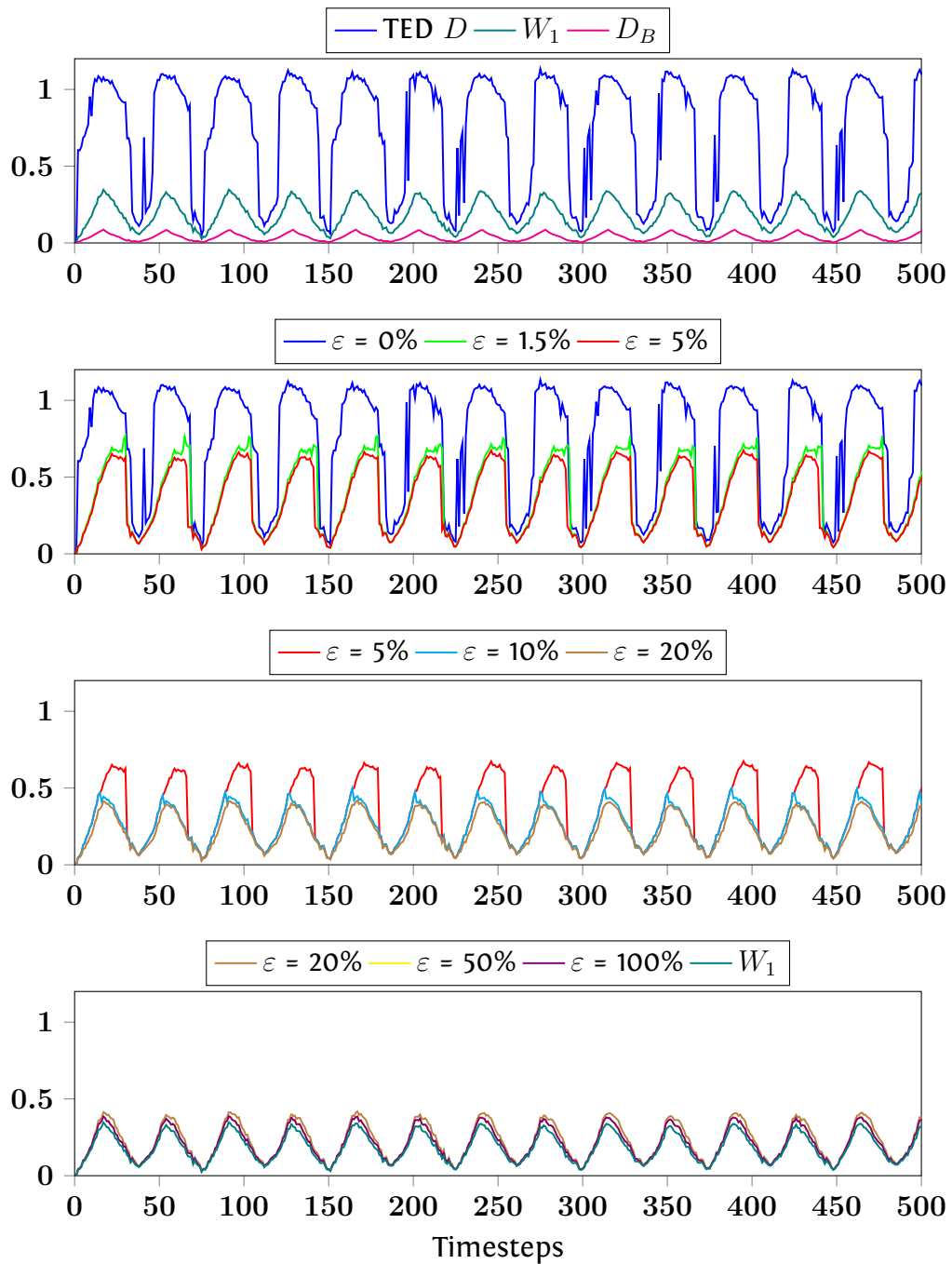
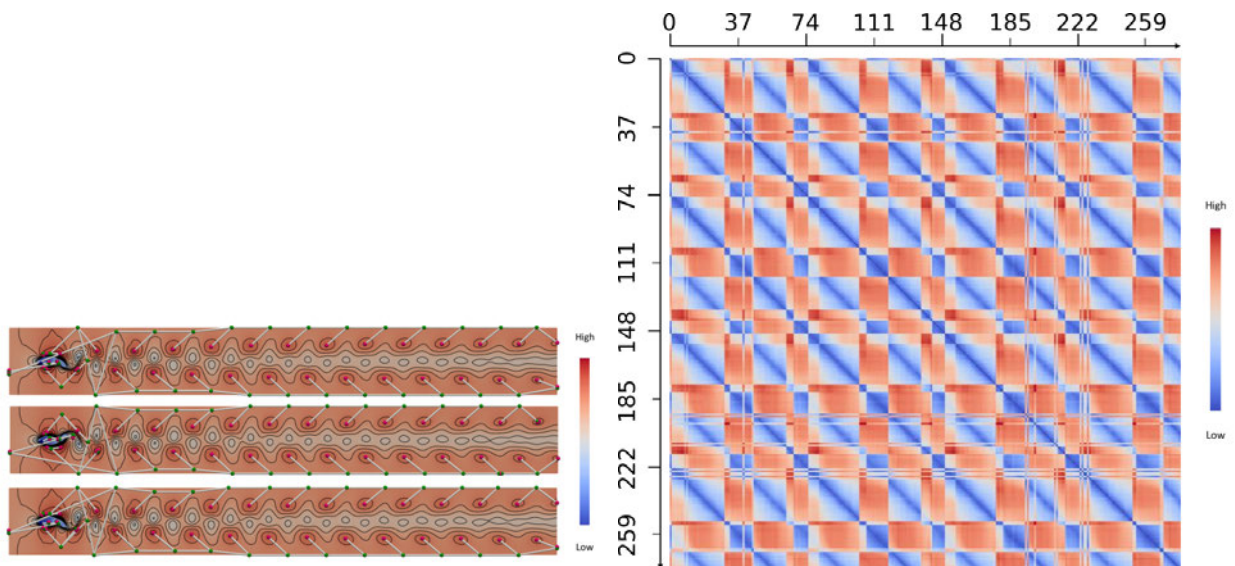


Figure 4.10: Comparing distance measures on the von Kármán vortex street dataset. (top) Plot of distance measures between the first time step and others when stability parameter ε is set to 0 and comparison with the Wasserstein distance W_1 and bottleneck distance D_B . (rows 2-4) Effect of stabilization parameter $\varepsilon = 0, 1.5, 5, 10, 20, 50, 100\%$ and comparison with Wasserstein distance W_1 . Results are shown in three plots to reduce clutter.



(a) Three timesteps from the flow around a cylinder simulation. (b) Distance matrix highlights the periodicity.

Figure 4.11: (a) Time step 0 (top), 37 (middle) and 74 (bottom) of the von Kármán vortex street dataset. The split tree and critical points are overlaid. (b) A truncated version of the DM showing the tree edit distance measure between all pairs of timesteps. Blue bands indicate periodicity with time period 74-75. A half period of 37, corresponding to the alternating nature of vortex shedding, is also visible.

4.4.2.3 Periodicity in time-varying data

Earlier studies of the Bénard-von Kármán vortex street dataset have successfully identified periodicity in the dataset. Narayanan et al. [88] detect both a half period of 38 and the full period of 75. We also aim to identify periodicity. Towards this, we compare the split tree of time step 1 with the remaining 1000 timesteps of the dataset. We plot the tree edit distance for timesteps 1 – 500, see top plot of Figure 4.10. We rerun the experiment and compare all 1000 timesteps with all other timesteps. The distances are stored in a distance matrix (DM). Each split tree contains approximately 55 – 65 nodes. The distances were computed in parallel using 12 threads and took approximately 25 minutes. A truncated version is shown in Figure 4.11(b) for clarity. From Figure 4.11(b), we can also observe a periodicity of 37, which matches with the results reported by Narayanan et al. [88]. The tree edit distance was computed in this experiment without stabilization. We also show the full version containing all the 1000 timesteps in Figure 4.12.

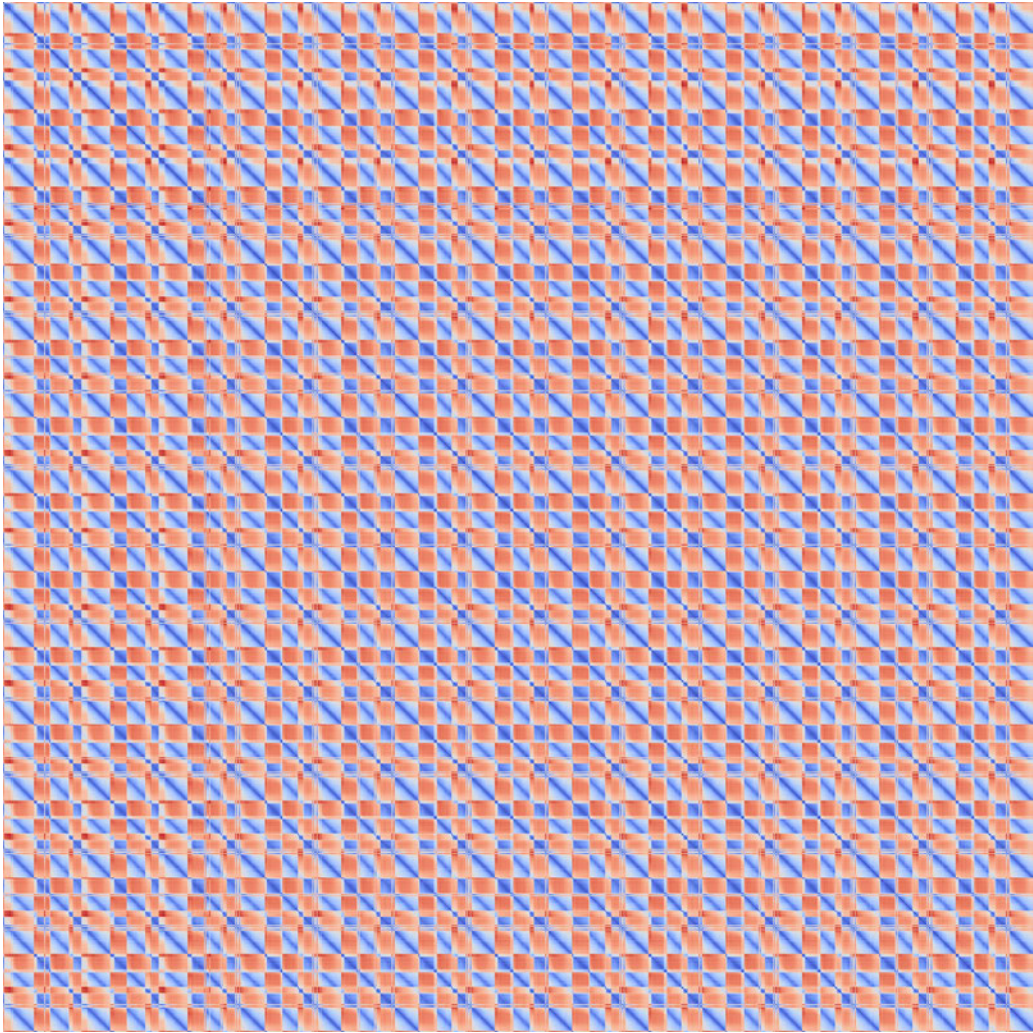


Figure 4.12: The full 1000×1000 distance matrix shows a half-period of 37 in addition to the periodicity of 74-75.

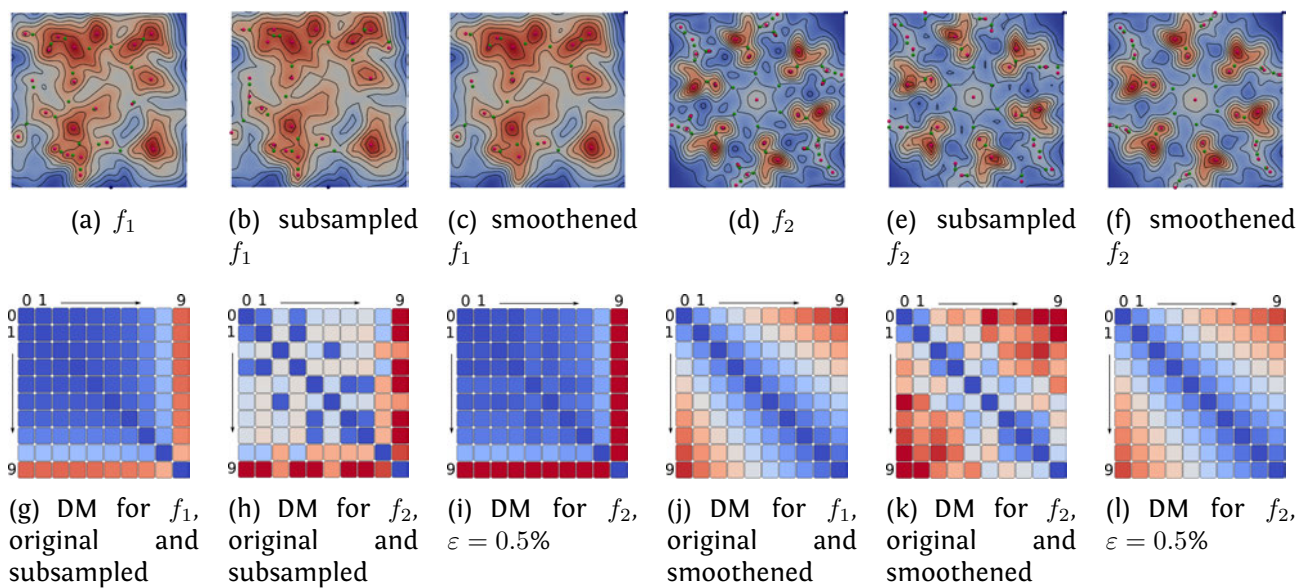


Figure 4.13: Measuring the effect of subsampling and smoothing. (a),(d) Two synthetic functions sampled over a 300×300 grid. (b),(e) Subsampled down to 30×30 over 9 iterations. (c),(f) Smoothed in 9 iterations. (g)-(i) DMs showing distance between all pairs of subsampled datasets without and with stabilization. (j)-(l) DMs showing distances for all pairs of smoothed datasets. Row and column indices correspond to the iteration number, 0 corresponds to the original, 9 corresponds to the lowest resolution/extreme smoothing. Red indicates high and blue indicates low values. Colormaps for f_1 and f_2 are not on the same scale.

4.4.2.4 Topological effects of subsampling and smoothing

The size of datasets are ever increasing and this mandates the use of subsampling and/or smoothing of the data as a preprocessing step. The aim of this preprocessing is to reduce the data size while ensuring a limited effect on geometric accuracy. However, the effect on the topological features of the scalar field is often not quantified. We want to observe how the tree edit distance measure captures these topological effects. We consider two synthetically generated datasets of size 300×300 (iteration 0), see Figures 4.13(a), 4.13(d). The data is downsampled over 9 iterations to a 30×30 grid by reducing the number of samples in each dimension by 30 within each iteration. We also apply 9 iterations of Laplacian smoothing on both 300×300 datasets. Next, we compare all merge trees corresponding to the subsampled and smoothed datasets pairwise.

The distance matrix (DM) for the function f_1 indicates that the distances are monotonic, which conforms to the expected behavior. But we see a different pattern in the case of function f_2 . A small stabilization applied on f_2 with $\varepsilon = 0.5\%$ results in distance matrices that conform to the expected behavior. This indicates that the stabilization may indeed be required, particularly when the scalar functions contain flat regions and multi-saddles. In both datasets, we notice that the distances between the lowest resolution (30×30) dataset and others is relatively high. We identified two reasons for the high values. First, the number of critical points reduces significantly between iterations 8 and 9. For example, in the case of f_2 , it goes down from 66 – 70 in earlier iterations to 58 in iteration 9. Second, the function value at the critical points in the lowest resolution dataset are also different. Hence, the relabel costs increase significantly, up to a factor of 1.5 in some cases.

4.4.2.5 Detecting symmetry / asymmetry

Identifying symmetric or repeating patterns in scalar fields enables feature-directed visualization. For example, it supports applications such as symmetry-aware transfer function design for volume rendering, anomaly detection, and query-driven exploration. A distance measure is central to any method for identifying symmetry. Consider the synthetic dataset in Figure 4.14 that contains six regions corresponding to six subtrees of the merge tree. Four regions colored green in Figure 4.14(b) are symmetric copies. The remaining two regions, colored orange and magenta, are slightly perturbed to cause asymmetry. We compute the tree edit distance measure to compare each subtree corresponding to a region with other subtrees. The measure clearly distinguishes between symmetric and asymmetric regions as can be seen from the distance matrix (DM) in Figure 4.14(c). These results are consistent with the premise upon which the data is generated.

We present additional case studies that demonstrate the applicability of the tree edit distance measure to symmetry identification. EMDB¹ contains 3D electron microscopy density data of macromolecules, subcellular structures, and viruses. Some of these structures contain symmetric subunits. We study two structures, EMDB 1654, and 1897, see Figure 4.15. First, we compute the split tree for each structure. We then use a semi-automated method to extract subtrees corresponding to significant features from the merge tree based on user specified persistence and minimum scalar value thresholds. Next, we compute the tree edit distance measure between subtrees corresponding to these regions of interest. We observe two distinct groups from the DMs. The tree edit distance measure clearly identifies two groups with 4 and 8 regions each in EMDB 1654, see Figure 4.15(a). Similarly, it identifies two groups containing 3 and 6 symmetric regions each in EMDB 1897, see Figure 4.15(b).

Comparison with previous methods. Thomas and Natarajan [124] process the branch decomposition of contour trees by building feature descriptors, and use them to identify similar subtrees. We use merge trees instead of contour trees and avoid computation of extremum graphs, geodesic distances, or contour shape descriptors in contrast to previous methods [125, 126]. There are some disadvantages in our approach, the main problem being the need to cut the merge tree based on persistence to generate auxiliary trees which are compared using MTED. This requires the user intervention as the number of cuts are crucial, a wrong number would result in missing some of the symmetric regions. Also the auxiliary trees are not merge trees. So this is useful only when the user is already aware of the number of symmetries present in the data. Also MTED computation is costly compared to the hierarchy descriptor based comparison [124].

4.4.2.6 Shape matching

Shape matching involves comparing geometric shapes and finding similarity between them. A good distance measure helps quantify this notion of similarity more concretely. The TOSCA non-rigid world dataset² contains a set of different shapes, see Figure 4.16. The shapes are in different poses and the project aims to develop methods to identify similarity between shapes in a pose invariant manner. We compute the average geodesic distance field [62] on the surface mesh. This field is well studied in the literature and is known to be a good shape descriptor. We apply a persistence simplification threshold of 1% on the merge trees both to remove topological noise and to reduce the number of nodes. Next,

¹<https://www.ebi.ac.uk/pdbe/emdb/>

²http://tosca.cs.technion.ac.il/book/resources_data.html

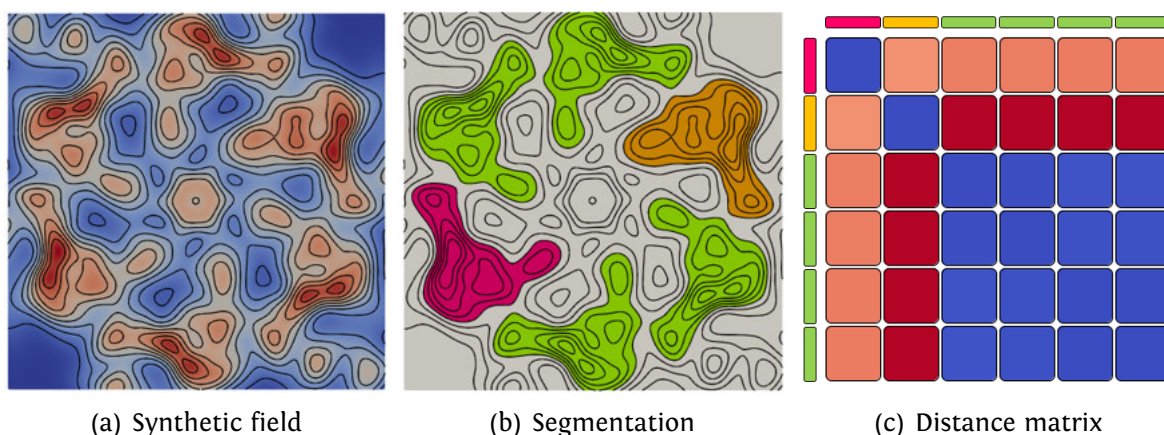
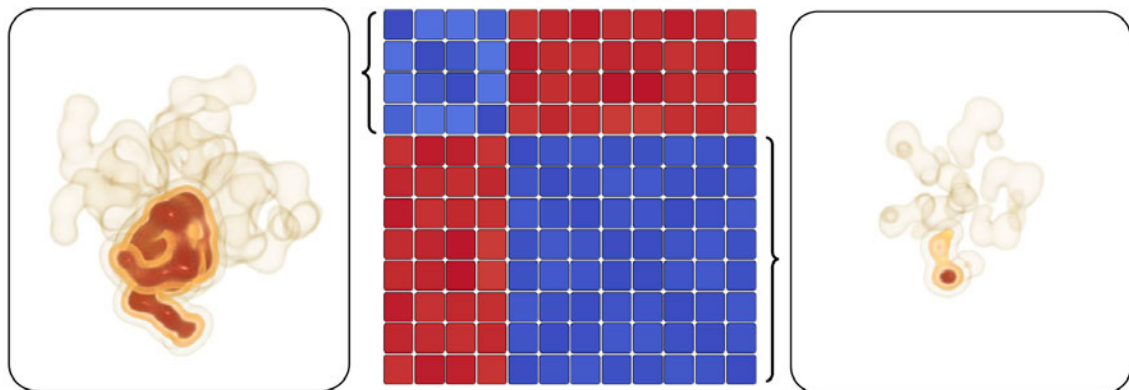


Figure 4.14: Identifying symmetry and asymmetry. (a) Sum of 2D gaussians. (b) The DM indicates presence of a symmetric group containing 4 regions. Two regions are correctly identified as being different from the rest. (c) DM between various subtrees of the merge tree.

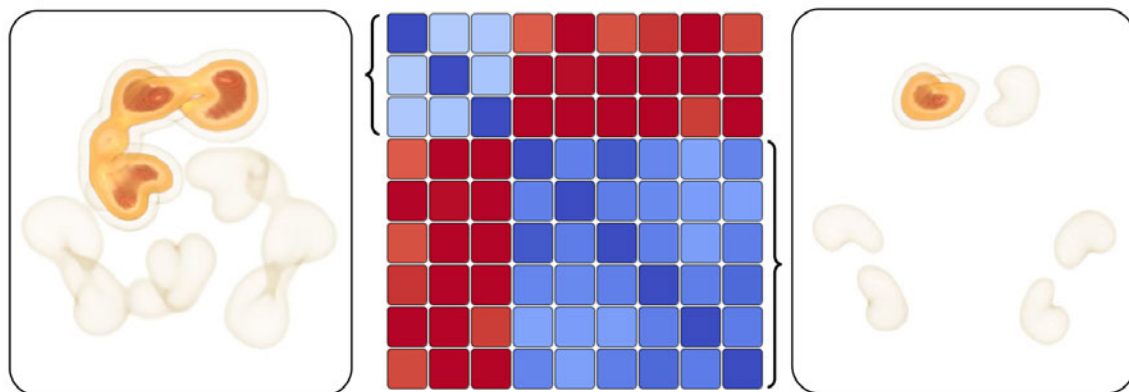
we compute the tree edit distance measure between all pairs of shapes. It takes around 15 seconds to generate the distance matrix with the same setup used for the periodicity experiment. Figure 4.17 shows the distance matrix. Each collection of shape appears as a blue block irrespective of variations in pose. We also observe higher values for a pair of shapes that are different. Note the blue blocks away from the diagonal. They correspond to Michael vs Victoria, David vs Victoria, David vs Michael, and David vs Gorilla. These pairs have similar shapes, which is more apparent in a few poses. Not all poses are shown in Figure 4.16.

Query based matching. We also showcase query based matching, by retrieving the top-4 best matches for a given query shape. Figure 4.18 shows the results for four such queries. We observe that in most cases we get shapes which are of the same class. We do get shapes from different class (last row) but in that case all the shapes are humanoid and the distance depends more on the pose than the class.

Comparison with the state-of-the-art. The shape matching problem has a long history and has been addressed by a variety of solutions, see the recent surveys by Biasotti et.al. [13], [14] and further updates by Zhou et.al. [147] and Sahillioglu [99] for more details. Use of topological descriptors are one of many such methods. While such descriptors come with some in-built advantages like being abstract skeletal representations of the data, being invariant to rigid body transformations, they discard geometry and are thus insufficient in many sce-



(a) EMD 1654



(b) EMD 1897

Figure 4.15: Detecting groups of symmetric regions in EMD datasets. (centre) DM showing tree edit distance between various pairs of subtrees of the merge tree. Low values are mapped to blue and high values to red. The DM indicates the presence of two distinct groups. All regions within a group are symmetric copies of each other. (left, right) Volume rendering where one region from each symmetric group is highlighted.

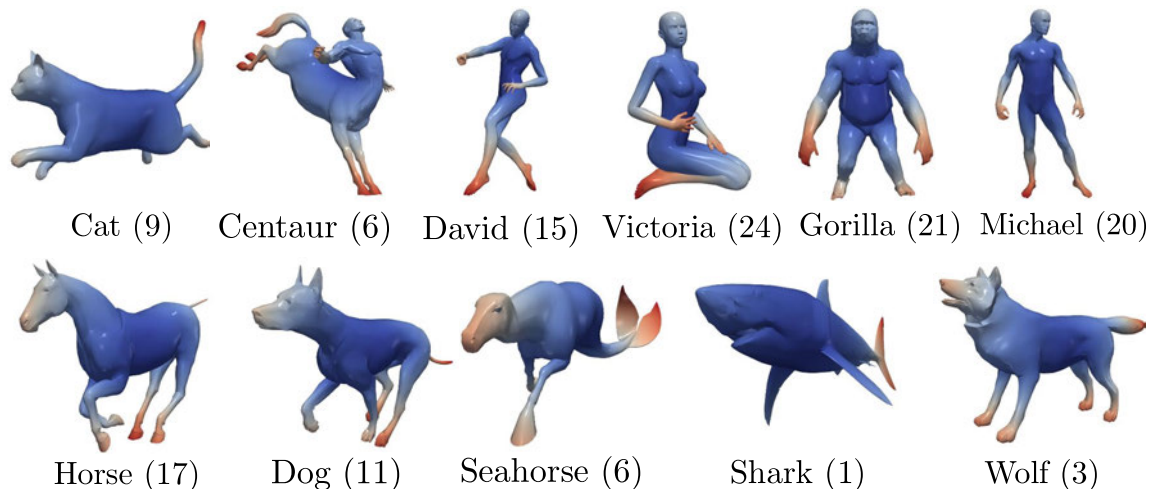


Figure 4.16: Collection of shapes from the TOSCA non-rigid world dataset. The average geodesic distance field [62] is computed on the surface. Each shape is available in multiple poses (number of poses mentioned within parenthesis), only one pose is shown here.

narios. Many of the recent descriptors which also consider geometry, along with learning based methods have resulted in algorithms which out perform topological descriptors. In this regard, we use MTED to showcase shape matching as a possible application. The state-of-the-art methods would be faster and more accurate in many cases compared to MTED based approach.

4.4.2.7 Data Summarization

Exploring large scientific data, particularly time-varying data, and identifying patterns of interest is often time consuming even with good visualization tools. Well designed abstract representations provide good overviews of the data and direct the user to features of interest. Abstractions such as the merge trees present a summary of spatial features. Temporal summaries enable effective visualization of time-varying data. Central to the design of a temporal summary is a good distance measure that can distinguish between periods of significant activity and inactive time periods.

In this experiment, we consider the 3D Bénard-von Kármán vortex street dataset. The velocity magnitude is available as a scalar field on a $192 \times 64 \times 48$ grid over 102 time steps [51]. Figure 4.19 shows volume renderings and isosurfaces for a few timesteps. Topological features of the velocity magnitude scalar field are represented using the split tree. Each split tree had approximately 180 – 200 nodes. We compute the tree edit distance between all pairs of timesteps. It takes around 4 seconds to generate the distance matrix (DM) with the same setup used for the periodicity experiment.

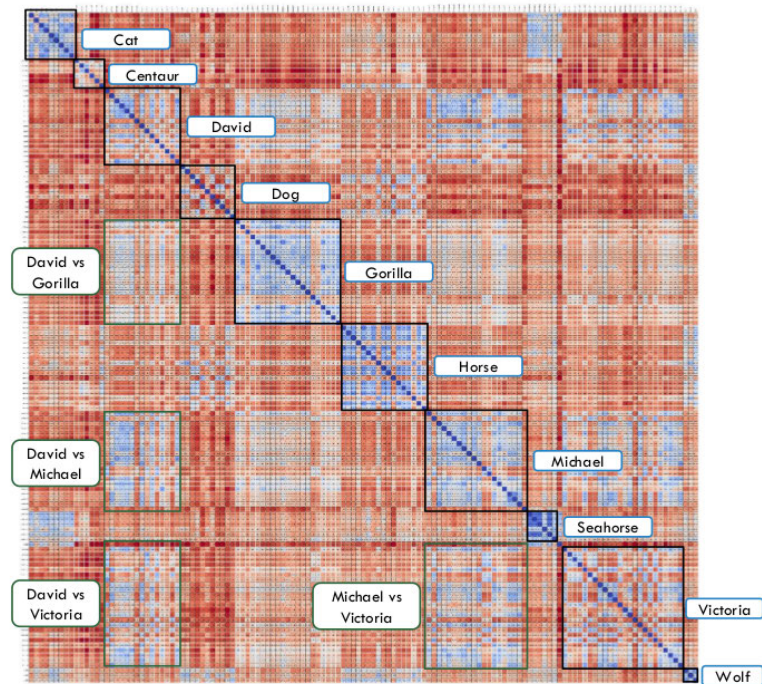


Figure 4.17: Tree edit distance matrix for all pairs of shapes from the TOSCA non-rigid world dataset. Blocks of low values (blue) correspond to similar shapes but in different poses.

The DM shown in Figure 4.20 contains multiple patterns. A fluid dynamics expert helped study and interpret the results. The distance between timesteps 2 – 28 are small because the flow does not contain any vortices and the features do not change. The top left blue block in the matrix corresponds to this time period. This is followed by the period when new vortex structures are formed (small block highlighted in green that corresponds to timesteps 29 – 39). Next, the vortices exhibit shedding, which is shown by the repeating patterns present in the larger green block in the matrix (timesteps 40 – 85). Finally, the vortices are significantly distorted, which is captured by the high values of distance in the bottom right block. Thus we can use the patterns that emerge in the distance matrix to distinguish between different types of behavior and summarize the scientific phenomena using these patterns.

4.5 Further developments

We have implemented a merge tree comparison framework in Java with python front end with the following functionalities.

- Computes bottleneck distance, Wasserstein distance, MTED with C_W and C_O .

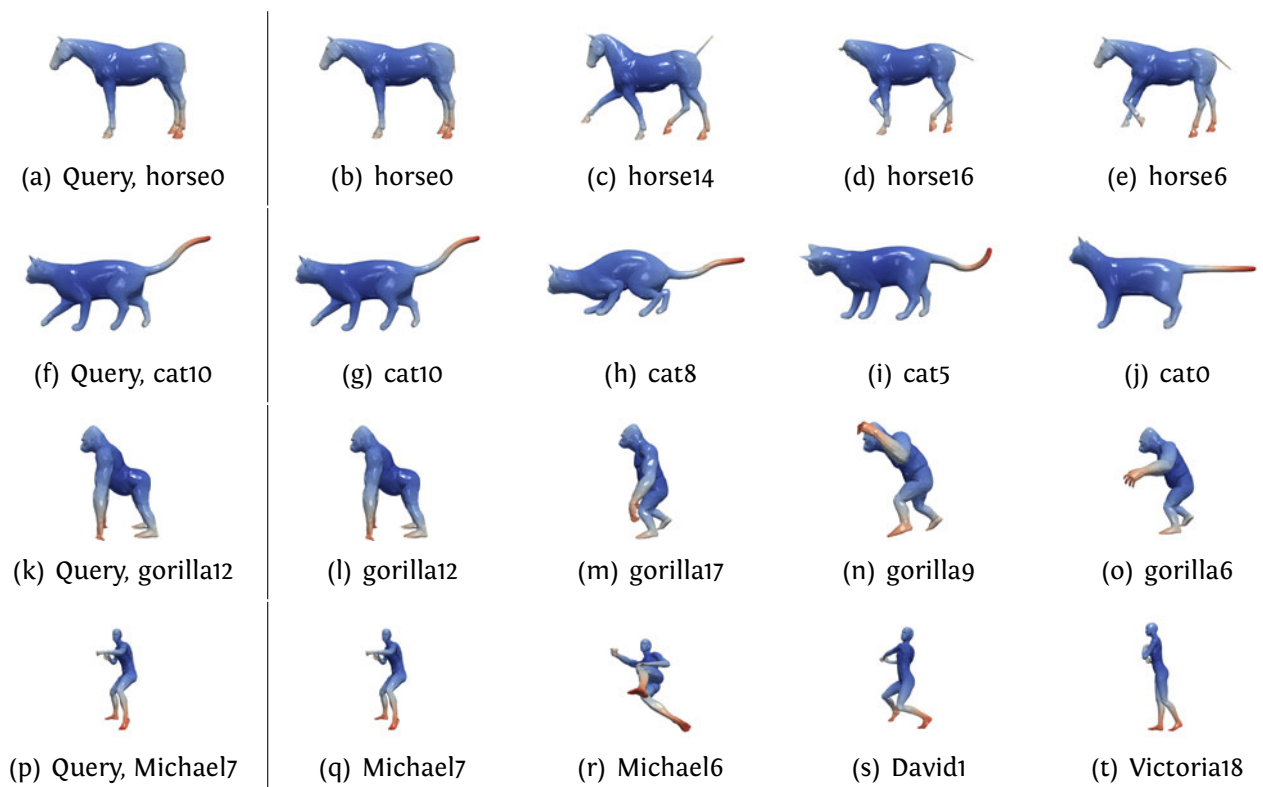


Figure 4.18: Top-4 matches for a given shape query. The number used as suffix is the index of the pose. In each row, the left-most figure is query, followed by the top-4 matches shown with their name and pose index. Note that in case of Michael along with other objects in the same class, we also get two objects from different classes (David,Victoria) since they are all humanoid figures.

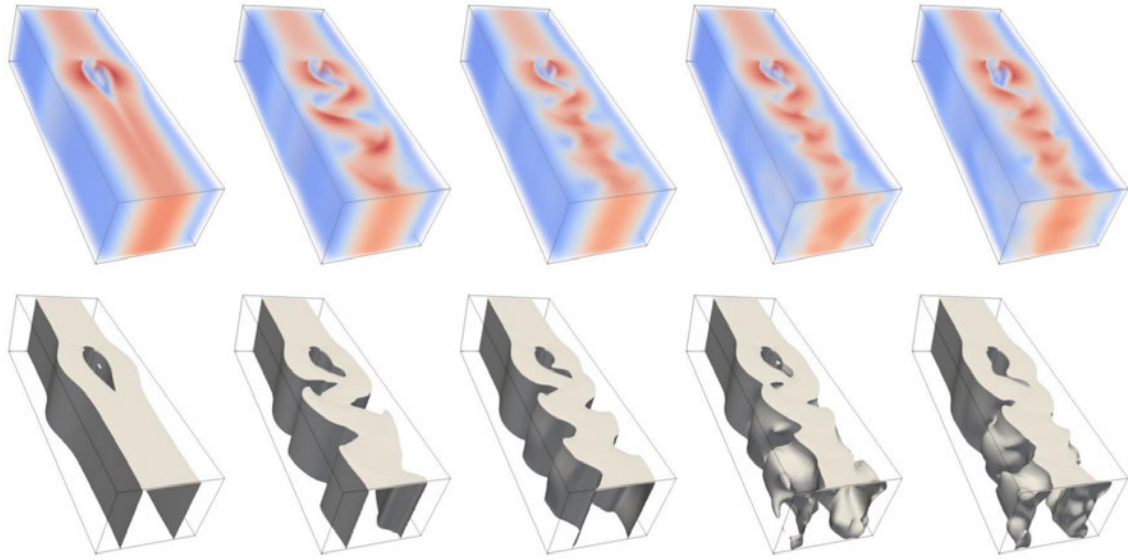


Figure 4.19: The 3D Bénard-von Kármán vortex street dataset. (top) Volume rendering of the velocity magnitude field for timesteps 15, 35, 58, 91, 98 ordered left to right. (bottom) Iso-surfaces at isovalue 0.7 extracted for the above timesteps.

- Allows a set of trees to be compared.
- Three comparison modes involving
 - Pair of trees
 - One tree vs the rest
 - All pairs of trees
- Provides simple visual output to show the mappings between the nodes of the compared trees.

We plan to release this framework in the near future with more options and implement it in C/C++ to offer better memory management and scalability.

4.6 Summary

We present two comparison distance measure OTED and MTED based on two models of tree edit distances. We define appropriate costs based on topological properties of the merge trees. The measures are parameter-free.

OTED works in case there is a generic gap model and an order can be imposed.

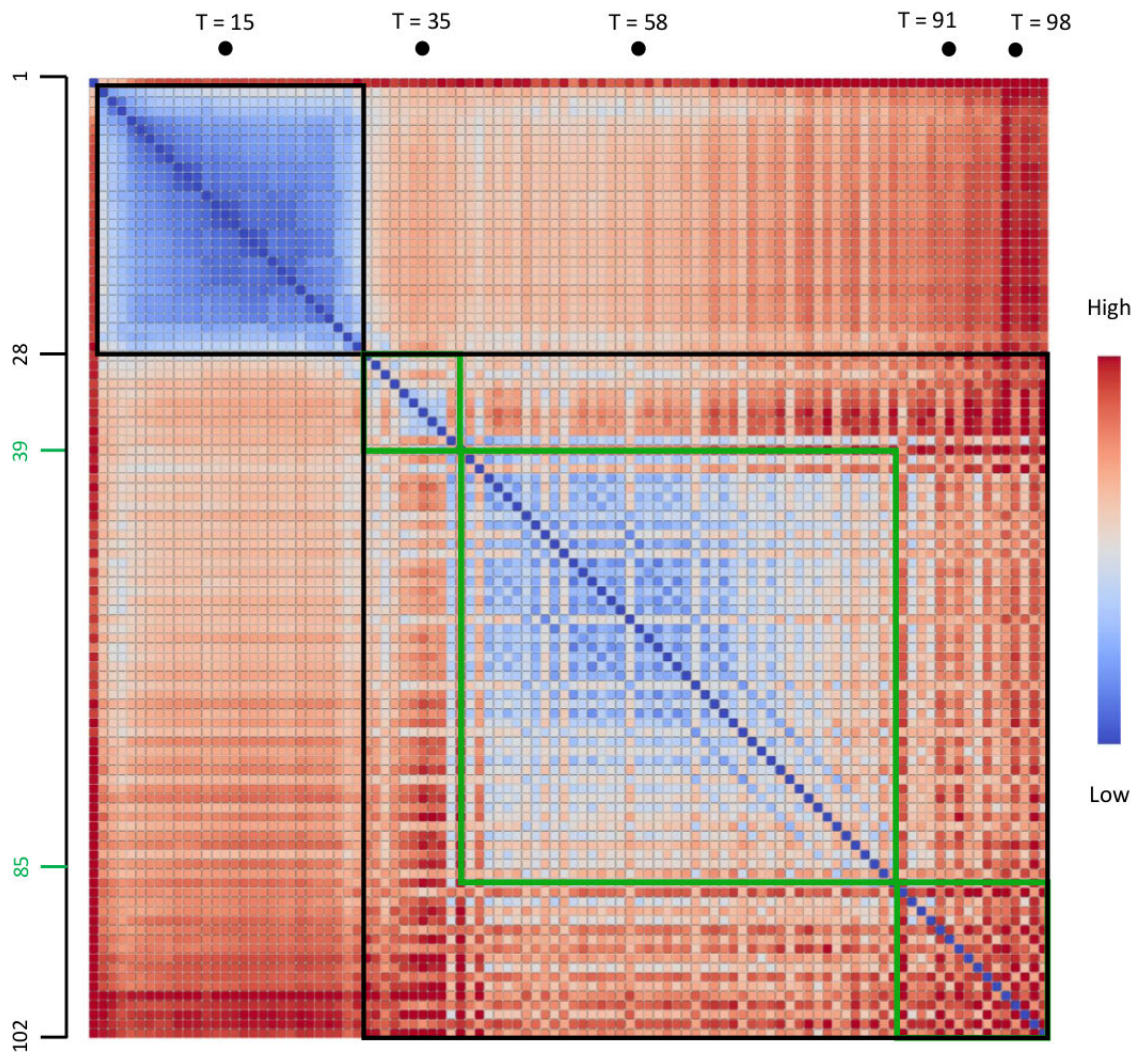


Figure 4.20: Tree edit distance matrix for all timesteps of the 3D Bénard-von Kármán vortex street dataset. Columns corresponding to timesteps 15, 35, 58, 91, 98 are highlighted. Patterns that help in generating a temporal summary are highlighted using black and green boxes.

MTED on the other hand is more versatile. The distance measure is defined as the minimum cost of a set of restricted edit operations that transforms one tree into another. The edit operations and the associated costs are both intuitive and mathematically sound. The measure satisfies metric properties, can be efficiently computed, and is useful in practice. We study the properties of the measure and demonstrate its application to data analysis and visualization using various computational experiments.

Chapter 5

Local edit distance for merge trees (LMTED)

In this chapter we present a comparison measure LMTED [117] to facilitate local comparison in merge trees. We also discuss various properties and applications.

5.1 Introduction

Most of the comparison measures described in Chapter 2 compare these structures globally. While global comparisons help us address a variety of interesting problems it cannot be used for finer analysis, specifically of the local structure or substructures of the corresponding topological features. For example, consider the split trees rooted at i and j in Figures 5.1(a) and 5.1(b). Global comparison measures convey the overall dissimilarity, but do not capture the similarity between the regions that map to the pairs of subtrees rooted at (i_{10}, j_7) , (i_8, j_6) and (i_7, j_5) respectively. This type of fine grained or multi-scale analysis leads to interesting applications and hence there is a need for a measure that detects similarity both locally and across multiple scales. See Chapter 2.2.5 for more details. In this chapter, we show how such merge tree edit distance MTED can be extended towards a fine-grained comparison of scalar fields. Specifically, the global MTED is restricted to cases where only the tree at the top of the hierarchy is guaranteed to be a merge tree. In contrast, LMTED facilitates comparison between all pairs of subtrees of merge trees by ensuring that all comparisons are between trees that are guaranteed to be merge trees.

5.2 Contributions

We propose a local tree edit distance based method to compare substructures of scalar fields across multiple scales. The comparison measure is an adaptation of the global tree edit distance for merge trees (MTED) introduced by Sridharamurthy et al. [120]. However, it

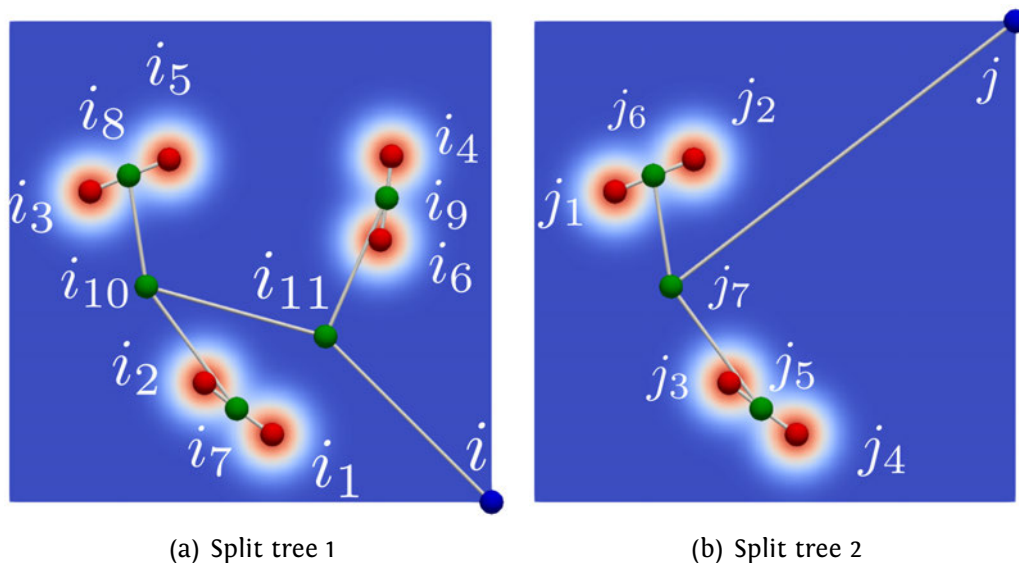


Figure 5.1: Scalar fields f_1 and f_2 that are not globally similar but contain locally similar regions. Split trees are overlaid on top of the scalar fields

is substantially different in terms of the definition, properties, approach to its computation, and applications.

1. A novel local tree edit distance (LMTED) to compare substructures in scalar fields.
2. A proof that it satisfies metric properties.
3. A dynamic programming algorithm to compute the LMTED efficiently.
4. A notion of truncated persistence to compute costs of matching / correspondences, which brings in the additional benefit of saving computation time by reducing the number of comparisons.
5. Experiments to demonstrate the practical value of the distance towards symmetry detection at multiple scales, analysis of the effects of smoothing and subsampling, a fine grained analysis of topological compression, and applications to feature tracking.

The MTED supports only a few of the above-mentioned applications. Even in these cases, it is restricted to comparisons on a coarser level or requires a higher level of user intervention. Feature tracking is not possible with MTED without significant modifications.

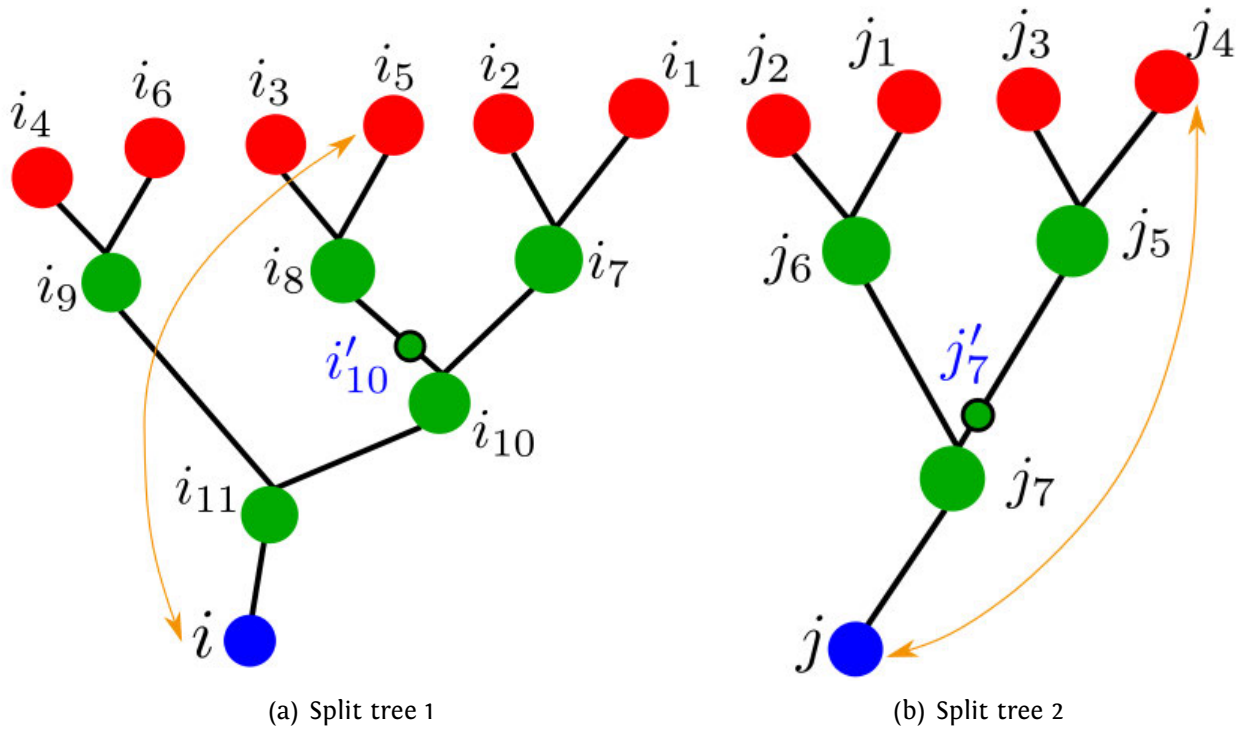


Figure 5.2: Split trees corresponding to the scalar fields shown in Figure 5.1. One persistence pair is shown within each split tree (orange link). When comparing subtrees $T_1[i_8]$ or $T_2[j_5]$, nodes i_5 and j_4 are unpaired. So, dummy nodes i'_{10} and j'_7 are inserted with $|f_1(i_{10}) - f_1(i'_{10})| < \varepsilon$ and $|f_2(j_7) - f_2(j'_7)| < \varepsilon$ to serve as root and as a pair of the unpaired node.

5.3 Local merge tree edit distance LMTED

In this section we describe the necessary modification required to go from MTED to LMTED, define LMTED, discuss the new cost model and its properties, describe the DP algorithm to calculate LMTED, describe the implementation details such as refinements and finally showcase applications.

5.3.1 Truncated persistence and truncated costs

The cost of edit operations in MTED [120] is based on the topological persistence of the node(s). Specifically, their L^∞ cost C_W is the L^∞ distance between the point pair in the persistence diagram corresponding to the two nodes (relabel) or the distance between the point in the persistence diagram and the diagonal (insert / delete). While comparing subtrees such as $T_2[j_5]$ (Figure 5.2(b)), a node j_4 whose persistence pair is the global root j contributes a large value to the edit distance. Using the same cost for j_4 while comparing all subtrees containing the node may not be appropriate. For example, subtrees $T_2[j_5]$ and

$T_2[j]$ map to two regions in the domain that are vastly different in size. To alleviate this inconsistency, we introduce the notion of dummy nodes and truncated persistence. While comparing the subtree $T_2[j_5]$ with other subtrees, we insert a dummy node j'_7 that serves as a root of the subtree $T_2[j_5]$ and as the pair of j_4 . The function value at the dummy node differs from the parent of j_5 at most by a small value ε .

An unpaired node in a subtree corresponds to a leaf whose persistence pair is outside the subtree. We use i_u and j_u to denote unpaired nodes in $T_1[i]$ and $T_2[j]$, respectively. Dummy nodes corresponding to $T_1[i]$ and $T_2[j]$ are denoted by i' and j' . For an unpaired node $i_u \in T_1[i]$ and a truncated root that is represented by a dummy node i' , we define *truncated persistence* as

$$tp_{i'}(i_u) = |f_1(i_u) - f_1(i')|. \quad (5.1)$$

Note that a leaf node will be unpaired in some subtree and, in some cases, it can be unpaired in multiple subtrees (for example, j_4). The subscript i' in the definition specifies the subtree under consideration.

The cost of the edit operations need to be updated based on the truncated persistence for unpaired nodes i_u and j_u . Let γ denote the original cost of an edit operation derived from the L^∞ cost C_W used in MTED. We define a new truncated cost γ' as

$$\gamma'(i \rightarrow j) = \begin{cases} \gamma(i \rightarrow \lambda), i \neq i_u, j = j_u | \lambda, \\ \gamma(\lambda \rightarrow j), i = i_u | \lambda, j \neq j_u, \\ \gamma(i \rightarrow j), i \neq i_u, j \neq j_u, \\ 0, i = i_u | \lambda, j = j_u | \lambda, \end{cases} \quad (5.2)$$

5.3.2 Definition

We now describe a new local tree edit distance that is appropriate for localized comparison of merge trees, discuss its properties, and present an algorithm for computing the distance. The local tree edit distance (LMTED) for a pair of trees rooted at i and j is denoted $\text{LMTED}(i, j)$ and defined as follows:

$$\text{LMTED}(i, j) = D'(i, j) + \Gamma(i_u \rightarrow j_u). \quad (5.3)$$

Here, $\Gamma(i_u \rightarrow j_u)$ denotes the relabel cost computed using the truncated persistence values of i_u and j_u . $D'(i, j)$ is the modified edit distance between the two trees that excludes the

cost between the unpaired nodes from each tree.

We now describe the recursive formulation of $D'(i, j)$. The key difference between LMTED and MTED is the way in which unpaired nodes are handled. The original formulation (as for D_c) applies as is for the paired nodes, not for the unpaired nodes. The unpaired node changes depending on the level of the subtree, as we move from the leaves towards the root of the merge tree. As we traverse a merge tree bottom-up considering all possible subtrees, a node can be unpaired until we reach the level containing its pair, following which it remains paired until the end. When the node is unpaired, at every level its contribution changes. We account for the contribution from the unpaired node to $D'(i, j)$ in a final step and so we are able to retain a recursive formulation. Once the node is paired, its contribution (equal to its persistence) does not change further. This demands a new recursive formulation which can handle both the scenarios. The new formulation D' will thus be a modification of D_c .

From Figure 5.3 and using similar terminology as before, let i_1, i_2, \dots, i_{n_i} be the children of i and j_1, j_2, \dots, j_{n_j} be the children of j . Let $T_1[i]$ denote the subtree rooted at i and $F_1[i]$ denote the unordered forest obtained by deleting the node i from $T_1[i]$. Again, i_u and j_u are unpaired nodes in $T_1[i]$ and $T_2[j]$, respectively. Let i_{u_i} be the child lying on the path between i and i_u in $T_1[i]$ and j_{u_j} be the child lying on the path between j and j_u in $T_2[j]$.

Recall that θ denotes the empty tree and $D_c(\cdot)$ denotes MTED. Then D' is recursively defined as follows:

$$D'(\theta, \theta) = 0, \tag{5.4}$$

$$D'(F_1[i], \theta) = \sum_{k=1, k \neq u_i}^{n_i} D_c(T_1[i_k], \theta) + D'(T_1[i_{u_i}], \theta), \tag{5.5}$$

$$D'(T_1[i], \theta) = D'(F_1[i], \theta) + \gamma'(i \rightarrow \lambda), \tag{5.6}$$

$$D'(\theta, F_2[j]) = \sum_{k=1, k \neq u_j}^{n_j} D_c(\theta, T_2[j_k]) + D'(\theta, T_2[j_{u_j}]), \tag{5.7}$$

$$D'(\theta, T_2[j]) = D'(\theta, F_2[j]) + \gamma'(\lambda \rightarrow j), \tag{5.8}$$

$$\min_{T_2} = \min \left\{ \begin{array}{l} \min_{1 \leq t \leq n_j, t \neq u_j} \{D_c(T_1[i], T_2[j_t]) - D_c(\theta, T_2[j_t])\}, \\ \{D'(T_1[i], T_2[j_{u_j}]) - D'(\theta, T_2[j_{u_j}])\} \end{array} \right.$$

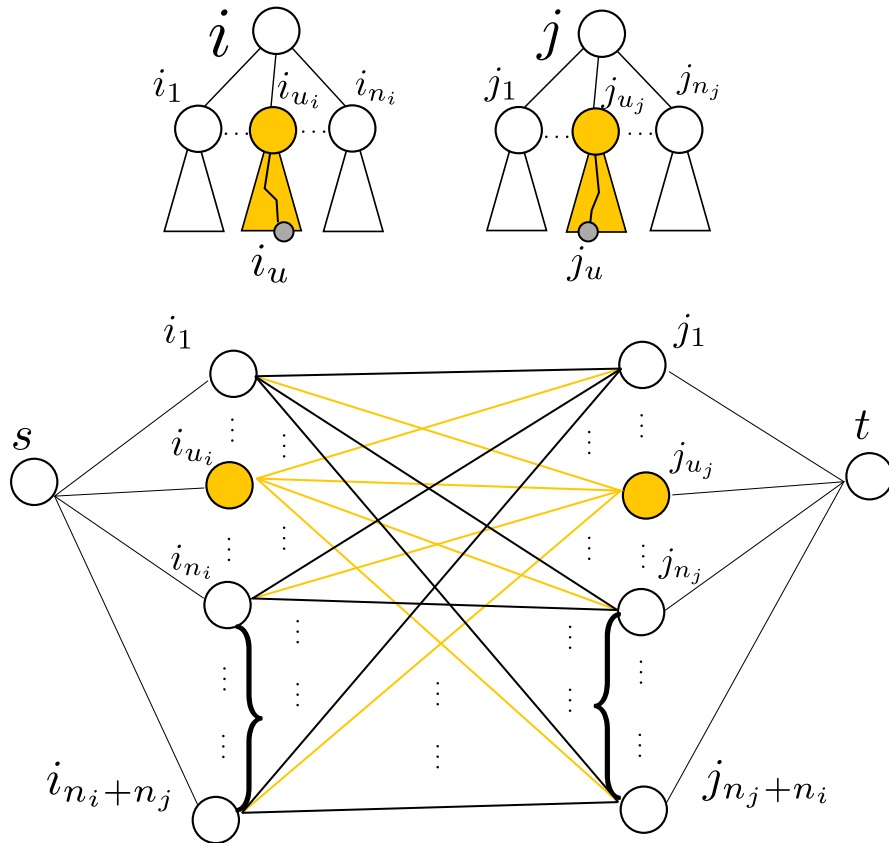


Figure 5.3: Illustrating LMTED. To compute LMTED between subtrees rooted at i and j , we treat the subtrees containing the unpaired nodes i_u, j_u , labeled as i_{u_i}, j_{u_j} and highlighted in orange, differently. For other nodes, the formulation is same as MTED. For i_u, j_u , we use truncated persistence to determine the costs. In the matching required to compute $M'_r(i, j)$, we consider truncated persistence to determine the weights of all edges incident on i_{u_i}, j_{u_j} (highlighted in orange). s and t are the source and destination nodes of the flow problem that is equivalent to the matching problem to determine $M'_r(i, j)$. The cardinalities of the two sides are made equal by inserting a set of n_j dummy nodes adjacent to s and n_i dummy nodes adjacent to t .

$$\min_{T_1} = \min \left\{ \begin{array}{l} \min_{1 \leq s \leq n_i, t \neq u_i} \{D_c(T_1[i_s], T_2[j]) - D_c(T_1[i_s], \theta)\}, \\ \{D'(T_1[i_{u_i}], T_2[j]) - D'(T_1[i_{u_i}], \theta)\} \end{array} \right.$$

$$\min_{F_j} = \min \left\{ \begin{array}{l} \min_{1 \leq t \leq n_j, t \neq u_j} \{D_c(F_1[i], F_2[j_t]) - D_c(\theta, F_2[j_t])\}, \\ \{D'(F_1[i], F_2[j_{u_j}]) - D'(\theta, F_2[j_{u_j}])\} \end{array} \right.$$

$$\min_{F_i} = \min \left\{ \begin{array}{l} \min_{1 \leq s \leq n_i, t \neq u_i} \{D_c(F_1[i_s], F_2[j]) - D_c(F_1[i_s], \theta)\}, \\ \{D'(F_1[i_{u_i}], F_2[j]) - D'(F_1[i_{u_i}], \theta)\} \end{array} \right.$$

$$D'(T_1[i], T_2[j]) = \min \left\{ \begin{array}{l} D'(\theta, T_2[j]) + \min_{T_2}, \\ D'(T_1[i], \theta) + \min_{T_1}, \\ D'(F_1[i], F_2[j]) + \gamma'(i \rightarrow j). \end{array} \right. \quad (5.9)$$

$$D'(F_1[i], F_2[j]) = \min \left\{ \begin{array}{l} D'(\theta, F_2[j]) + \min_{F_j}, \\ D'(F_1[i], \theta) + \min_{F_i}, \\ \min_{M'_r(i,j)} \gamma'(M'_r(i,j)). \end{array} \right. \quad (5.10)$$

All terms that involve subtrees containing the unpaired node (orange) are updated to incorporate D' , whereas the constrained edit distance D_c appears elsewhere. The bipartite graph formulation that is used to compute the minimum cost restricted mapping between forests is also updated to incorporate D' and is now denoted as $M'_r(i, j)$ (Figure 5.3, bottom).

5.3.3 Cost models and properties

We can employ either of the costs, the L_∞ cost C_W or the overhang cost C_O introduced for the MTED [120, Section 4.2]. Both costs are proven to be metrics. If they remain so even with the newly introduced cost based on the truncated persistence, then by Zhang [144] the LMTED satisfies metric properties. The proofs of non-negativity and symmetry is straightforward

because γ' together with Γ is defined in terms of γ , which in turn satisfies both properties because it is a combination of sum, max, and min of absolute values.

We now prove the triangle inequality for γ' together with Γ . Let $T_1[i]$, $T_2[j]$, and $T_3[k]$ be three subtrees with unpaired nodes i_u, j_u, k_u . We insert dummy nodes i', j', k' and construct trees $T_1[i']$, $T_2[j']$, and $T_3[k']$. The truncated persistence values $tp_{i'}(i_u), tp_{j'}(j_u), tp_{k'}(k_u)$ in subtrees $T_1[i], T_2[j], T_3[k]$ are respectively equal to the regular persistence values within trees $T_1[i'], T_2[j'], T_3[k']$. For a given triple $i_1 \in T_1[i], j_1 \in T_2[j], k_1 \in T_3[k]$, we will show that triangle inequality holds by considering different cases.

Case 1: Nodes i_1, j_1, k_1 are all unpaired or are all paired. When all the nodes i_1, j_1, k_1 are paired, $\gamma' = \gamma$ and hence triangle inequality holds. If all are unpaired, $\gamma'(i_1 \rightarrow j_1) = \gamma'(j_1 \rightarrow k_1) = \gamma'(i_1 \rightarrow k_1) = 0$. Further, $\Gamma(i_1 \rightarrow j_1), \Gamma(j_1 \rightarrow k_1)$ and $\Gamma(i_1 \rightarrow k_1)$ are equal to relabel costs (γ) for the trees $T_1[i'], T_2[j']$ and $T_3[k']$ and hence triangle inequality holds [120, Section 4.3].

Case 2: Two nodes are unpaired. The case where i_1 and k_1 are unpaired while j_1 is paired is impossible because of the constraint that unpaired nodes are mapped to unpaired nodes and the operation is forced to be a relabel.

Case 2.1: i_1 and j_1 are unpaired. Then the LHS

$$\gamma'(i_1 \rightarrow j_1) + \gamma'(j_1 \rightarrow k_1) + \Gamma(i_1 \rightarrow j_1) \quad (5.11)$$

$$= 0 + \gamma(\lambda \rightarrow k_1) + \Gamma(i_1 \rightarrow j_1) \quad (5.12)$$

$$= \gamma(\lambda \rightarrow k_1) + \Gamma(i_1 \rightarrow j_1), \quad (5.13)$$

and the RHS

$$\gamma'(i_1 \rightarrow k_1) = \gamma(\lambda \rightarrow k_1). \quad (5.14)$$

Since $\Gamma(i_1 \rightarrow j_1) \geq 0$ always, we have LHS \geq RHS.

Case 2.2: j_1 and k_1 are unpaired, then the LHS

$$\gamma'(i_1 \rightarrow j_1) + \gamma'(j_1 \rightarrow k_1) + \Gamma(j_1 \rightarrow k_1) \quad (5.15)$$

$$= \gamma(i_i \rightarrow \lambda) + 0 + \Gamma(j_1 \rightarrow k_1) \quad (5.16)$$

$$= \gamma(i_i \rightarrow \lambda) + \Gamma(j_1 \rightarrow k_1), \quad (5.17)$$

and the RHS

$$\gamma'(i_1 \rightarrow k_1) = \gamma(i_1 \rightarrow \lambda). \quad (5.18)$$

Since $\Gamma(j_1 \rightarrow k_1) \geq 0$ always, we again have LHS \geq RHS.

Case 3: A single node is unpaired. This is impossible because of the constraint that unpaired nodes are mapped only to unpaired nodes via a relabel edit operation.

In all cases, the truncated cost γ' together with Γ satisfies the triangle inequality. So, it follows from Zhang [144] that LMTED is indeed a metric.

5.3.4 Algorithm and computation

We propose a modified dynamic programming based algorithm for computing LMTED between merge trees. The use of truncated persistence as cost of the edit operations implies that LMTED can be computed using solutions to non-overlapping sub-problems for computing D_c . However, the dynamic programming formulation needs to be modified because D' recursively depends both on D' and D_c .

5.3.4.1 Dynamic Programming tables

Consider the subtrees $T[i_{10}]$ and $T[i_8]$ in Figure 5.2. The node i_5 is unpaired. Within the subtree $T[i_8]$, node i_5 has truncated persistence $tp_{i_8'}(i_5)$. But, its truncated persistence is equal $tp_{i_{10}'}(i_5)$ when considering the subtree $T[i_{10}]$. Dynamic programming works by storing the results of sub-problems within a table so that it can be reused. Entries corresponding to both (in general, more than two) values of truncated persistence are required for the computation.

Consider a subtree $T[i]$ with an unpaired node i_u and the path from i_u to the global root r as shown in Figure 5.4. Let i_v be the pair of i_u , clearly $i_v \in \text{ancestor}(i)$. While processing the unpaired node i_u , we need to distinguish between two cases:

- O:** The contribution of i_u is measured by persistence as defined in the usual sense.
- M:** The contribution of i_u is measured by an appropriate instance of truncated persistence.

Case **O** corresponds to all subtrees rooted at $i_a \in \text{ancestor}(i)$, where $i_v \leq i_a \leq r$ in the directed path highlighted in green in Figure 5.4. Case **M** corresponds to trees rooted at i_a such that $i_u \leq i_a < i_v$, which also includes i as highlighted in red in Figure 5.4.

We wish to design an algorithm that makes a single pass over the two input merge trees and computes LMTED between all pairs of subtrees. In order to achieve this objective, we

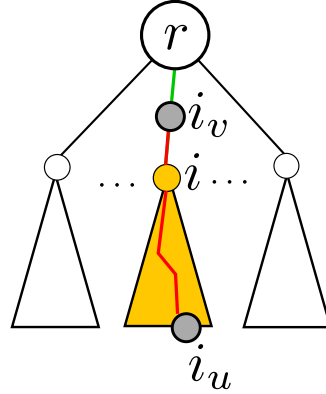


Figure 5.4: Illustration of Cases O and M. The portion of the path colored in red corresponds to case M while the portion of the path colored in green corresponds to case O.

propose a modified dynamic programming method that uses two tables. One table stores the values of D_c for sub-problems, as defined and proposed for MTEB [144]. We introduce a second table that stores D' , partial solutions to the modified edit distance for sub-problems. Figure 5.5 and Figure 5.6 show the dynamic programming table corresponding to D_c and D' , respectively. Entries in the second table D' are defined as follows:

1. To compute the entry (i, j) , we need first populate the entries for the subtrees of $T_1[i]$ and $T_2[j]$. Say, we need to compute the entry at (i_k, j_l) , where $i_k \in T_1[i]$, $j_l \in T_2[j]$. If the subtrees rooted at i_k and j_l do not contain the unpaired nodes i_u and j_u , i.e., $i_u \notin T_1[i_k]$ and $j_u \notin T_2[j_l]$, then we pick the corresponding entry from D_c , namely $D_c(T_1[i_k], T_2[j_l])$. Figure 5.7 and Figure 5.8 shows the entries required to compute the entry (i, j) in both tables.
2. If the subtrees rooted at i_k and j_l contain the unpaired nodes, we refer to the modified table D' , whose entries are computed using truncated persistence.

Figure 5.5 shows the original table used to compute D_c . It does contain entries corresponding to different pairs of subtrees, but only the global entry (i, j) corresponds to a distance between two merge trees. Other local comparisons involve subtrees that are not merge trees. Figure 5.6 is the modified table storing values of D' . Figures 5.7 and 5.8 show the difference between the original and modified tables, and the entries that are required to calculate $D_c(i_{10}, j_7)$ and $D'(i_{10}, j_7)$. The labels correspond to the merge trees from Figure 5.2. We denote tree distance entries by $d_c(\cdot)$ and $d'(\cdot)$, and forest entries by $f_c(\cdot)$ and $f'(\cdot)$. The node i_5 is unpaired in $T_1[i_{10}]$ and j_4 is unpaired in $T_2[j_7]$. In case of D_c , there is no distinction between paired and unpaired nodes. So, $D_c(i_{10}, j_7)$ depends only on the following

	-1	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i
-1													
j_1													
j_2													
j_3													
j_4													
j_5									✗				
j_6													
j_7													
j													✓

Figure 5.5: Dynamic programming table for D_c . The only useful entry in this table is the one corresponding to (i, j) . Other entries do not represent a meaningful distance because the subtrees compared to compute the entries are not merge trees. Entries in the row and column labeled -1 correspond to comparison with the empty tree θ .

	-1	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i
-1													
j_1													
j_2													
j_3													
j_4													
j_5										✓			
j_6													
j_7													
j													✓

← $|g(j_4) - g(j_7)|$

↑ $|f(i_5) - f(i_{10})|$

Figure 5.6: The modified DP table corresponding to D' , which is used to compute LMTED. This table contains an additional row and column when compared to the table for D_c . This additional row and column stores the truncated persistence values for the unpaired nodes corresponding to the current level. Entries in this table represent valid (and useful) distance between merge trees, which are subtrees of the trees rooted at i and j , respectively.

entries of the original table - $d_c(i_{10}, -1)$, $d_c(-1, j_7)$, $d_c(i_7, j_7)$, $d_c(i_8, j_7)$, $d_c(i_{10}, j_5)$, $d_c(i_{10}, j_6)$, and $f_c(i_{10}, j_7)$. In case of D' , for all subtrees involving unpaired nodes ($T_1[i_8], T_2[j_5]$), the corresponding entries are read from the modified table, while the remaining entries are read from the table for D_c . So, the required entries include $d'(i_{10}, -1)$, $d'(-1, j_7)$, $d_c(i_7, j_7)$, $d'(i_8, j_7)$, $d'(i_{10}, j_5)$, $d_c(i_{10}, j_6)$, and $f'(i_{10}, j_7)$. Further, the entries in the additional row and column are also required. They contain the truncated persistence values for the unpaired nodes corresponding to the current level.

5.3.4.2 Pseudocode and analysis

Zhang described an algorithm for computing the tree edit distance for labeled unordered trees [144]. It is a dynamic programming based algorithm that follows from the properties

	-1	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i
-1													
j_1													
j_2													
j_3													
j_4													
j_5													
j_6													
j_7													
j													

Figure 5.7: Entries required to calculate $D_c(i, j)$ (\star) and $D'(i, j)$ (\star) in the table for D_c . The required entry is denoted by a green square. The lower triangle entries correspond to forests and upper triangle entries to trees.

	-1	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i
-1													
j_1													
j_2													
j_3													
j_4													
j_5													
j_6													
j_7													
j													

Figure 5.8: Entries required to calculate $D'(i, j)$ (\star) in the modified table. The required entry is denoted by a green square. The lower triangle entries correspond to forests and upper triangle entries to trees.

discussed in Section 3.7.1.2. We adapt this dynamic programming based method for computing LMTED but compute and maintain two tables D_c and D' simultaneously. The algorithm fills entries in both tables D_c and D' iteratively. An entry within a table is computed and filled if entries corresponding to all sub-problems are already filled in previously. This implicitly corresponds to traversing the two input trees in a bottom up fashion in tandem. After both tables are filled, LMTED is computed for all pairs of subtrees following the definition *i.e.*, as a sum of D' and Γ . Algorithm 2 computes the LMTED. Here γ denotes the original cost model and γ' denotes the truncated cost model. Line 2 initializes the distances between two empty trees to 0. The loops spanning lines 3 – 8 and 9 – 14 fill the table entries for both D_c and D' corresponding to the distances between the empty tree and all trees and forests. Note that lines 6, 7 and 12, 13 are new additions compared to the MTED algorithm, which depend on values from both D_c and D' . The nested loops spanning lines 15 – 26 fill the entries that correspond to distances between non-empty forests and trees.

Again, lines 19 – 24 are additions to the MTED algorithm. To avoid clutter, the expressions $\min_{F_j}, \min_{F_i}, \min_{T_2}, \min_{T_1}$ are written separately, though they are part of the expressions calculating D' in lines 23, 24. Though the expressions look complicated, if we substitute D' with D_c , γ' with γ , and M'_r with M_r in the RHS of the expressions in lines 23, 24 we get the original MTED expressions which are in lines 17, 18. The entry $D_c(T_1[m], T_2[n])$ in the table with $m = |T_1|$ and $n = |T_2|$ corresponds to the final result for MTED. In case of LMTED, if we are interested in the distance between the pair of subtrees rooted at i and j , then the distance is given by

$$\text{LMTED}(i, j) = D'(i, j) + \Gamma(i_u \longrightarrow j_u). \quad (5.19)$$

$\Gamma(i_u \longrightarrow j_u)$ denotes the relabel cost computed using the truncated persistence values of i_u and j_u .

Algorithm 2: LocalTreeEditDistance (LMTED) [144]

Data: Merge trees T_1, T_2 .

Result: $D'(T_1[i], T_2[j])$ and $D_c(T_1[i], T_2[j])$, where $1 \leq i \leq |T_1|, 1 \leq j \leq |T_2|$

```

1 begin
2    $D_c(\theta, \theta) = 0, D'(\theta, \theta) = 0$ 
3   for  $i = 1$  to  $|T_1|$  do
4      $D_c(F_1[i], \theta) = \sum_{k=1}^{n_i} D_c(T_1[i_k], \theta)$ 
5      $D_c(T_1[i], \theta) = D_c(F_1[i], \theta) + \gamma(i \longrightarrow \lambda)$ 
6      $D'(F_1[i], \theta) = \sum_{k=1, k \neq u_i}^{n_i} D_c(T_1[i_k], \theta) + D'(T_1[i_{u_i}], \theta)$ 
7      $D'(T_1[i], \theta) = D'(F_1[i], \theta) + \gamma'(i \longrightarrow \lambda)$ 
8   end
9   for  $j = 1$  to  $|T_2|$  do
10     $D_c(\theta, F_2[j]) = \sum_{k=1}^{n_j} D_c(\theta, T_2[j_k])$ 
11     $D_c(\theta, T_2[j]) = D_c(\theta, F_2[j]) + \gamma(\lambda \longrightarrow j)$ 
12     $D'(\theta, F_2[j]) = \sum_{k=1, k \neq u_j}^{n_j} D_c(\theta, T_2[j_k]) + D'(\theta, T_2[j_{u_j}])$ 
13     $D'(\theta, T_2[j]) = D'(\theta, F_2[j]) + \gamma'(\lambda \longrightarrow j)$ 
14  end
15 end

```

In the worst case, the algorithm computes LMTED between all pairs of subtrees in $O(|T_1| \times |T_2| \times (\deg(T_1) + \deg(T_2)) \times \log_2(\deg(T_1) + \deg(T_2)))$ time, similar to MTED. In practice,

16

17

for $i = 1$ **to** $|T_1|$ **do**

18

for $j = 1$ **to** $|T_2|$ **do**

19

$$D_c(F_1[i], F_2[j]) = \min \begin{cases} D_c(\theta, F_2[j]) + \min_{1 \leq t \leq n_j} \{D_c(F_1[i], F_2[jt]) - D_c(\theta, F_2[jt])\}, \\ D_c(F_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D_c(F_1[i_s], F_2[j]) - D_c(F_1[i_s], \theta)\}, \\ \min_{MM(i,j)} \gamma(MM(i,j)). \end{cases}$$

20

$$D_c(T_1[i], T_2[j]) = \min \begin{cases} D_c(\theta, T_2[j]) + \min_{1 \leq t \leq n_j} \{D_c(T_1[i], T_2[jt]) - D_c(\theta, T_2[jt])\}, \\ D_c(T_1[i], \theta) + \min_{1 \leq s \leq n_i} \{D_c(T_1[i_s], T_2[j]) - D_c(T_1[i_s], \theta)\}, \\ D_c(F_1[i], F_2[j]) + \gamma(i \rightarrow j). \end{cases}$$

21

$$\min_{F_j} = \min \begin{cases} \min_{1 \leq t \leq n_j, t \neq u_j} \{D_c(F_1[i], F_2[jt]) - D_c(\theta, F_2[jt])\}, \\ \{D'(F_1[i], F_2[j_{u_j}]) - D'(\theta, F_2[j_{u_j}])\} \end{cases}$$

22

$$\min_{F_i} = \min \begin{cases} \min_{1 \leq s \leq n_i, s \neq u_i} \{D_c(F_1[i_s], F_2[j]) - D_c(F_1[i_s], \theta)\}, \\ \{D'(F_1[i_{u_i}], F_2[j]) - D'(F_1[i_{u_i}], \theta)\} \end{cases}$$

23

$$\min_{T_2} = \min \begin{cases} \min_{1 \leq t \leq n_j, t \neq u_j} \{D_c(T_1[i], T_2[jt]) - D_c(\theta, T_2[jt])\}, \\ \{D'(T_1[i], T_2[j_{u_j}]) - D'(\theta, T_2[j_{u_j}])\} \end{cases}$$

24

$$\min_{T_1} = \min \begin{cases} \min_{1 \leq s \leq n_i, s \neq u_i} \{D_c(T_1[i_s], T_2[j]) - D_c(T_1[i_s], \theta)\}, \\ \{D'(T_1[i_{u_i}], T_2[j]) - D'(T_1[i_{u_i}], \theta)\} \end{cases}$$

25

$$D'(F_1[i], F_2[j]) = \min \begin{cases} D'(\theta, F_2[j]) + \min_{F_j}, \\ D'(F_1[i], \theta) + \min_{F_i}, \\ \min_{M'_r(i,j)} \gamma'(M'_r(i,j)) \end{cases}$$

26

$$D'(T_1[i], T_2[j]) = \min \begin{cases} D'(\theta, T_2[j]) + \min_{T_2}, \\ D'(T_1[i], \theta) + \min_{T_1}, \\ D'(F_1[i], F_2[j]) + \gamma'(i \rightarrow j). \end{cases}$$

the computation is restricted to a much smaller set of entries that need to be filled within both tables. This restricted set of entries is identified in a preprocessing step as described in the following section. Also, note that the time is amortized over all pairs of subtrees.

5.3.5 Refinement and optimization

Given two merge trees, we observe that in many applications it is unnecessary to compute `LMTED` between all pairs of subtrees. This section describes refinements that reduces the number of pairs of subtrees that are considered for `LMTED` computation. This optimization leads to faster computation times. This step may be skipped if it is necessary to compare all pairs of subtrees. We also describe a refinement that ensures that all subtrees considered for comparison are merge trees.

5.3.5.1 Ordering subtrees

The dynamic programming algorithm works for any ordering of nodes. The entries required to compute the distance between a pair of subtrees are computed if they are not already available from the table. However, we choose to order the nodes by assigning a priority based on the size of the subtree rooted at the node and on the number of grid points in the domain mapped to the subtree (*i.e.*, volume of the corresponding region in the domain). This ordering facilitates easy identification of similar regions via visual inspection of the distance matrix because subtrees of similar size appear in the close vicinity of one another within the matrix.

5.3.5.2 Comparison refinement

Since `LMTED` is computed between all pairs of subtrees of the two input trees, the number of comparisons is determined by the size of the two trees. In general, the scalar fields to be compared are unrelated, defined on different domains, and have different ranges. So, comparing all pairs of subtrees is necessary and unavoidable. However, in several applications, it is not necessary or meaningful to compare all pairs of subtrees. We describe two such scenarios to motivate a refinement step that reduces the number of comparisons.

1. **Symmetry detection:** The scalar field is compared with itself. So, we can discard comparisons between subtrees with vastly different sizes (for example, $T_1[i_4]$ and $T_1[i_{10}]$ in Figure 5.2) or a subtree that is contained within another subtree (like $T_1[i_{11}]$ and $T_1[i_8]$).
2. **Time-varying data analysis:** While analyzing a time-varying scalar field, assuming a fine enough temporal resolution, we may discard comparisons between subtrees with

vastly different sizes. We may also discard comparisons between subtrees that map to regions in the domain with significantly different sizes (area / volume).

We define a set of criteria used to direct the refinement. Each criterion is a ratio between measures or a statistic computed for a subtree. Let T_1, T_2 be the two input merge trees with nodes $\{i_1, i_2, \dots, i_{n_i}\}$ and $\{j_1, j_2, \dots, j_{n_j}\}$, and roots r_1 and r_2 , respectively. Consider a subtree $T_1[i_k]$ and the mapping to its associated region in the domain. This region is a connected component of the preimage of the range of scalar values corresponding to $T_1[i_k]$. Assuming that the scalar function is defined over a 3D domain, the volume of this region may be approximated by counting the number of sample points (vertices or grid points, depending on whether the domain is represented using a tetrahedral mesh or a cube grid). The aggregate persistence P_{i_k} of the subtree is computed as the sum of persistence of all persistence pairs contained within the subtree. Given a pair of subtrees $T_1[i_k]$ and $T_2[j_l]$, we use the ratio between

1. number of nodes in the subtrees $|T_1[i_k]| / |T_2[j_l]|$,
2. volume (or area) of the domain that maps to the two subtrees, and
3. aggregate persistence of the subtrees P_{i_k} / P_{j_l}

to determine the refinement. Thresholds for the ratios are determined empirically. For each criterion, we plot the number of pairs of subtrees against increasing values of the ratio, identify the value of the ratio corresponding to a sharp decline or the 'knee' of the curve, and choose this value of the ratio as threshold. If the plot does not exhibit a clear knee then we set the threshold to 0.5. Further, if $T_1 = T_2$ then we discard all comparisons between $T_1[i_k]$ and all subtrees contained within $T_1[i_k]$.

5.3.5.3 Subtree refinement

Next, we preprocess the input to ensure that we compare only those subtrees that are merge trees.

The subtrees that constitute the sub-problems in the dynamic programming algorithm for MTED [120] are not necessarily merge trees. Consider the scenario in Figure 5.2. While trees $T_1[i]$ and $T_2[j]$ rooted at i and j are merge trees, $T_1[i_8]$ and $T_2[j_5]$ are not merge trees as they contain unpaired nodes, namely i_5 and j_4 . The entry $D_c(i_8, j_5)$ in the DP table (Figure 5.5) contains a value that is used to compute the MTED between $T_1[i]$ and $T_2[j]$. But, it is not a meaningful distance between subtrees $T_1[i_8]$ and $T_2[j_5]$ because the cost model used for operations related to unpaired nodes i_5, j_4 depends on their original persistence. From

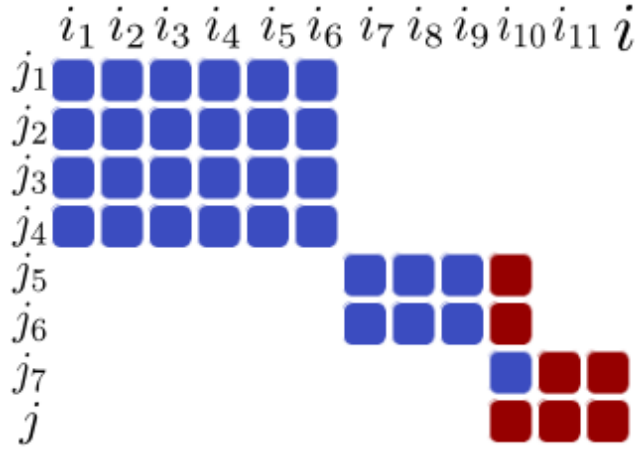



Figure 5.9: Understanding LMTED. Columns of the distance matrix (DM) represent subtrees rooted at nodes of merge tree T_1 , rows represent subtrees rooted at nodes of tree T_2 . Nodes are ordered as per the priority described in Section 5.3.5.1. LMTED values are shown using a blue-red colormap (0  0.1)

Figure 5.1, it is clear that the regions associated to the two subtrees are similar to each other but the value of $D_c(i_8, j_5)$ does not reflect this similarity.

We insert a dummy node i'_{10} in $T_1[i_8]$ with $|f_1(i'_{10}) - f_1(i_{10})| < \varepsilon$ for a small $\varepsilon > 0$. In subsequent computations, we consider $T_1[i'_{10}]$ as the merge tree corresponding to the subtree $T_1[i_8]$. Similarly, $T_2[j_5]$ contains the unpaired node j_4 . We insert a dummy node j'_7 and consider $T_2[j'_7]$ as the merge tree corresponding to $T_2[j_5]$. This conversion is consistent with the mapping between subtrees and regions of the domain and hence results in meaningful distances.

5.4 Applications

In this section, we demonstrate the utility of LMTED in applications like symmetry detection, feature tracking, and spatio-temporal exploration of scientific data. We also describe results of a comprehensive analysis of the effects of subsampling, smoothing, and topologically controlled compression. In all cases where a global comparison is meaningful, results based on MTED [120] are taken as a baseline.

5.4.1 Understanding the local tree edit distance

We begin with a simple study to understand LMTED, by comparing two scalar fields shown in Figures 5.1, whose split trees are shown in Figure 5.2. Figure 5.9 shows the distance matrix (DM), entries corresponding to subtree pairs that are discarded during the refinement step

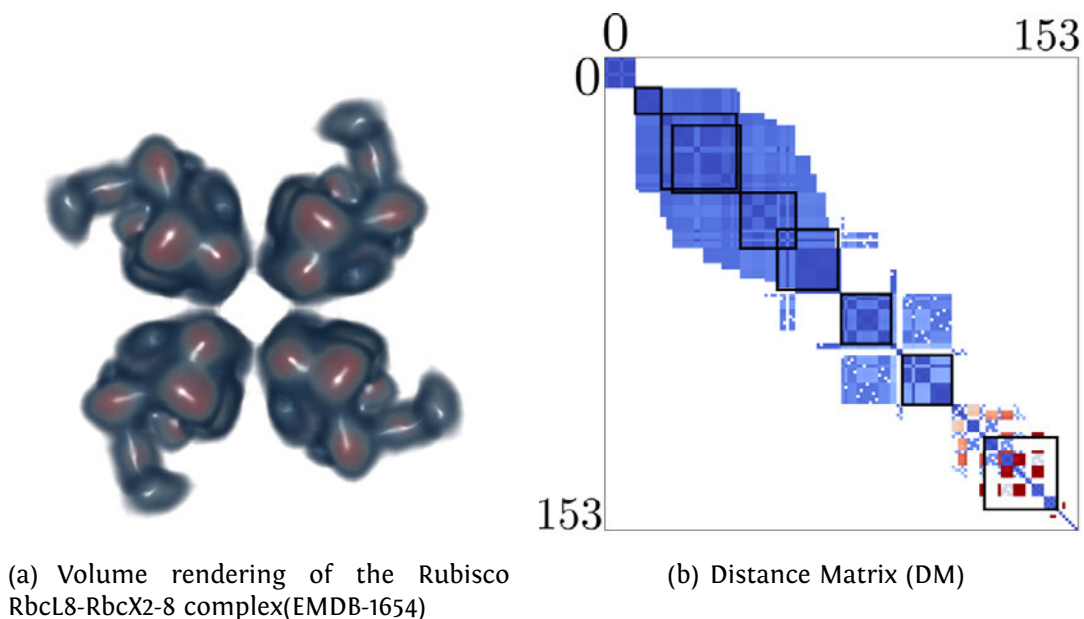


Figure 5.10: LMTED values in the DM are shown using a blue-red colormap (0  0.1)

are blank. We observe two blue blocks of size 4×6 and 2×3 in the DM, confirming that there are two sets of similar regions and the pair of similar subtrees $T_1[i_{10}], T_2[j_7]$. Note that the distances between these similar regions are very small ≤ 0.000093 in contrast to the larger value of MTED ($= 0.33$) between the two trees. Such instances of local similarity without global similarity is common in scientific data. Further, LMTED also captures similarity at different scales.

5.4.2 Symmetry Detection

Finding symmetric structures in scalar fields is a challenging problem [124, 125, 126]). The MTED driven approach [120] extracts a particular set of high persistent subtrees that are known to be symmetric and compares them to verify symmetry. We take a different approach where we detect symmetry directly based on local similarity by comparing the scalar field with itself. We use CryoEM data from EMDB¹, which contains 3D electron microscopy density data of macromolecules, subcellular structures, and viruses. We first compute the simplified merge tree (using a small persistence threshold $< 1\%$) and consider pairs of subtrees after refinements described in Section 5.3.5.

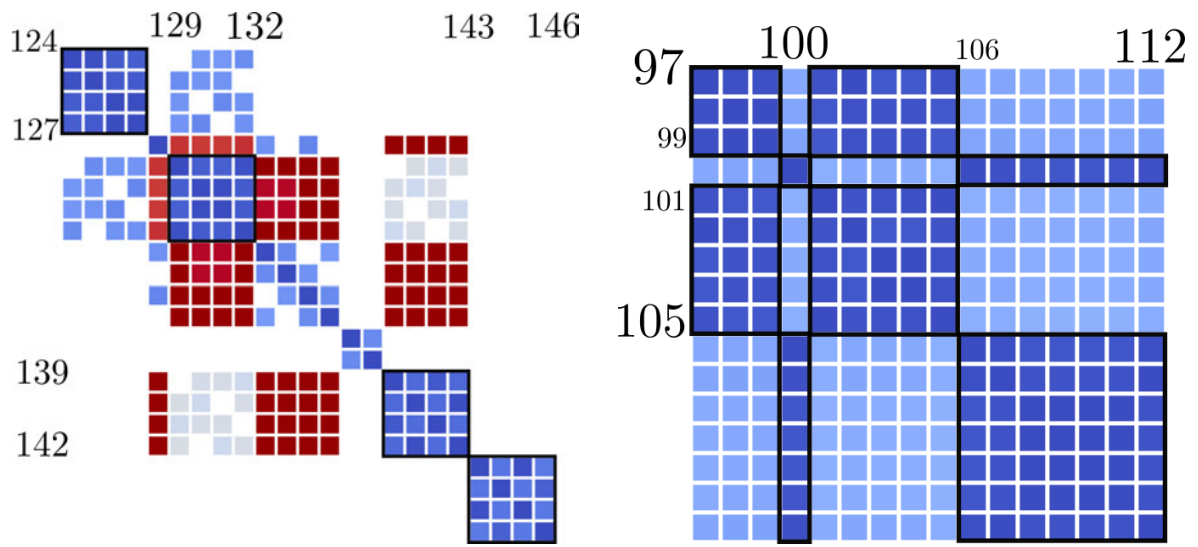
We illustrate and analyse the results using the Rubisco RbcL8-RbcX2-8 complex (EMDB-1654) shown in Figure 5.10(a). We compute LMTED between all pairs of subtrees of its merge

¹<https://www.ebi.ac.uk/pdbe/emdb/>

tree after the refinement step. The resulting DM is shown in Figure 5.10(b). Blank regions correspond to subtree pairs that are discarded during the refinement step. Submatrices highlighted in black correspond to regions in the data that are symmetric. For clarity, we have shown these submatrices together with the corresponding regions in Figures 5.11, 5.12. We observe that LMTED detects symmetric regions at different scales.

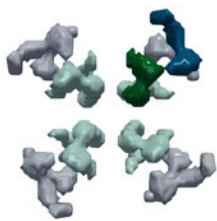
Any selection of submatrices from Figure 5.10(b) with a common color corresponds to a set of symmetric regions, we highlight some of them. In some cases matrix entries corresponding to symmetric regions may not appear adjacent to each other as a submatrix. But, it may be possible to visually identify the entries as belonging to a single cluster. A row/column reordering helps the identification of these clusters, see Behrisch et al. [9] for details. The reordering may be restricted to a chosen submatrix to save computation time. We have chosen two examples – EMDB 1603 (12 Angstrom resolution cryo-electron microscopy reconstruction of a recombinant active ribonucleoprotein particle of influenza virus) to show how the symmetric regions are found without any matrix reordering and EMDB 1897 (AMP-Activated Protein Kinase) to illustrate the case where reordering might be required. The volume rendering of EMDB 1603 is shown in Figure 5.13(a) The modified DP is calculated for the merge tree of EMDB 1603, which has pairs of subtrees marked based on refinement criteria to get the distance matrix DM as shown in Figure 5.13(b). The empty regions in the DM corresponds to pairs of subtrees which are not being compared as they are eliminated by the refinement steps discussed in Section 5.3.5. Consider the submatrices highlighted, these correspond to set of regions in the data which are symmetric. For clarity, we have shown the submatrices and the corresponding set of regions in Figures 5.14(a), 5.14(b), 5.14(c), 5.14(d). We can observe that we are able to detect multiple set of symmetric regions in different scales. Note that the set of regions corresponding to 4×4 submatrix given by 122, 125 is detected even though it belongs to the noisy regions outside the molecule because of its large size. Since the method prioritizes larger regions, the submatrix occurs at the bottom right.

A volume rendering of EMDB 1897 is shown in Figure 5.15(a). The distance matrix DM is shown in Figure 5.15(b). We observe that the symmetric regions do not appear as submatrices. It is difficult to visually inspect the matrix and detect the submatrices (unlike EMDB 1654). Matrix reordering techniques (leaf-reordering [9]) are applied on the DM to obtain the matrix shown in Figure 5.15(c). After reordering, we observe that the symmetric regions appear together. The highlighted submatrices correspond to a set of symmetric regions in the data. For clarity, we have shown the submatrices and the corresponding set of regions in Figure 5.16. We observe multiple sets of symmetric regions at different length scales.

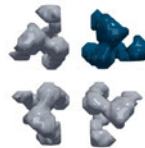


(a) DM 124

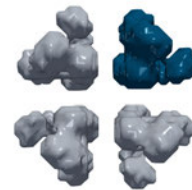
(b) DM 97



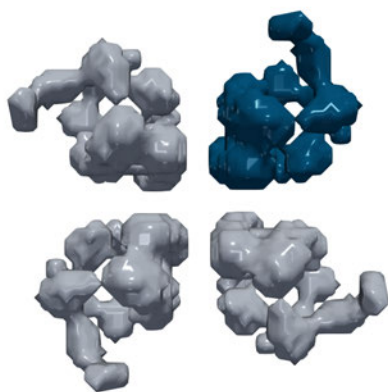
(c) volume 97



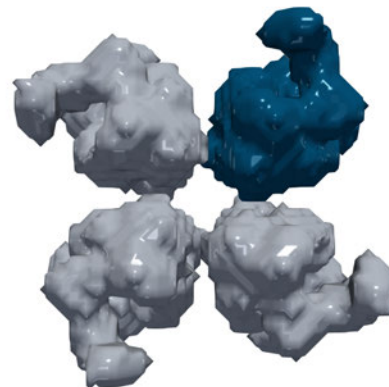
(d) volume 124



(e) volume 129



(f) volume 139



(g) volume 143

Figure 5.11: Highlighted submatrices from Figure 5.10 and corresponding regions. (d)-(g) Regions corresponding to submatrices highlighted in (a). (c) Region corresponding to submatrix shown in (b)

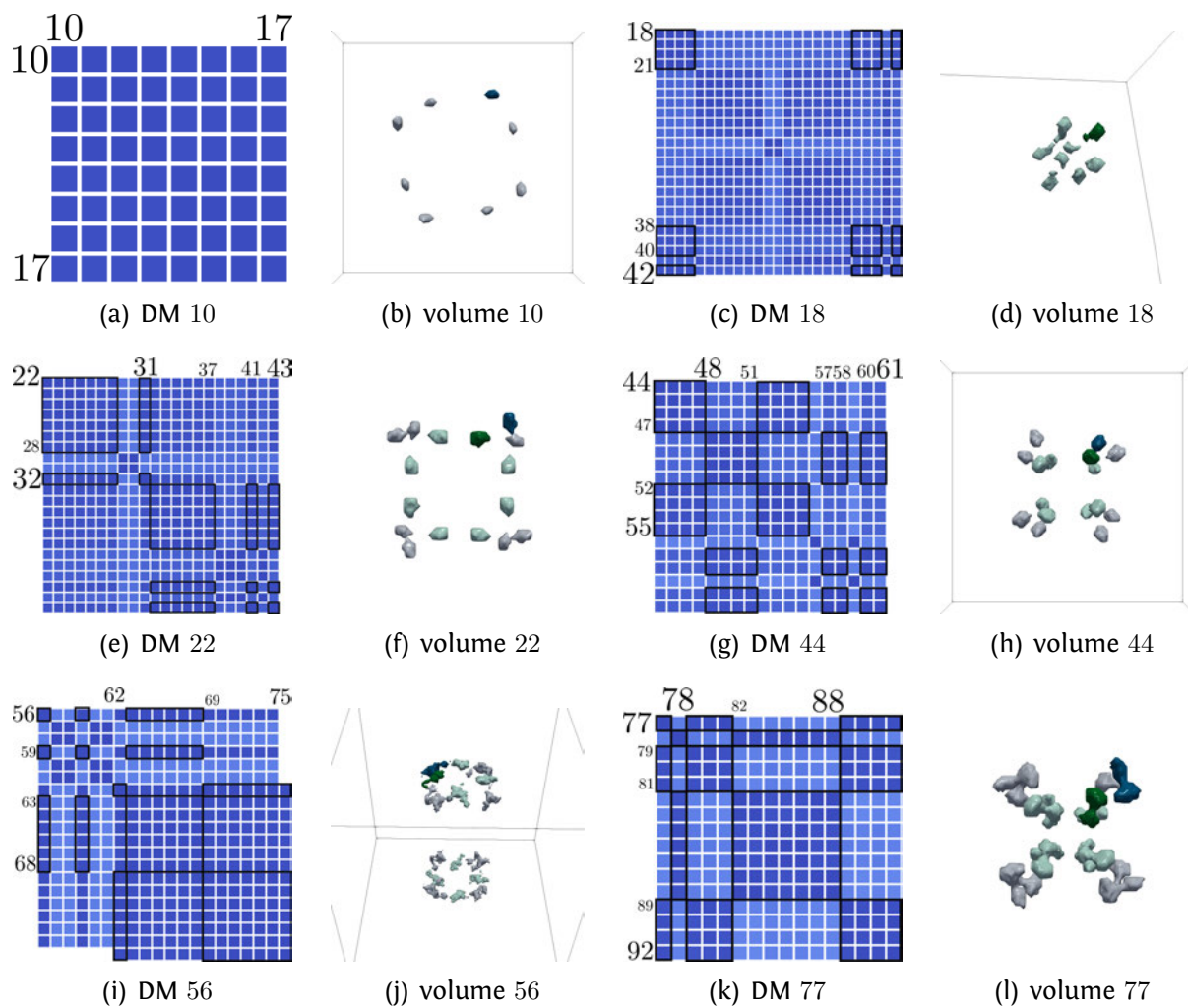


Figure 5.12: Smaller regions of the Rubisco RbcL8-RbcX2-8 complex (EMDB-1654) corresponding to the highlighted submatrices.

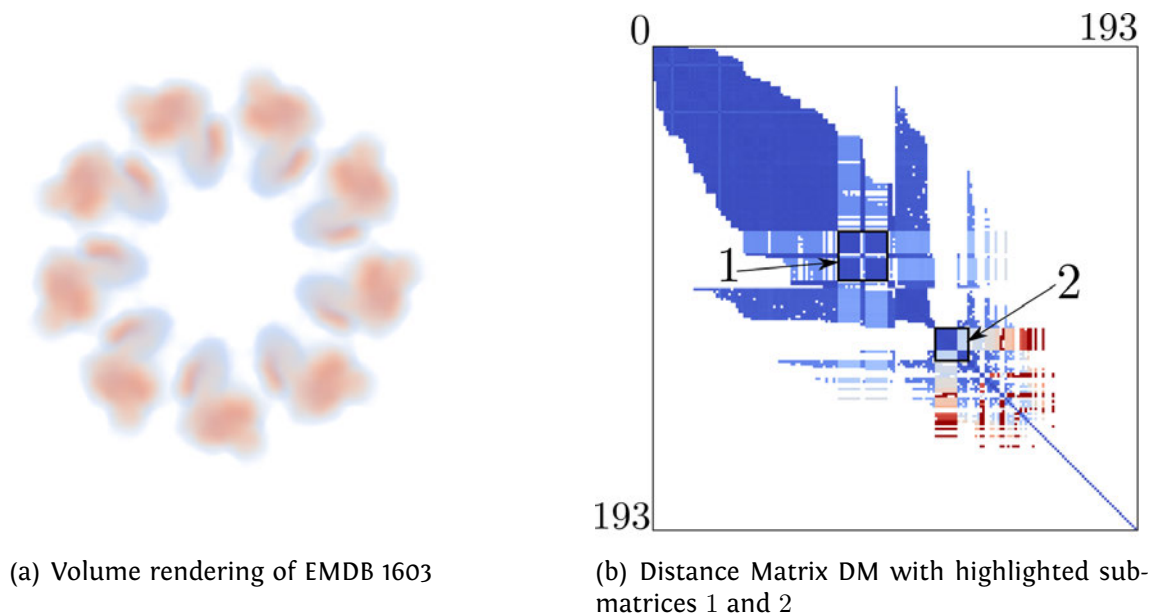


Figure 5.13: LMTED values in the DM are shown using a blue-red colormap (0  0.1)

Comparison with previous methods. Thomas and Natarajan [124] process the branch decomposition of contour trees by building feature descriptors, and use them to identify similar subtrees. The main limitation of this approach to symmetry detection is that it is based exclusively on the structure and may fail when symmetric regions do not manifest as repeating subtrees. For example, if the field is noisy, subtrees corresponding to noise have high persistence, or when the field has large flat regions. Their proposed hierarchy descriptor and similarity measure is a good estimate but not as accurate as examining the complete hierarchy. It also ignores the geometry of repeating regions leading to regions with different geometry grouped together and regions with similar geometry grouped differently. We use grid points mapped to subtrees, as an easy-to-compute substitute for geometric information. This also helps us to find symmetry in multiple scales. We use merge trees instead of contour trees and avoid computation of extremum graphs, geodesic distances, or contour shape descriptors in contrast to previous methods [125, 126]. LMTED computation is costly compared to the hierarchy descriptor based comparison [124]. We observe results similar to previous methods based on explicit geometric shape descriptors [126], but a theoretical guarantee requires further study.

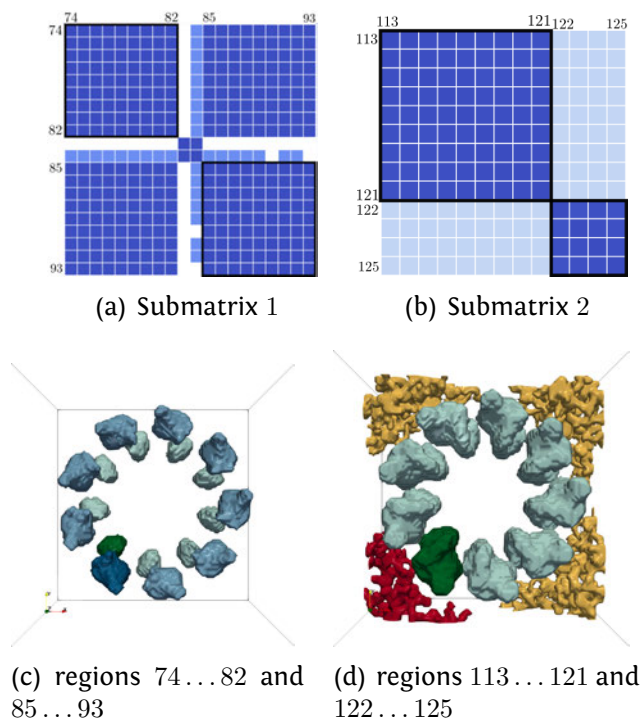
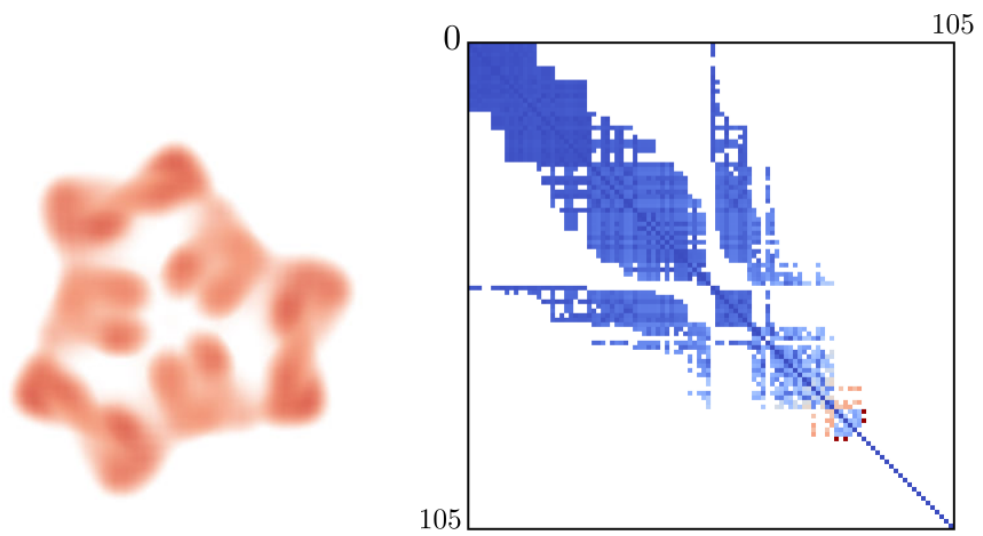
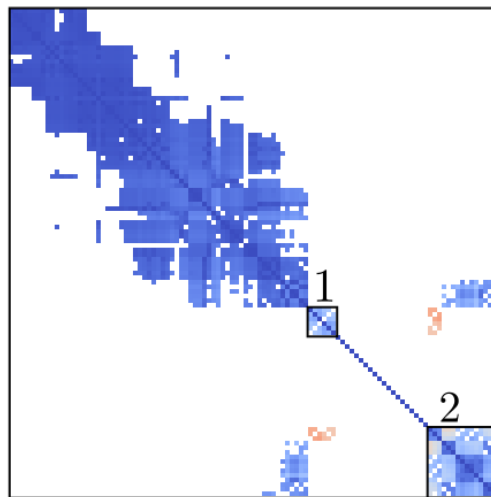


Figure 5.14: Regions and the highlighted submatrices. Each of the submatrices highlighted in Figure (a) and (b) with the corresponding sets of symmetric regions (c), (d). In (c) two representative regions are shown in dark blue and dark green respectively along with regions which are symmetric to these two colored with lighter shade of blue and green. In (d) two representative regions are shown in dark green and red respectively along with regions which are symmetric to these two colored with lighter shade of green and orange.




(a) Volume rendering of EMDB 1897

(b) Distance Matrix DM



(c) Reordered Distance Matrix DM with highlighted submatrices 1 and 2

Figure 5.15: LMTED values in the DM are shown using a blue-red colormap (0  0.2)

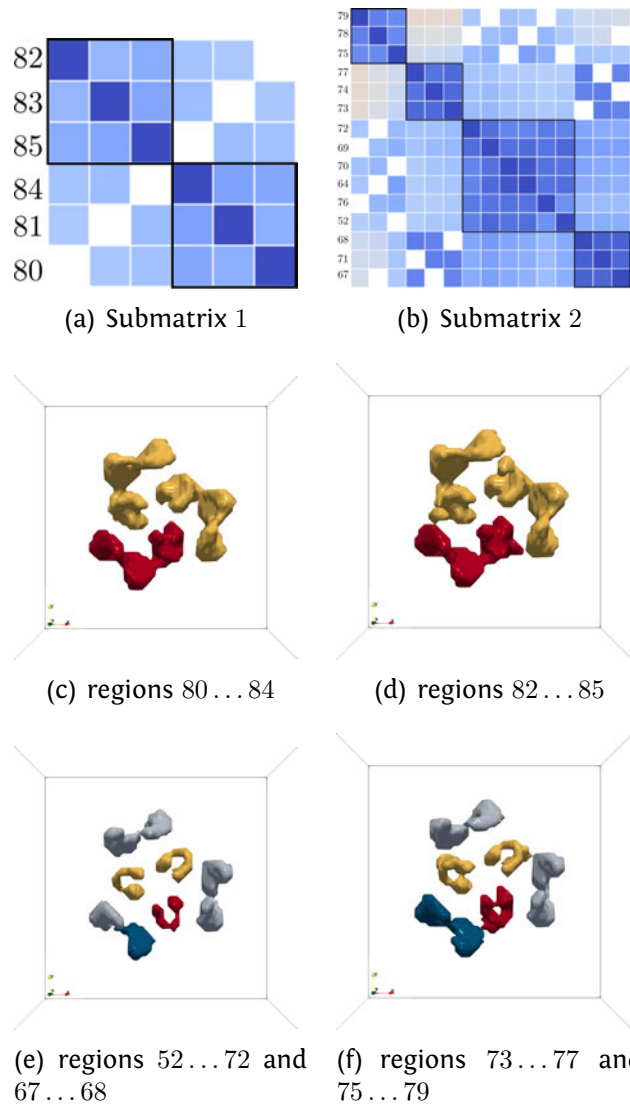


Figure 5.16: Regions and the highlighted submatrices. Figure (a) and (b) are zoomed in versions of the submatrices 1 and 2 highlighted in 5.15(c). Each of the submatrices highlighted in Figure (a) and (b) with the corresponding sets of symmetric regions (c), (d), (e), (f). In (c), (d) a representative region is colored in red and the symmetric regions colored in yellow. In (e), (f) two representative regions are shown in red and blue respectively along with region symmetric as yellow and grey.

5.4.3 Analysis of subsampling, smoothing, and topology based compression

We analyse the effects of subsampling, smoothing and topology based compression [115]. While subsampling and smoothing is applied uniformly across the domain, the effects of compression vary in different parts of the domain. We showcase how LMTED can be used to analyse these effects meaningfully.

Effects of subsampling and smoothing. Topology changes due to subsampling and smoothing are not thoroughly quantified. While previous work does present some analysis based on the MTED, it is global and not capable of providing fine-grained analysis or explain the non-monotonic variation in many cases. We present a fine-grained analysis using LMTED on a scalar field denoted as f_2 [120, Section 5.4] and we use the images of the scalar fields and the DMs from [120, Figure 14] in Figure 5.17 to illustrate the benefits.

The non-monotonic variation of the distance along a row / column can be due to multiple factors. While the subsampling and smoothing affects the number of critical points, and therefore affects the distance, it is not the only deciding factor. The distance is also affected by (a) type of critical points inserted / removed, (b) their function values, and (c) changes in persistence and pairing. We construct the DMs of the MTED and LMTED for the subsampled functions. To highlight the utility of LMTED, we pick the non-monotonic entries indexed $(3, 4), (3, 5), (3, 6)$ from Figure 5.17(d). The trees are $|T_3| = 62, |T_4| = 66, |T_5| = 62, |T_6| = 66$. We observe that $|T_3| = |T_5|$ but $D_c(T_3, T_5) > D_c(T_3, T_4)$ and $D_c(T_3, T_5) > D_c(T_3, T_6)$ even though $|T_3| \neq |T_4|, |T_3| \neq |T_6|$. Thus, size cannot explain the non-monotonicity. Also, we notice that T_3 and T_5 are structurally similar, all edits are relabels and there is negligible difference in the function values of the critical points too. The DMs (Figures 5.18(a), 5.18(b)) show small changes in the pattern, but the values are similar. The bottom-right portions of the DMs along with the values are shown in Figures 5.18(c), and 5.18(d). The diagonal entries in left portion of Figure 5.18(c) related to $(3, 4)$ shows a gradual increase, while in case of Figure 5.18(d) related to $(3, 5)$ we observe an upward spike in the last entry. The corresponding entries for MTED in both cases change gradually, even though for $(3, 5)$ the increase is higher. LMTED uses truncated persistence for all subtrees and effect of change in persistence pairings is seen only in the global comparison, causing a jump. So, the change in distance means that the subsampling has caused a change in persistence pairing when we go from resolution 4 to 5 and 5 to 6 but no such change when we go from 3 to 4. Observation of the pairings confirms this. We also saw that the pairing changes in 4 to 5

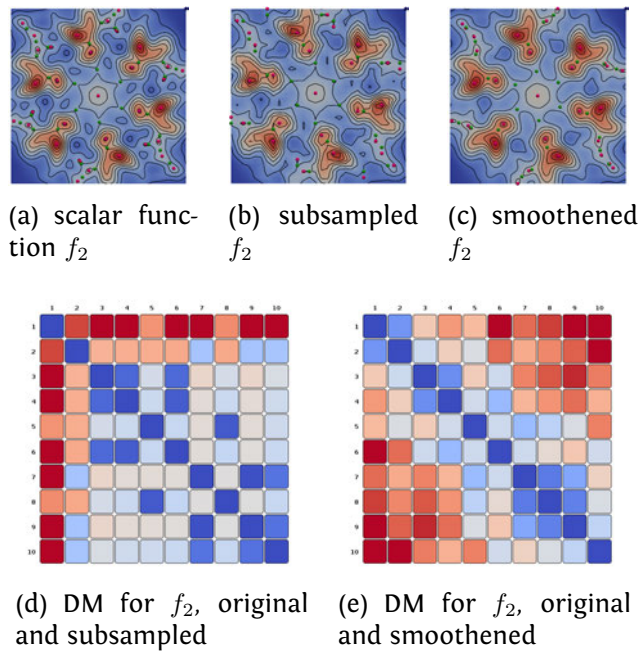



Figure 5.17: Measuring the effect of subsampling and smoothing (Images sourced from Figure 14 from [120, Section 5.4]). (a) A synthetic function f_2 sampled over a 300×300 grid. (b) f_2 subsampled down to a 30×30 grid over 9 iterations. (c) f_2 smoothed in 9 iterations. (d) DM showing distance between all pairs of subsampled datasets. (e) DMs showing distances between all pairs of smoothed functions. Row and column indices correspond to the iteration number, 1 corresponds to the lowest resolution/extreme smoothing, 10 corresponds to the original. We again use a blue-red colormap (low  high). The scales on colormaps for (h) and (k) are different

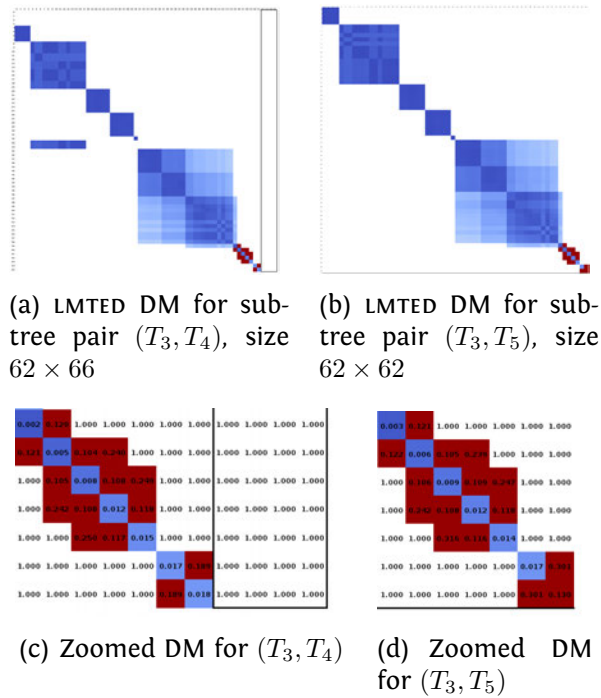



Figure 5.18: Measuring the effect of subsampling using LMTED. We use a blue-red colormap for the distances (0  0.1). Entries that are discarded due to the refinement are marked with 1.000.

was reversed from 5 to 6, thus resulting in a lower value in the entry $(3, 6)$.

Fine-grained analysing using LMTED. LMTED can be used in conjunction with MTED to quantify the changes caused by subsampling (or smoothing). This is achieved by computing MTED across all resolutions and checking if the variation is monotonic. If yes, then the subsampling is likely to have caused changes only in terms of (a) the number of critical points, (b) the function values of the critical points, and (c) persistence of critical points. If the variation is non-monotonic with a jump in the last entries of corresponding LMTED, then irrespective of other factors, there are changes in persistence pairing resulting in changed matching costs and large changes in distance. Due to the use of truncated persistence in LMTED, we can detect such changes as jumps in distances. While both MTED and LMTED may be unstable, we observe in practice that they are more discriminative than bottleneck and Wasserstein distances.

Effect of topologically controlled lossy compression. Soler et al. [115] describe a method to compress scalar fields that guarantees topology preservation. The method ensures that

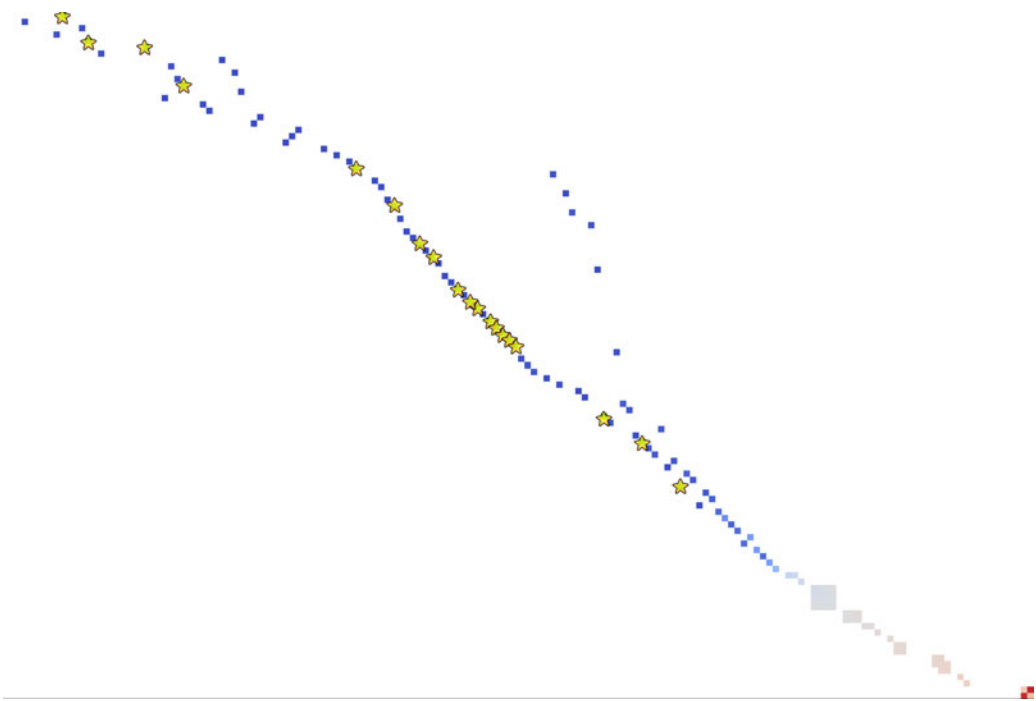



Figure 5.19: Topological effects of compression. Yellow stars in the DM for the subtree pair $T_{0.5}, T_1$ correspond to regions that remain unchanged. Distances in the DM are shown using a blue-red colormap (0  0.4).

the bottleneck distance between the persistence diagrams of the compressed and uncompressed field is less than a user specified threshold. Naturally, the method does not consider spatial or hierarchical structure since it is restricted to the persistence diagrams. We present here a fine-grained analysis of the effects of compression using LMTED. Soler et al. employ a topological compression followed by zfp. In our experiments, we use only the former. We begin by computing merge trees for both the compressed (T_c) and uncompressed data (T_u). Since the two scalar fields are defined over a common domain, we select pairs of subtrees of T_c and T_u that correspond to the same region in the domain and order them based on region size. We compute LMTED between these subtree pairs and note that as we move up the tree hierarchy, the distance remains 0.0 for some pairs. The largest among the pairs represent regions that remain unchanged post compression. Other LMTED values follow a staircase pattern, staying level for a few pairs followed by a jump in value. The jump indicates that compression has caused a change in the corresponding subtree. Thus we may identify and isolate regions where compression has no effect in terms of the function value followed by regions that are affected, and traversing the hierarchy of the merge tree lends itself to a multi-scale analysis of the effects of topological compression.

We show results of our analysis applied on AMP-Activated Protein Kinase (EMDB-1897). We apply topological compression using compression thresholds 0.5%, 1%, 2%, and compute merge trees $T_{0.5}, T_1, T_2$ using TTK [128]. To reduce the tree sizes in the experiment, we consider $T_{0.5}$ as the baseline uncompressed data. We choose regions with 100% overlap and compute LMTED. In Figure 5.20, we highlight region(s) that remain unchanged for various thresholds of compression at multiple scales together with a region that is affected due to compression. Figure 5.19 shows the DM for the subtree pair $T_{0.5}, T_1$, highlighting unchanged regions by a yellow star. We notice that 19 regions remain unchanged between $T_{0.5}$ and T_1 , and 3 regions remain unchanged between $T_{0.5}$ and T_2 . A threshold on LMTED may be used to highlight regions that are either affected or remain unaffected for various compression thresholds.

5.4.4 Spatio-temporal exploration and feature tracking

We demonstrate an application of LMTED to time-varying scalar fields, in particular for identifying and tracking features across time. We consider two scenarios – identify and track all features to provide an overview and an interactive query-driven mode for feature tracking.

In order to identify and track all interesting topological features, we begin by computing a sequence of LMTED DMs between consecutive timesteps. We apply the refinements

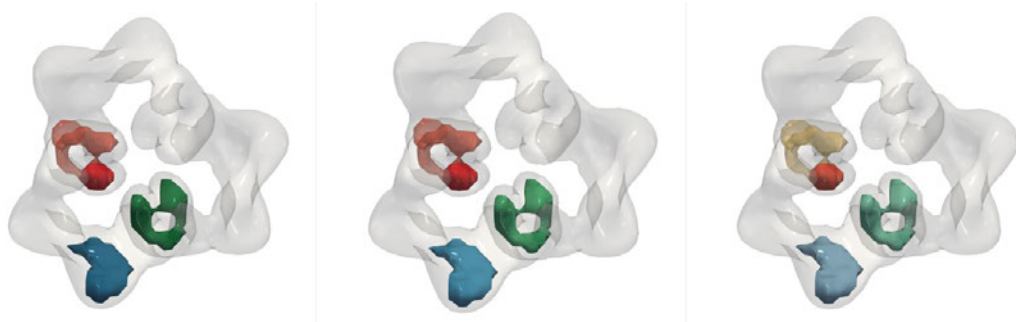


Figure 5.20: CryoEM image of AMP-Activated Protein Kinase (EMDB-1897) using different compression thresholds - 0.5%, 1%, and 2%. The region in red is the largest region that remains unaffected, the region in light red is the largest region that remains unaffected for compression threshold of 1%, and orange corresponds to 2%. Regions in shades of green are symmetric to the regions in light red and orange but are affected by the compression. The regions in shades of blue are also affected by compression. The entire protein is rendered grey and transparent for context.

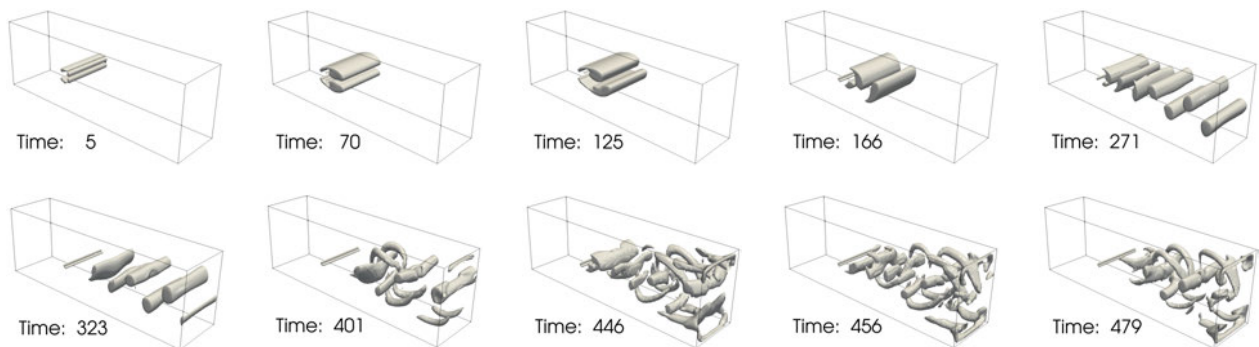


Figure 5.21: Visualizing the top k tracks in the 3D von Kármán vortex street data. The tracks are generated based on LMTEd and spatial overlaps, and sorted based on the weights of the tracks. The top tracks capture the temporal evolution of a set of primary and secondary vortices.

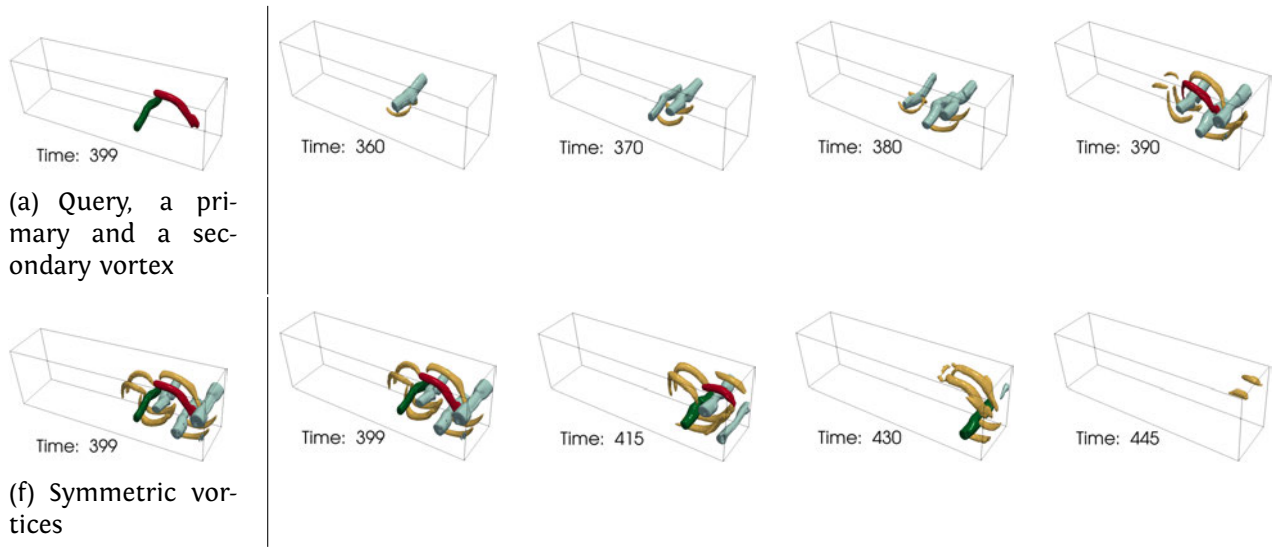


Figure 5.22: Tracking query regions in the 3D von Kármán vortex street data. A query containing a primary (green) and secondary (red) vortex in time step 399 is selected (a). LMTED is used to compute regions (light green and orange) in the same symmetry class as the query (f). All regions in the symmetry class are tracked backward in time (Top row), and forward in time (Bottom row).

described in Section 5.3.5 and compute spatial overlaps between regions that correspond to the reduced set of subtree pairs. We construct a track graph whose nodes represent each region and insert an edge between two nodes in consecutive timesteps if there is a significant overlap between the corresponding regions. Long paths in the track graph correspond to long-lived features. We visualize all long-lived features and their evolution over time including birth, death, split, and merge events. The individual tracks are also used as a starting point for further analysis. An alternative approach is to allow the user to specify one or many features within a particular timestep. We compute regions that are symmetric to the given feature, compute tracks for each of these regions, and visualize the tracks.

We demonstrate both scenarios using a 3D Bénard-von Kármán vortex street dataset. The Okubo-Weiss criterion, indicative of high vorticity regions, is sampled on a regular grid [101]. The scalar field is available on $192 \times 64 \times 48$ grid with 508 timesteps. We compute merge trees for all timesteps and simplify them using a small persistence threshold of 0.8% to remove noise. We compute LMTED on the simplified merge trees after applying the appropriate refinement steps mentioned in Section 5.3.5. The weight of an edge in the track graph is set equal to the spatial overlap (volume of overlap normalized by the volume of union) between the corresponding regions. Overlaps below a 2% threshold are considered

negligible and not included into the track graph.

We process the track graph to enumerate top tracks ordered by either the length of the track or the sum of weights (high to low). Further, short tracks (length < 10) and tracks whose sum of weights is low (< 3.0) are removed from consideration. We observe that the first track is a thin region close to the cylinder obstruction, which remains almost stationary. Other tracks that appear at the top of the list include the primary and secondary vortices as identified by [101], see Figure 5.21. These vortices are represented as isosurfaces (isovalue 0.1).

In the second scenario, we use a primary and secondary vortex from time step 399 as a query feature, see Figure 5.22. First, we compute symmetric regions within the same time step in order to highlight other primary and secondary vortices. Next, we compute tracks that contain the query regions and visualize them. We observe that in the first step LMTED can discriminate between the primary and secondary vortices and, next, it helps efficiently track the features (vortices) over time. This demonstrates the utility of LMTED in the exploration of time-varying data.

There are a few exceptional situations where LMTED is unable to discriminate between primary and secondary vortices. This happens when, say, the chosen vortex is a secondary vortex, corresponds to a leaf node in the merge tree, and matches with a leaf node that corresponds to a primary vortex. Further spatial overlap tests are necessary to identify that the two regions do not correspond to each other. To summarize, LMTED supports the generation of a good overview visualization and serves as a starting point for feature detection and tracking. Subsequent interaction and visualization tasks are often necessary and these tasks may closely depend on application specific requirements.

5.5 Summary

We described a local comparison measure (LMTED) between two scalar fields by comparing subtrees of their merge trees. The comparison measure supports local and fine-grained analysis and visualization of similarities and differences between two scalar fields. The measure satisfies metric properties and can be efficiently computed. We demonstrate its practical utility via applications to feature tracking, study of topology controlled compression, and symmetry identification.

Chapter 6

Comparing extremum graphs (PDEG) and (GWEG)

In this chapter [118] we describe comparison measures for a different topological structure called the extremum graph.

6.1 Introduction

Most of the comparative measures cater to set-based or graph-based structures as observed by Yan et.al [141] and very few have been defined for complex-based structures like Morse/Morse-Smale complex or its substructure, the extremum graph. While structures like persistence diagrams, merge trees and Reeb graphs capture the topology very well, they don't capture the geometry and all these structures need explicit augmentation in case geometry is required. On the other hand Morse/Morse-Smale complexes and extremum graphs, to some extent, capture both these aspects well and as topological descriptors they provide a holistic picture of the scalar field. The extra geometric information is crucial in many applications.

Many studies from physics (exploration of cosmic filaments [113]), biology (electron transition studies [77]), chemistry (exploration of molecular systems [12] or combustion studies [16]), material sciences (capturing pore networks [57], interstitial geometry extraction [59], study of ion diffusion in AIMD simulations [60], study of granular materials [90]), either naturally give rise to complex based structures or can be analyzed better using such complex based structures and comparison measures designed specifically for such structures would be beneficial in such scenarios.

One important reason for lack of many comparison measures for complex-based struc-

tures is their sensitivity to noise. Because they depend on the gradient field, small variations in the scalar function results in drastic changes in the structure.

6.2 Contributions and challenges

We describe the contributions and challenges in this section.

6.2.1 Contributions

In this chapter we propose two ways to facilitate comparison of scalar fields via their extremum graphs, Gromov-Wasserstein distance (GWEG) and persistence distortion distance between extremum graphs (PDEG). The measures are adaptations of Gromov-Wasserstein distance defined by Memoli [81] and the persistence distortion distance for metric graphs (PDMG) defined by Dey et.al [34] to extremum graphs. We make the following key contributions:

1. Two comparison measures GWEG and PDEG between extremum graphs.
2. Choice of suitable underlying metric and its relevance to facilitate the two measures.
3. Theoretical properties.
4. Experimental comparison of the two measures with emphasis on ease of interpretation.
5. Experiments to demonstrate the practical value of the measures using various applications.

6.2.2 Challenges

We discuss various challenges in adapting d_{PD} and d_{GW} extremum graphs and their solutions. Firstly, we discuss the common challenges which are to be addressed for both measures then we move to the specific ones.

6.2.2.1 Common challenges

Choice of the extremum graph: Extremum graph can be defined and computed in multiple ways. It can be defined as a sub-structure of the Morse-Smale complex and thus computed by first computing the Morse-Smale complex followed by extraction of the graph. Further the Morse-Smale complex can be discrete ([98]) too. It can also be defined directly using just the maxima and $n - 1$ -saddles and their connectivity and computed likewise. There

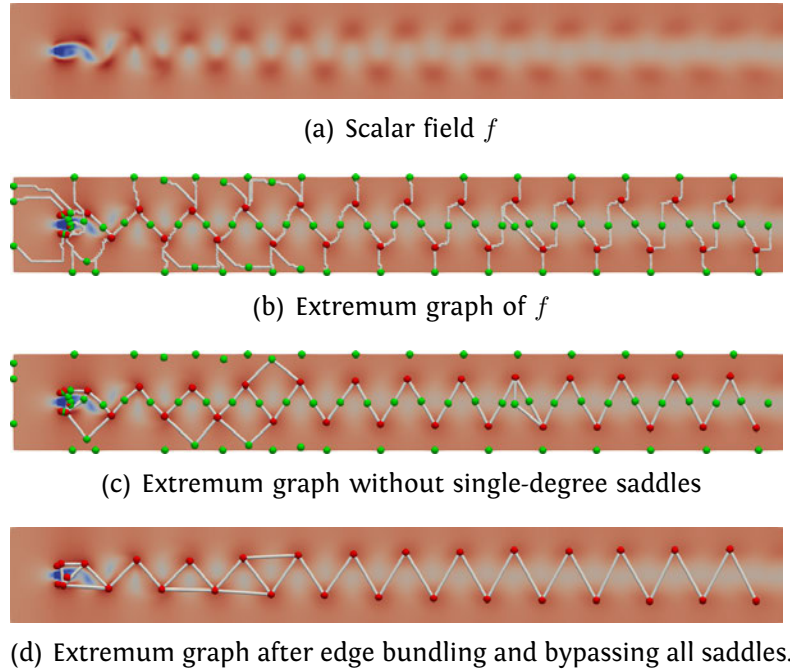


Figure 6.1: Available extremum graph alternatives. (a) Scalar field f showing 3rd time-step of the 2D vortex street. (b) Extremum graph of f embedded in the domain with all possible connections. (c) Extremum graph of f with single degree saddle removed. (d) Saddles are known to be unstable compared to the maxima, so all the saddles can be bypassed and discarded so that only the maxima remain in the base point set for computation of PDEG.

can be many single degree $n - 1$ -saddles (Figure 6.1(b)), also between two maxima there can be multiple $n - 1$ -saddles (Figure 6.1(c)). All these saddles can be retained as it is or except the saddle with the highest function value, rest can be removed (by edge bundling) to avoid multiple paths. If we retain only one saddle between two maxima which has the highest function value, it will lead to destruction of cycles of length 4 since such cycles will be reduced to a path of length 2. Such cycles occur at high activity regions and may be important based on the application. They don't affect GWEG much since it depends mainly on the all pairs shortest path (APSP) distances, but it has a substantial effect on PDEG since it considers both 0-th and extended persistence diagrams. This leads to two variants, one where only one saddle is retained and 0-th persistence diagram is considered (for GWEG) and the other where all such saddle except single degree saddles are retained and both persistence diagrams are considered (for PDEG).

Choice of the metric: The extremum graph if considered with just the location of the nodes, comes with the usual metric, *i.e.* the euclidean distance d_E . If the graph is embedded in the domain it comes with the geodesic distance d_{GD} since it captures some geometry of

the scalar field. These two might seem like the obvious choice, but these metrics while capturing the connectivity of the graph, if used alone, fail to capture the properties of the scalar function since they don't consider the function values at the critical points. Other options include, the function metric d_f defined by Bauer et.al [7], a variation d'_f *modified function metric* where we add the intervals instead of taking a single interval, and then take minimum over all paths, or a combination of one of these. Wherever the scalar fields are to be directly compared, the function values are relevant, the function metric d_f or its variation d'_f can be used. If the scalar field is constructed to be a substitute to derive connectivity, then d_E or d_{GD} is a better choice. While there is no rigorous justification, we make the case for these choices experimentally in further sections.

6.2.2.2 Challenges specific to persistence distortion distance

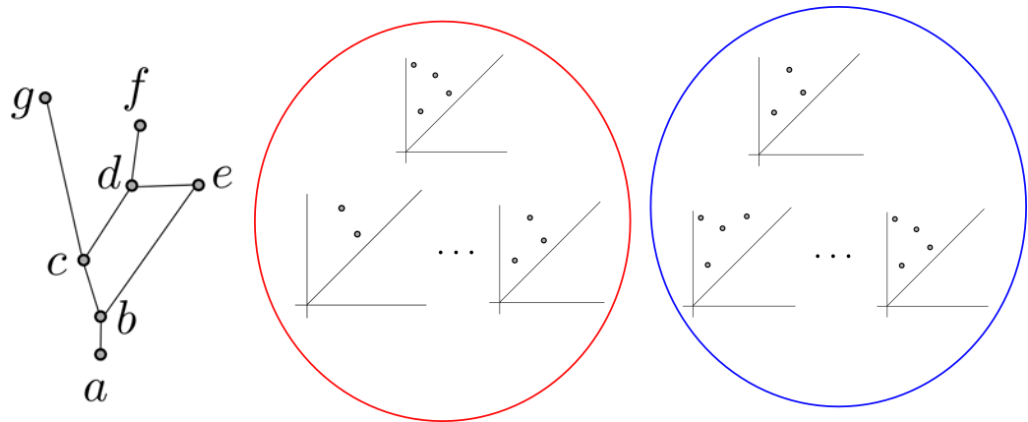
Choice of d_{PD} version: There are two version of d_{PD} , *continuous* and *discrete*. The continuous version considers all points (including the points internal to the arcs) as base points, the discrete version considers only the nodes of the graph as base points. If m denotes the total number of nodes and arcs of the graph and n denotes the number of nodes, the running time of continuous d_{PD} is given by $O(m^{12} \log m)$ and the running time of discrete d_{PD} is given by $O(n^2 m^{1.5} \log m)$. For an extremum graph, the node set consisting of the maxima and $n - 1$ -saddles are more meaningful compared to the regular points. Also, the running time of continuous d_{PD} though polynomial is prohibitively expensive to be useful in scenarios which need interactive visualization. Due to these two reasons, we choose the discrete version.

Choice of base points: The saddles in general are known to be unstable (Figure 6.1(d)) compared to the maxima, so we restrict the base points to be just the maxima of the extremum graph. This also speeds up the running time in practice.

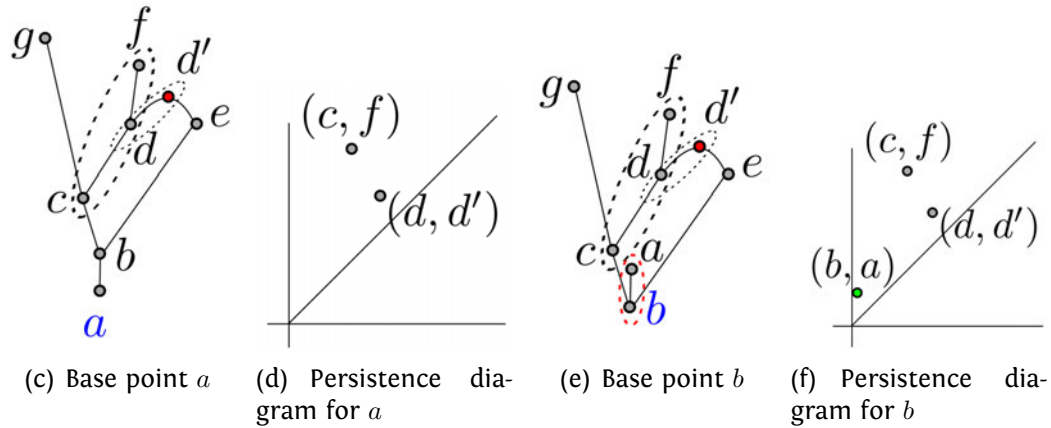
6.2.2.3 Challenges specific to Gromov-Wasserstein distance

Choice of weight matrix W , probability distribution p and loss function L : The weight matrix W has entries for a pair of nodes i, j computed as the all-pairs shortest path distance using the appropriate metric. We use uniform distribution as it has been used in many applications before. Since we retain only the extrema we want to emphasize all those extrema equally, thus opting for a uniform distribution. We use quadratic loss function as defined in Section 3.6.3.

With these challenges addressed, we proceed to define PDEG and GWEG as follows.



(a) Original graph (b) Computing PDEG as Hausdorff distance between sets of persistence diagrams



(c) Base point a (d) Persistence diagram for a (e) Base point b (f) Persistence diagram for b

Figure 6.2: Computing PDEG. Figure (a) shows the extremum graph, (c), (e) show different choice of base points, (d), (f) show corresponding persistence diagrams and (b) illustrates two such sets built in the same way, computation of Hausdorff distance between them.

6.3 The persistence distortion (PDEG) and Gromov-Wasserstein (GWEG) distances for extremum graphs

Given two metric extremum graphs $\mathcal{G}_1 = (G_1, d_{G_1})$ and $\mathcal{G}_2 = (G_2, d_{G_2})$ where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are extremum graphs and d_{G_i} is one of the metrics discussed in Section 6.2.2.1. Let M_1, M_2 be the maxima sets of G_1, G_2 .

Consider a point $\mathbf{s} \in G$ as a base point. For any point $x \in G$, we can use the distance from \mathbf{s} , $f(x) = d_G(\mathbf{s}, x)$, to construct the 0, 1-dimensional persistence diagrams $\mathcal{D}_{0f}, \mathcal{D}_{1f}$.

Consider base points $\mathbf{s} \in M_1$, $d_{G_1, \mathbf{s}} : G_1 \rightarrow \mathbb{R}$ and $\mathbf{t} \in M_2$, $d_{G_2, \mathbf{t}} : G_2 \rightarrow \mathbb{R}$ and corresponding persistence diagrams as $P_{\mathbf{s}}$ and $Q_{\mathbf{t}}$. Map the metric graphs \mathcal{G}_1 and \mathcal{G}_2 to

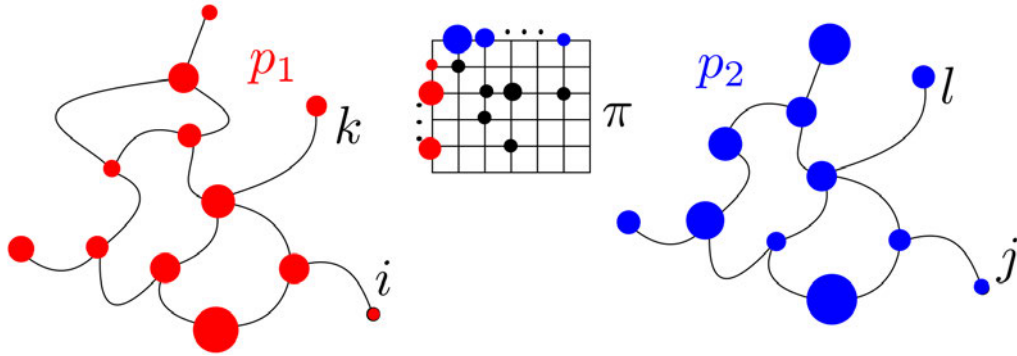


Figure 6.3: Illustrating computation of GWEG. $W_1(i, k)$ and $W_2(j, l)$ comes from the APSP entries, p_1, p_2 are probability distribution supported by the nodes of the graphs. $C_{i,j}$ and $C_{k,l}$ comes from the coupling matrix π .

the set of (infinite number of) points in the space of persistence diagrams \mathbb{D} given by $\mathcal{C} := \{P_s | s \in M_1\}$ and $\mathcal{F} := \{Q_t | t \in M_2\}$.

Definition PDEG: The persistence-distortion distance between two metric extremum graphs \mathcal{G}_1 and \mathcal{G}_2 , denoted by $d_{PDE}(\mathcal{G}_1, \mathcal{G}_2)$, is the Hausdorff distance $d_H(\mathcal{C}, \mathcal{F})$ between the two sets \mathcal{C} and \mathcal{F} where the distance between two persistence diagrams is measured by the bottleneck distance.

$$d_{PDE}(\mathcal{G}_1, \mathcal{G}_2) = d_H(\mathcal{C}, \mathcal{F}) = \max\{\max_{P \in \mathcal{C}} \min_{Q \in \mathcal{F}} d_B(P, Q), \max_{Q \in \mathcal{F}} \min_{P \in \mathcal{C}} d_B(P, Q)\}. \quad (6.1)$$

Figure 6.2 illustrates how PDEG can be computed.

Similarly, given two metric extremum graphs of the form $G_1(V_1, W_1, p_1)$ and $G_2(V_2, W_2, p_2)$ with function metrics d_f, d_g respectively, where W_1, W_2 are the matrices containing the APSP entries for d_f, d_g and p_1, p_2 are uniform distributions,

Definition GWEG: The Gromov-Wasserstein distance between two metric extremum graphs G_1 and G_2 , denoted by $d_{GWE}(G_1, G_2)$, is

$$d_{GWE}(G_1, G_2) = \frac{1}{2} \min_{C \in \mathcal{C}} \sum_{i,k \in [n_1], j,l \in [n_2]} |W_1(i, k) - W_2(j, l)|^2 C_{i,j} C_{k,l} \quad (6.2)$$

where $[n_1], [n_2]$ are index sets corresponding to G_1, G_2 , $|V_1| = n_1, |V_2| = n_2$. $C_{i,j}$ and $C_{k,l}$ comes from the coupling matrix π . Figure 6.3 illustrates how GWEG can be computed.

6.3.1 Properties

We discuss the properties of both PDEG and GWEG in this section.

6.3.1.1 Properties of PDEG

Stability: Dey et.al. [34, Theorem 3] prove that persistence distortion distance is stable w.r.t the Gromov-Hausdorff distance.

$$d_{PD}(\mathcal{G}_1, \mathcal{G}_2) \leq 6d_{GH}(\mathcal{G}_1, \mathcal{G}_2) \quad (6.3)$$

This proof considers the fact that the two graphs are metric graphs and Gromov-Hausdorff quantifies the metric distortion of any pair of metric spaces.

In the present situation, the extremum graphs are metric graphs, but they are highly sensitive to the changes in the scalar function. So this proof cannot be extended to construct a proof similar to stability of the bottleneck distance where the distance is always bounded by the infinity norm of the two scalar functions. We conjecture that there are scalar fields where small changes lead to a drastic change in the connectivity of the extremum graphs leading to high metric distortion. But as long as the extremum graphs are stable, PDEG would also be stable. This is not always true but it is a valid assumption in cases like time-varying scalar field where the extremum graphs may not change drastically between two time-steps.

Discrimination: PDEG is more discriminative than the bottleneck distance, since it considers the extremum graph as the underlying topological structure and depends on an optimization involving many persistence diagrams computed from the base points instead of the original function.

6.3.1.2 Properties of GWEG

Stability: The original Gromov-Wasserstein d_{GW} distance will be lower bounded by the Gromov-Hausdorff distance d_{GH} since d_{GW} satisfies additional constraints. While Mémoli [81] proves that shape signatures/invariants are quantitatively stable under d_{GW} the same cannot be directly assumed in case of extremum graphs. We again conjecture that given the sensitive nature of the extremum graph w.r.t perturbations in the original scalar fields, stability w.r.t to the fields may not be possible. But if the graphs are stable, like PDEG the GWEG would also be stable.

Discrimination: Since d_{GW} is lower bounded by d_{GH} , in theory, GWEG should be more discriminative compared to PDEG (within the constant multiplicative factor of 6 due to Dey et.al. [34, Theorem 3]) assuming the graphs are stable. But in practice we need to explore the possibility by ensuring stable extremum graphs.

6.3.2 Algorithm and computation

We discuss the preprocessing steps and algorithm used to compute both the distances.

6.3.2.1 Computation of PDEG

Preprocessing: We list the preprocessing steps along with justification.

- The implementation provided by Dey et.al [34] is specific to meshes which arise in shape matching application and hence they sub-sample the mesh to reduce computation. In case of extremum graphs which are typically small compared to such meshes the sub-sampling can be skipped.
- They consider the edge weights of the mesh edges using euclidean distance which is appropriate for shapes. In case of extremum graphs different metrics are required based on application such as d_f and so on.
- In case of d_f , there can be nodes which have the same distance w.r.t a base point, the simple version of simulation of simplicity used by Dey et.al [34] to break ties lead to contour tree instead of split tree. Since the computation is dependent only on persistence diagrams, this is not a huge problem, but we order nodes to consistently construct split trees.
- We remove single-degree saddles, since they occur mostly on the boundaries of the data and are not important.
- We do edge-bundling depending on the application, in that case we add one more step to store cycles of length 4 which would be eliminated due to edge bundling so that extended persistence diagrams are calculated correctly.

Algorithm: After these initial steps, computation of PDEG follows the same algorithm given by Dey et.al [34] and has the same analysis in terms of running time $O(n^2 m^{1.5} \log m)$.

6.3.2.2 Computation of GWEG

In contrast with the extensive preprocessing required for PDEG, the computation of GWEG only depends on the APSP matrix (W) and a set of probabilities (p) corresponding to the nodes of the extremum graph. Once the APSP matrix is constructed. In general GWEG is NP-Hard, but many heuristics have been proposed with run time complexity of $O(n^3)$. We compute GWEG using the implementation provided by the python optimal transport (POT) library [48].

6.3.3 Discussion

There are some crucial differences between the two comparison measures. Understanding will help in choosing the appropriate measure based on the requirement of the particular application.

Correspondences: PDEG is based on Hausdorff distance between a set of points which are persistence diagrams themselves where the individual distance is measured by bottleneck distances. So we finally get the distance as the “weight” of one edge and since Hausdorff distance does not provide one to one correspondences between the sets, this would eventually mean that it also does not provide such correspondences between the nodes of the graph. Even when the Hausdorff distance is replaced by other distances which provide correspondences, we still get them for a partial set of nodes which represent the features of the base point function, many of the nodes of the extremum graph may become “regular” points for base point functions. From Figures 6.4(a),6.4(b) we see that nodes labelled 1, 2, 3, 5, 6, 7 and so on become “regular” points w.r.t base point function from the node 0.

GWEG on the other hand provides a coupling which optimizes the distance, which has correspondences but the correspondences between the nodes are probabilistic or “soft”, i.e. there can be multiple nodes mapped to one and vice versa. This is sometimes desirable when the application involved requires such one-to-many or many-to-one mappings but not in cases where the mapping needs to be bijective. In case a one-to-one mapping is needed, the node with highest probability can be chosen.

Transparency: The computation of PDEG is straightforward and with an embedding of the graph as per the chosen metric, changes in the distance can be traced back to particular changes in the graph with some effort. This can also be inferred from the stability of PDEG w.r.t Gromov-Hausdorff distance which measures to what extent two metric spaces differ. This gives the user a better understanding of the distance.

GWEG on the other hand is oblivious to the structure of the graph as it works on an APSP matrix, so it is difficult to infer how changes in the graph affects the distance without substantial knowledge of the internals, also GWEG solves an optimization problem using Sinkhorn iterations [93]. That also mandates understanding of the internal working of the optimization to interpret the distance.

6.4 Applications

In this section we provide illustrative examples to understand and interpret both PDEG, GWEG and showcase their utility in applications like periodicity detection.

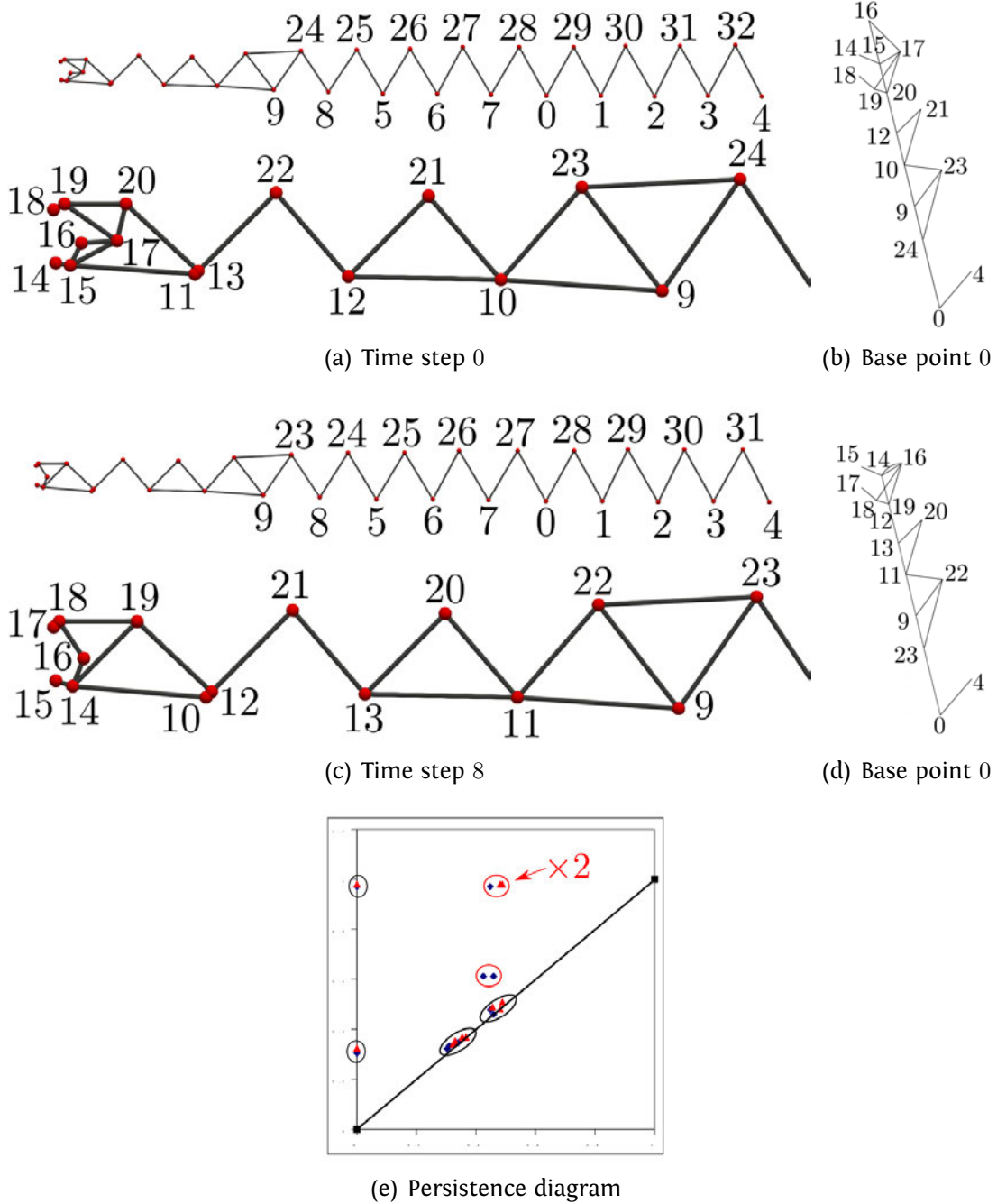


Figure 6.4: Understanding pDEC. Figures (a),(c) represent labeled extremum graphs for time steps 0, 8, the area where vortices are formed is zoomed in to avoid clutter. Figures (b),(d) represents the graphs w.r.t the base point 0. Figure (e) represents the persistence diagrams (blue points for 0, red points for 8). The points inside black ellipses have negligible distances, while points inside the red ellipses are unmatched.

6.4.1 Understanding the comparison measures

We illustrate the comparison measures PDEG and GWEG by considering a pair of time steps from the 2D von Kármán vortex street data set. Specifically we pick the time steps given by 0 and 8, with $\varepsilon = 0\%$ from the graphs in Figure 6.8 we observe that

$$d_{PDE}(0, 8) = 0.36791$$

$$d_{GWE}(0, 8) = 0.00061$$

We analyse both the measures to explain how the computation happens and why there might be such a discrepancy.

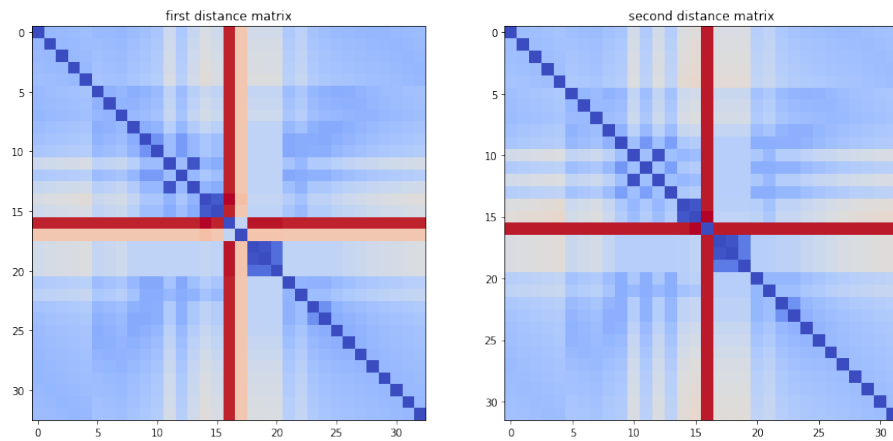
Figures 6.4(a) and 6.4(c) show the extremum graphs constructed for two time steps 0, 8. The single degree saddles have been removed, edge bundling has been applied, cycles of length 4 which have been eliminated by edge bundling have been stored separately, and saddles have been bypassed while taking their contribution into account. We use d_f as the metric.

Figures 6.4(b) and 6.4(d) show the graphs reoriented w.r.t the base point labelled 0 and with nodes contributing to 0-th and extended persistence diagrams labelled. This is the base point for both the time steps which gives the PDEG and hence this is considered for our analysis.

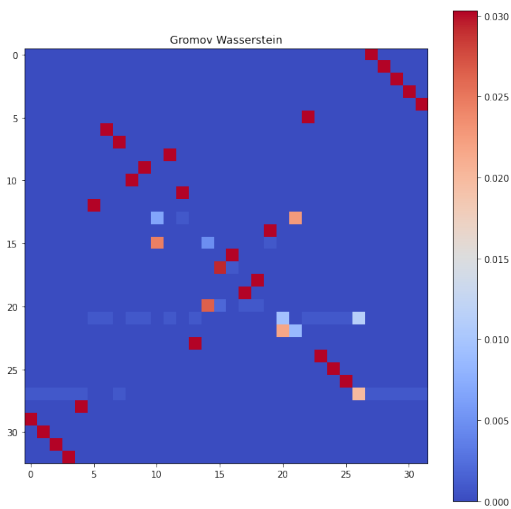
Figure 6.4(e) shows the persistence diagrams of both time steps 0, 8 overlapped. We represent the diagram for time step 0 with blue rhombuses and the diagram for time step 8 with red triangles. We highlight two sets of points using black and red ellipses. The points in the black ellipses are close to each other and their contribution to the bottleneck distance will be negligible. The points in the red ellipses are unmatched. In the case highlighted with $\times 2$, there are two red points but only one blue point. The persistence of these three points add up resulting in a high PDEG.

By constructing the graphs with respect to base points and analysing the persistence diagrams, we can explain the results of PDEG.

Figure 6.5(a) shows the all pairs shortest path (APSP) matrices for time steps 0, 8. We observe that while there is some difference around the row/column 16, the matrices mostly look the same. By computing APSP we tend to ignore the cycles which results in a low distance in case of GWEG. PDEG without the extended persistence diagram which captures the 1-cycles is also low (0.02479) confirming this. Figure 6.5(b) shows the coupling matrix which provides the optimal coupling. As mentioned before the “mapping” is probabilistic,



(a) All pairs shortest path matrices for time steps 0, 8



(b) GWEG mapping for 0, 8

Figure 6.5: Understanding GWEG. The similarity in the APSP matrices show that some information is lost especially related to the cycles which leads to low GWEG value.

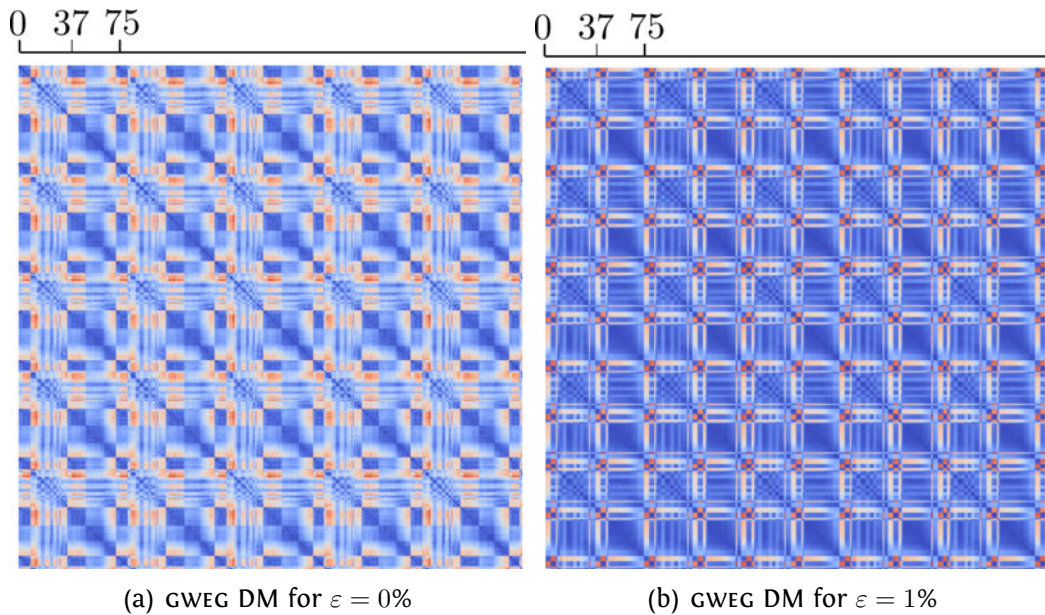


Figure 6.6: Distance matrices for GWEG for two simplification thresholds 0%, 1% We show the matrix for time steps 0 – 380

higher probability value implies “strong” mapping.

In general GWEG may not be amenable to easy interpretation, since there are cases where despite the matrices being dissimilar, GWEG may still be low. We explore this aspect in the future.

6.4.2 Periodicity detection

Earlier studies of the 2D von Kármán vortex street dataset like Sridharamurthy et.al. [119, 120] and Narayanan et al. [88] have successfully identified periodicity in the dataset. They detect both a half period of 38 and the full period of 75. We also repeat the same experiment to showcase the utility of both PDEG and GWEG. Towards this, we construct extremum graphs for all the time steps, compare the extremum graph of time step 0 with the remaining 1000 time steps of the dataset. We show the plots for both PDEG and GWEG for 500 steps, the rest of the graph is similar and we restrict it to 500 to avoid clutter. Each graphs contains approximately 40 nodes since only maxima are considered. From the top plot in Figure 4.10 we see that GWEG detects both the periods successfully irrespective of the simplification threshold, but we see that the distance increase with the threshold. We also show the distance matrices for two simplification thresholds 0%, 1% for GWEG in Figure 6.6. Note that while the distances for 1% might be higher the variation is smoother compared to the 0% as can be seen from Figures 6.6(a), 6.6(b). We also show the distance matrix of PDEG for

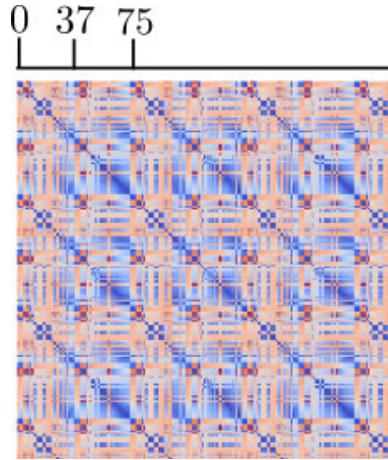


Figure 6.7: Distance matrices for PDEC for simplification threshold 1% We show the matrix for time steps 0 – 250

the simplification threshold of 1% in Figure 6.7. In case of the rows 2,3 in Figure 6.8 we see that while PDEC detects both half and full periods with simplification threshold 1% in case of 0% when the extended persistence diagram is used, it fails to detect the half period. We also observe that the graphs have a lot of spikes, and the two graphs representing only extended persistence and combination of 0 and extended persistence coincide.

These anomalies are due to the following reasons. Extremum graph is not as stable as merge trees/contour trees and small changes in the function value might add new features which may even have high persistence values in the sense of extended persistence. Also the base point function results in split trees which are simple and have features which have small persistence resulting in low bottleneck distances for the 0 persistence diagrams causing the extended persistence to dominate the bottleneck distances. These factors combine resulting in spiky behaviour and also the missing of the half period. In other words while PDEC is stable with respect to changes in the graph, it is not stable with respect to changes in the scalar function.

6.4.3 Comparing pore networks

Comparing pore networks in materials is an extremely important problem. Such comparison helps in designing new materials via computation and study its properties before actual physical design. It is crucial to find structures in such hypothetical materials designed computationally which are similar to well-known structures with desirable physical properties.

Lee et.al. [69] study similarity between pore structures in nanoporous materials. They

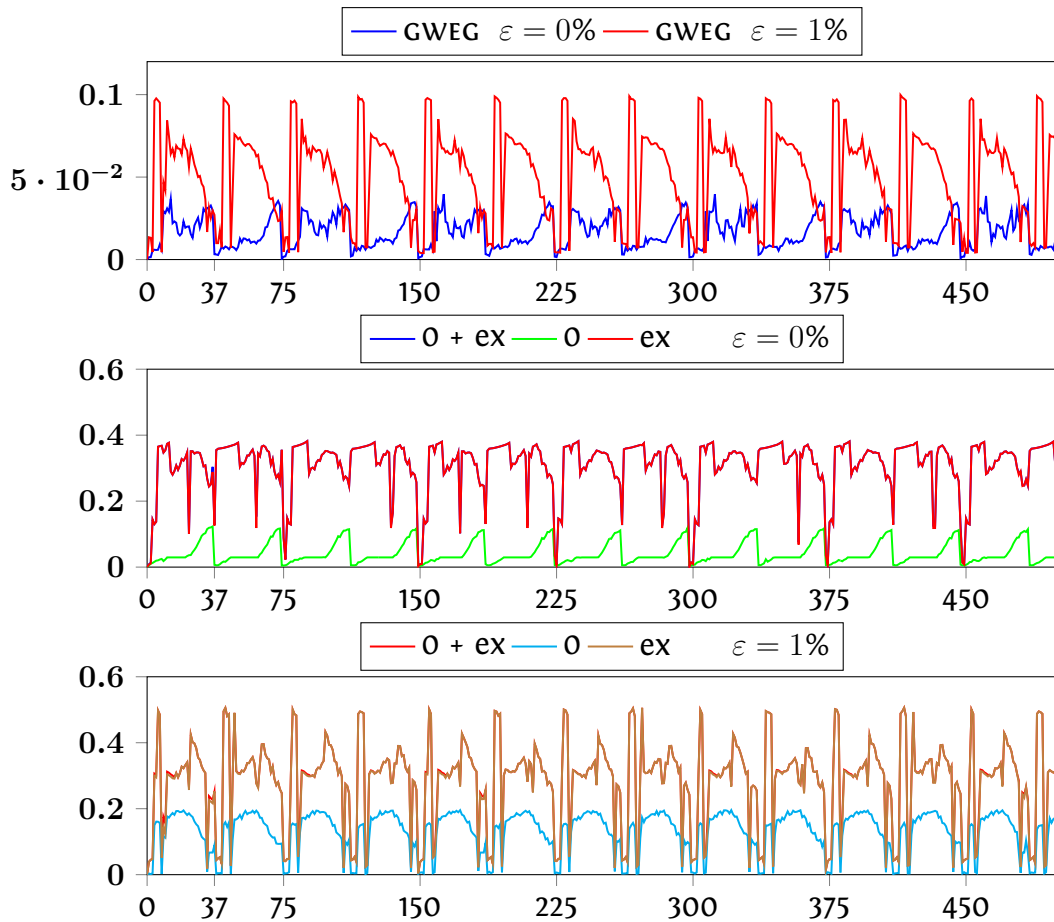


Figure 6.8: Comparing distance measures on the von Kármán vortex street dataset. (top) Plot of GWEG distance measures between the first time step and others for simplification threshold values of 0%, 1%. (rows 2-3) show PDEG for simplification threshold values of 0%, 1% with PDEG computed using both 0 and extended persistence diagrams, using only 0-persistence diagram and using only extended persistence diagram. Note that in row 1 the scale is an order of magnitude less.

use zeolite structures from IZA¹, hypothetical PCOD database² containing around 3000000 zeolites and CoRE-MOF database³. Typically the materials are represented by a unit cell which forms the foundation block which can be repeated in 3D to build the actual material. Since these cells vary in size depending on the material, to make the comparison meaningful, they first build super cells from the unit cells so that all the cells are roughly of the same size. Then they use persistence landscape distance [19] to compare their persistence barcodes which are built by using VR-complexes of the pore surfaces corresponding to particular probe molecules.

We start from constructing the super cells in similar fashion. Then we construct distance fields for these structures using Zeo++ (by Martin et.al [74]) and compute extremum graphs which gives the structure of the pore network. In this case we choose to construct minimum graphs since they capture features that are similar to those captured by the VR-complexes in contrast to the maximum graphs we use in periodicity detection. Since we want to capture the prominent cycles which are captured by the VR-complex, we use a high persistence threshold of around 20% to eliminate many of the smaller cycles. The distance field serves as a placeholder to aid the computation of the pores but in contrast to the 2D and 3D vortex street, the distance field has no meaningful physical interpretation when it comes to the properties of the pore. So instead of the function metric d_f we choose euclidean distance as the metric in this experiment. Then we compare these structures using both PDEG and GWEG.

We report some preliminary results, we compare a set of materials found to be similar by Lee et.al. [69] (Figure 3). We construct the minimum graphs of the materials (AFUPEX, SEHSUU, SEHTEF) as shown in Figure 6.9 and report both PDEG and GWEG.

We use euclidean distance and geodesic distance as underlying metrics. The PDEG and GWEG values for euclidean distance are as follows

$$d_{PDE}(\text{AFUPEX}, \text{SEHSUU}) = 0.104664$$

$$d_{PDE}(\text{AFUPEX}, \text{SEHTEF}) = 0.119698$$

$$d_{PDE}(\text{SEHSUU}, \text{SEHTEF}) = 0.194006$$

$$d_{GWE}(\text{AFUPEX}, \text{SEHSUU}) = 0.017195$$

¹<http://www.iza-structure.org/databases/>

²<https://zenodo.org/record/4030232>

³<https://zenodo.org/record/3370144>

$$d_{GWE}(AFUPEX, SEHTEF) = 0.015988$$

$$d_{GWE}(SEHSUU, SEHTEF) = 0.027770$$

The PDEG and GWEG values for geodesic distance are as follows

$$d_{PDE}(AFUPEX, SEHSUU) = 0.118493$$

$$d_{PDE}(AFUPEX, SEHTEF) = 0.120278$$

$$d_{PDE}(SEHSUU, SEHTEF) = 0.250497$$

$$d_{GWE}(AFUPEX, SEHSUU) = 0.032347$$

$$d_{GWE}(AFUPEX, SEHTEF) = 0.017636$$

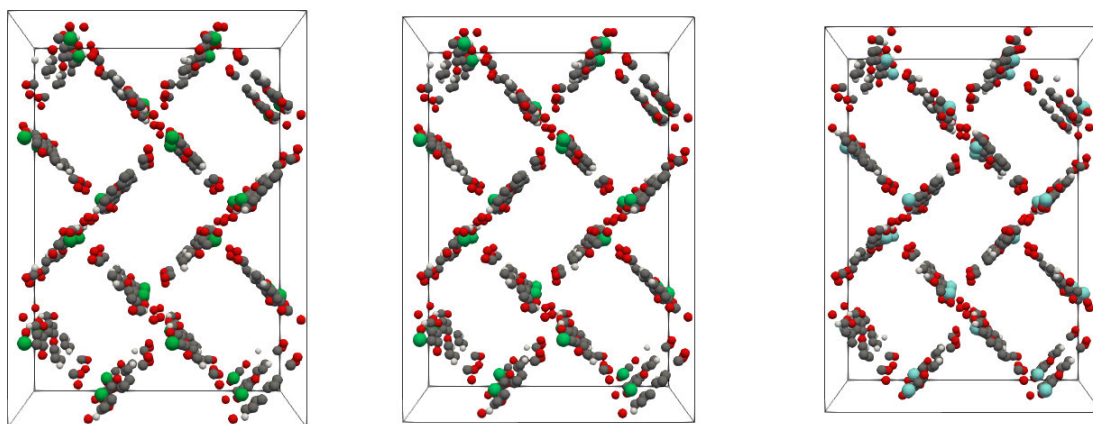
$$d_{GWE}(SEHSUU, SEHTEF) = 0.035498$$

We observe that in general PDEG is always an order of magnitude greater than GWEG as it is the case in periodicity detection too. While we expect that the distance between these zeolites to be similar in case of PDEG there is some difference, e.g. $d_{PDE}(SEHSUU, SEHTEF)$ is almost two times the distance between the other two pairs in both euclidean and geodesic distance. One crucial aspect which might be influencing the distance is the persistence simplification parameter which might have to be different for different data set instead of 20% which we have applied to all. We explore this in the future work.

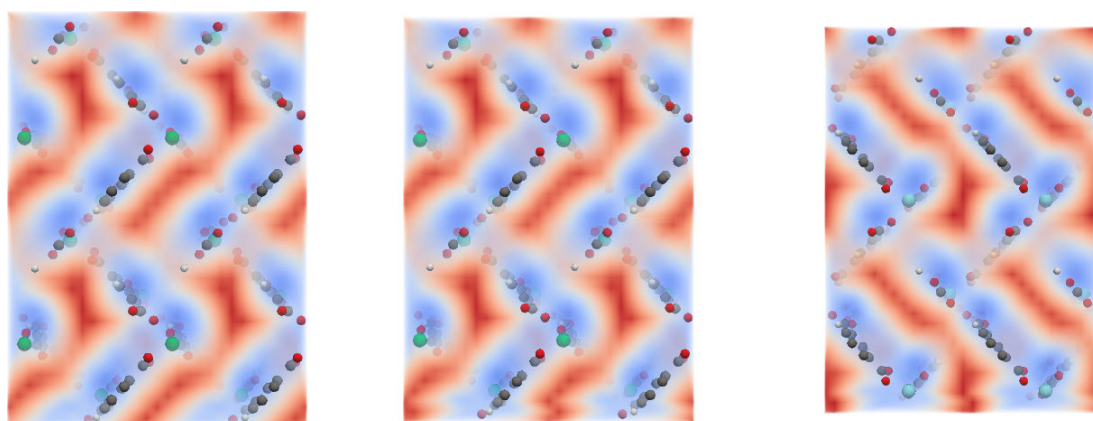
Comparison with Lee et.al [69]: Lee et.al [69] uses persistence homology where each pore is represented by a 1-cycle in the VR-complex. While they do not provide the exact distance values we conjecture the values to be very small. In our case we use minimum graphs, we assume that to make a fair comparison, we should be able to retain all the 1-cycles which represent the pores at the persistence threshold which we use. In some cases it may not be so if we use a high persistence threshold. On the other hand, lower thresholds lead to graphs with too many nodes and arcs causing many spurious cycles. We explore this too in future.

6.5 Summary

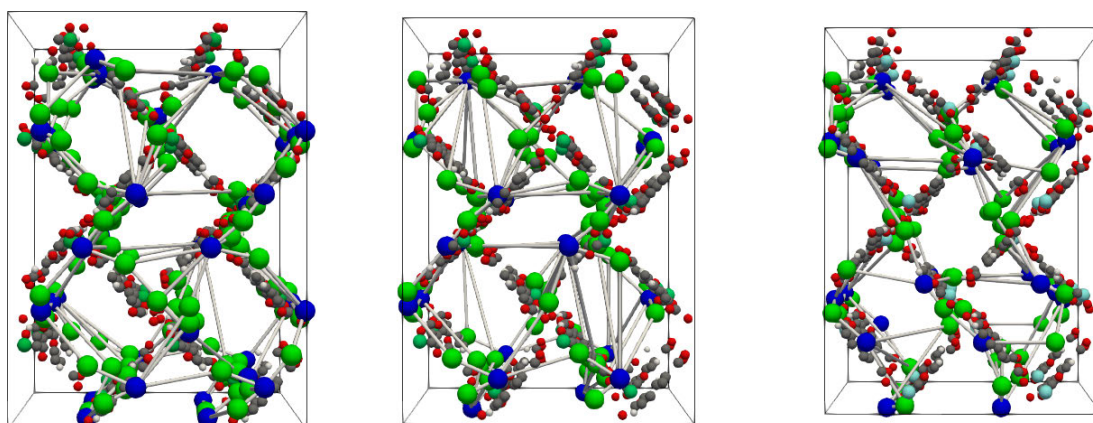
We introduced two comparison measures PDEG and GWEG to compare extremum graphs to facilitate comparative analysis and visualization of scalar fields. We discuss theoretical properties and we demonstrate their utility in periodicity detection and similarity detection in pore networks.



(a) Supercells of AFUPEX, SEHSUU, SEHTEF



(b) Distance fields constructed for AFUPEX, SEHSUU, SEHTEF



(c) Minimum Graphs constructed for AFUPEX, SEHSUU, SEHTEF

Figure 6.9: We construct the supercells followed by the distance fields for three materials AFUPEX, SEHSUU, SEHTEF followed by minimum graph construction and comparison by PDEG and GWEG

Chapter 7

Time Varying Extremum Graphs (TVEG)

In this chapter, we present the time-varying extremum graph (TVEG), an extension of extremum graphs to facilitate visual analysis of time-varying scalar fields [31]. We discuss the construction and showcase its utility in various experiments.

7.1 Introduction

TVEG captures temporal events like creation/destruction, merge/split of topological features that are represented within the extremum graph of the scalar field at individual time steps. Figure 7.1 shows a time-varying 2D scalar field together with the extremum graph of the field at each time step. The temporal arcs of the TVEG correspond to split, merge, and continuation of a collection of topological features.

7.2 Contributions

We introduce the time-varying extremum graph (TVEG), a topological structure for representing the combinatorial structure of the Morse decomposition of a scalar field and its evolution over time. We discuss its applications to feature exploration and tracking. While tracking is an important application, TVEG is not limited to tracking, it is a time-varying data structure with potential for expansion. Key contributions include

- A definition of a novel topological structure, the time-varying extremum graph (TVEG).
- An algorithm for constructing the TVEG based on a formulation as an optimization problem.
- Application to the study of interesting features and topological events in synthetic and simulation datasets that demonstrate the utility of the topological structure.

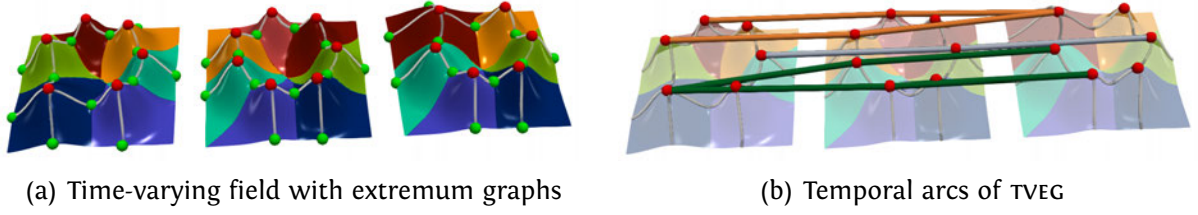


Figure 7.1: TVEG of a 2D time-varying scalar field. (a) Three time steps f_1, f_2, f_3 of a time-varying scalar field together with the respective extremum graphs G^1, G^2, G^3 embedded on the domain. Each peak is associated with a unique segment. Maxima are shown in red, saddles in green. (b) A subset of the temporal arcs of the TVEG. Green arcs correspond to a feature in G^1 that splits into two in G^2 and continues in G^3 . The orange arcs correspond to two features in G^1 that continue onto G^2 but merge in G^3 . The grey arcs correspond to a continuation of a feature from G^1 onto G^3 . The remaining (grey) arcs that correspond to continuation of features and the saddle critical points are not shown for clarity.

7.3 Time varying extremum graphs TVEG

We describe a design of TVEG for representing the temporal dynamics of a time-varying scalar field \mathcal{F} . The design aims to capture the temporal dynamics of both individual topological features and of a cluster of features that lie within a spatial neighborhood. We require that temporal slices of the TVEG at all sampled time steps of \mathcal{F} result in an extremum graph. The node set of the TVEG consists of maxima and $(n - 1)$ -saddles of \mathcal{F} at every time step. An arc in the TVEG either belongs to an extremum graph of \mathcal{F} restricted to a given time step or is a temporal arc that connects nodes between two consecutive time steps.

Let $\mathcal{G}^t(V^t, E^t)$ denote the extremum graph of \mathcal{F} at a given time step $t, 1 \leq t \leq T$. The TVEG $\mathcal{G}^*(V^*, E^*)$ is a graph that contains all \mathcal{G}^t as subgraphs in addition to arcs between nodes from consecutive time steps. Thus, the node set V^* can be expressed as $V^* = \bigcup_{t=1}^T V^t = \{\bigcup_{t=1}^T M^t, \bigcup_{t=1}^T S^t\}$. We further distinguish between the maxima and $(n - 1)$ -saddles of V^t , and collect them into sets M^t and S^t , respectively. The arc set E^* can be expressed as $E^* = \{\bigcup_{t=1}^T E^t, \bigcup_{t=1}^{T-1} A^t\}$, consisting of two types of arcs namely the extremum graph arcs E^t and the temporal arcs A^t . An arc in E^t connects a maximum and a $(n - 1)$ -saddle. A temporal arc in A^t represents the correspondence between two maxima that lie in extremum graphs of two consecutive time steps. In other words, an element $e \in A^t$ is a temporal arc such that $e = (m_i^t, m_j^{t+1})$ representing correspondence between maximum $m_i^t \in M^t$ and $m_j^{t+1} \in M^{t+1}$ belonging to extremum graphs at time steps t and $t + 1$, respectively.

Figure 7.2 shows a subgraph of a TVEG consisting of temporal arcs between time steps t

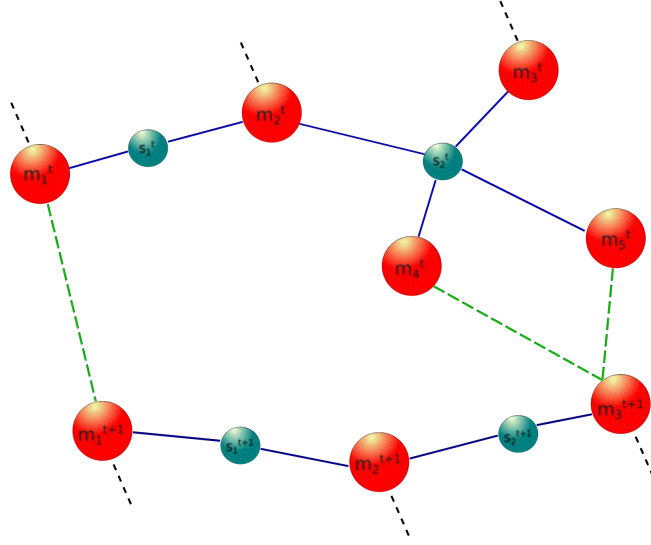


Figure 7.2: An illustration of a TVEG showing two time steps t and $t + 1$. Sets $\{m_1^t, \dots, m_5^t\}$ and $\{s_1^t, s_2^t\}$ consists of maxima and $(n - 1)$ -saddles from the extremum graph at time step t . Similarly, sets $\{m_1^{t+1}, m_2^{t+1}, m_3^{t+1}\}$ and $\{s_1^{t+1}, s_2^{t+1}\}$ are maxima and saddles of the extremum graph at time step $t + 1$. Arcs of the extremum graph are in blue, temporal arcs are shown as green dashed edges, and connections to nodes outside the figure are shown as black dashed edges. We observe that maxima m_2^t and m_3^t do not have a suitable temporal correspondence and hence die at time step t whereas m_2^{t+1} is born. Maxima m_4^t and m_5^t merge into m_3^{t+1} .

and $t + 1$. Note that A^t does not include temporal arcs between saddles. Maxima exhibit relatively higher stability over time in terms of spatial movement when compared to saddles. So, we chose to restrict temporal correspondence to those between maxima. The arcs set A^t of \mathcal{G}^* can be effectively used to represent the temporal variation in \mathcal{F} together with topological events such as split, merge, generation, and deletion of features. Figure 7.1 illustrates how the TVEG captures the different events in a 2D time-varying scalar field. Arcs in E^t (light-gray) connect maxima with saddles within a time step. Arcs in A^t (green, orange, gray) highlight temporal arcs that capture different topological events.

7.3.1 Temporal correspondences

A temporal arc in A^t represents the correspondence between a pair of maxima from two consecutive time steps. We now elaborate upon an optimization criterion that determines the temporal arcs.

Equation 7.2 describes two correspondences for a maximum m_i^t with two maxima $m_{j_1}^{t+1}$ and $m_{j_2}^{t+1}$ from time step $t + 1$. The scores associated with the two correspondences are expressed as the smallest and second smallest value of an objective function over all maxima

$m_j^{t+1} \in M^{t+1}$ subject to some structural constraints (Equation 7.3). We define the arc set A^t as the maximum cardinality set of arcs between maxima m_i^t in time step t and their correspondences in time step $t + 1$:

$$\begin{aligned} & \max |A^t|, \text{ where} \\ & A^t \subseteq \bigcup_{m_i^t \in M^t} \{(m_i^t, m_{j1}^{t+1}), (m_i^t, m_{j2}^{t+1})\} \end{aligned} \quad (7.1)$$

$$m_{j1}^{t+1} = \underset{m_j^{t+1} \in M^{t+1}}{\operatorname{argmin}} \mathcal{P}(m_i^t, m_j^{t+1}) + \mathcal{J}(m_i^t, m_j^{t+1}) + \mathcal{D}(m_i^t, m_j^{t+1}) + \mathcal{N}(m_i^t, m_j^{t+1}) \quad (7.2)$$

$$m_{j2}^{t+1} = \underset{m_j^{t+1} \in M^{t+1} \setminus m_{j1}^{t+1}}{\operatorname{argmin}} \mathcal{P}(m_i^t, m_j^{t+1}) + \mathcal{J}(m_i^t, m_j^{t+1}) + \mathcal{D}(m_i^t, m_j^{t+1}) + \mathcal{N}(m_i^t, m_j^{t+1})$$

$$\begin{aligned} & \text{subject to the constraint that} \\ & (m_k^t, m_{j1}^{t+1}) \notin A^t \text{ and } (m_k^t, m_{j2}^{t+1}) \notin A^t \text{ for all } m_k^t \in M^t \setminus m_i^t \end{aligned} \quad (7.3)$$

We allow for two correspondences between m_i^t and maxima in time step $t + 1$, thereby supporting the representation of split events. The framework is designed to support any number of correspondences per maximum if desired. However, a larger number of correspondences results in increased complexity of the extremum graph and the resulting TVEG tracks, which might be difficult to visualize. Further, the extremum graphs within each time step are simplified prior to TVEG construction, which removes critical point pairs that are functionally similar and hence reduces the number of potential correspondences.

The objective function for the scores consists of four components – functional persistence of the maxima, function value at maxima, spatial location of the maxima, and neighborhood of the maxima within the respective extremum graphs \mathcal{G}^t and \mathcal{G}^{t+1} . The persistence component $\mathcal{P}(m_i^t, m_j^{t+1})$ is equal to the absolute difference between the topological persistence of m_i^t and m_j^{t+1} . The functional component $\mathcal{J}(m_i^t, m_j^{t+1}) = |f(m_i^t) - f(m_j^{t+1})|$ measures the absolute difference between function values at the maxima. The distance component $\mathcal{D}(m_i^t, m_j^{t+1})$ is equal to the Euclidean distance between the maxima. The neighborhood component $\mathcal{N}(m_i^t, m_j^{t+1}) = |\eta(m_i^t) - \eta(m_j^{t+1})|$, where $\eta(m)$ for a maximum m is defined as $\eta(m) = \sum_{i=1}^{|N(m)|} |f(m) - f(i)|$ i.e., the sum of absolute difference of function values between the maximum m and all saddles in its neighborhood $N(m)$. While the first three compo-

nents depend exclusively on the maxima and the topological feature that they represent, the component \mathcal{N} captures the differences between the local connectivity of the maxima in consecutive steps. Hence, this component is crucially dependent on the extremum graph. Topological persistence is a good stable measure of the topological feature represented by a maximum-saddle pair, and hence included in the objective function. The functional and spatial components are local properties of the maxima that incorporates finer grained analysis of similarity and correspondence between consecutive time steps. Finally, the fourth component helps capture topological similarity in terms of the neighborhood of the maxima within the respective extremum graphs.

In order to reduce the effect of noise on the score, low persistent maxima are canceled in a first step based on a user specified persistence threshold and the objective function is computed post-simplification [42, 44]. Critical point pair cancellations are performed independently within the two time steps. Finally, the structural constraints ensure that no arc participates in both a merge and split event by disallowing 'z'-shaped configurations.

7.3.2 Algorithm

We compute TVEG in two steps. The first step constructs the extremum graphs for all the time steps of the time-varying scalar field \mathcal{F} by either employing the approach of Correa et al. [30] or computing the MS complex [110, 32, 12] and extracting the substructure. The Topology toolkit (TTK) by Tierny et.al. [127] provides an alternate method to compute MS complex based on Discrete Morse Theory [49]. In either case, the graph is simplified using a persistence threshold ε to remove noise. Corresponding to each node of the extremum graph, we store a tuple that contains attributes related to the critical point (see Table 7.1). An arc connecting a pair of critical points v^1, v^2 is stored in E^t as $(v^1.id, v^2.id)$.

The second step computes the correspondences between maxima from consecutive time steps. We now describe the algorithm TEMPORALARCS (Algorithm 3) that computes and records temporal arcs of the TVEG for all pairs of consecutive time steps over a given range $1 \leq p < r \leq T$. Apart from computing temporal arcs, it also detects and records topological events such as merge, split, deletion, and generation in the sets \mathcal{E}^{m*} , \mathcal{E}^{s*} , \mathcal{E}^{d*} and \mathcal{E}^{g*} , respectively. TEMPORALARCS begins by initializing the set A^{t*} that stores all temporal arcs, sets M^0 and M^1 that store the collection of maxima in the two time steps, and sets for recording topological events. Iterating over pairs of consecutive time steps (lines 4-24), the algorithm computes the correspondence scores for all maxima using COMPUTESCORES (Algorithm 5). The scores are recorded in \mathcal{S} , and further refined using FILTERSCORES (Algorithm 6) based on a threshold derived from the variance of the scores. The score computation and refinement

Algorithm 3: TEMPORALARCS

Input : A set of extremum graphs $[\mathcal{G}^p, \dots, \mathcal{G}^r]$

Output: Temporal arc set A^{t*}

Topological event sets $\mathcal{E}^{m*}, \mathcal{E}^{s*}, \mathcal{E}^{d*}$, and \mathcal{E}^{g*}

```
1 Initialization:  $A^{t*} \leftarrow \emptyset; A^t \leftarrow \emptyset; M^0 \leftarrow \emptyset; M^1 \leftarrow \emptyset;$ 
  /* Initialize  $M^0$  as maxima set of  $\mathcal{G}^p$  */
2  $M^0 \leftarrow M^p$ 
  /* Initialize all topological event sets */
3  $\{\mathcal{E}^{m*}, \mathcal{E}^{s*}, \mathcal{E}^{d*}, \mathcal{E}^{g*}\} \leftarrow \emptyset$ 
4 for  $i \leftarrow p + 1$  to  $r$  do
  /* Initialize  $M^1$  as maxima set of  $\mathcal{G}^i$  */
5    $M^1 \leftarrow M^i$ 
6    $\mathcal{S} \leftarrow \text{COMPUTESCORES}(M^0, M^1)$ 
7    $\mathcal{S} \leftarrow \text{FILTERSCORES}(\mathcal{S})$ 
  /* Compute the temporal arc set  $A^t$  */
8   foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
9      $A^t \leftarrow A^t \cup (m^0, m^1)$ 
10  end
  /* Detect topological events */
11   $\mathcal{E}^m \leftarrow \text{DETECTMERGE}(\mathcal{S}, i)$ 
12   $\mathcal{E}^s \leftarrow \text{DETECTSPLIT}(\mathcal{S}, i)$ 
13   $\mathcal{E}^d \leftarrow \text{DETECTDEL}(\mathcal{S}, M^0, i)$ 
14   $\mathcal{E}^g \leftarrow \text{DETECTGEN}(\mathcal{S}, M^1, i)$ 
  /* Remove z-shape configurations. */
15   $\mathcal{W} \leftarrow \mathcal{E}^m \cap \mathcal{E}^s$ 
16  repeat
17     $w \leftarrow \text{MAXSCOREEDGE}(\mathcal{W})$ 
18     $A^t \leftarrow A^t \setminus w$ 
19     $\mathcal{E}^m, \mathcal{E}^s \leftarrow \text{UPDATEMERGESPLITEDGES}(\mathcal{E}^m, \mathcal{E}^s, w)$ 
20     $\mathcal{W} \leftarrow \mathcal{E}^m \cap \mathcal{E}^s$ 
21  until  $\mathcal{W} = \emptyset$ 
  /* Populate temporal arc set */
22   $A^{t*} \leftarrow A^{t*} \cup A^t$ 
  /* Update topological event sets */
23   $\mathcal{E}^{m*} \leftarrow \mathcal{E}^{m*} \cup \mathcal{E}^m$ 
24   $\mathcal{E}^{s*} \leftarrow \mathcal{E}^{s*} \cup \mathcal{E}^s$ 
25   $\mathcal{E}^{d*} \leftarrow \mathcal{E}^{d*} \cup \mathcal{E}^d$ 
26   $\mathcal{E}^{g*} \leftarrow \mathcal{E}^{g*} \cup \mathcal{E}^g$ 
  /* Re-initialize for next iteration */
27   $A^t \leftarrow \emptyset, M^0 \leftarrow M^1$ 
28 end
```

Table 7.1: List of different attributes of the critical points.

Fields	Description
id	A unique id assigned to the critical point
$index$	Index
\bar{x}	Coordinates
$pers$	Topological persistence
$ascmfold$	Ascending manifold
$dscmfold$	Descending manifold
$geom$	Ascending / Descending manifold geometry
t	The time stamp of the scalar field

is described later in this section. After refinement, \mathcal{S} records the set of correspondences and scores and set A^t is updated with the temporal arcs.

Next, the various topological events are recorded in sets $\mathcal{E}^m, \mathcal{E}^s, \mathcal{E}^d$, and \mathcal{E}^g (lines 11-14). We then resolve any co-occurrences of merge and split or ‘z’-shaped configurations between two time steps (lines 15-21). These co-occurrences are recognized as a subset of the intersection of \mathcal{E}^m and \mathcal{E}^s . This subset is computed using a greedy approach that iteratively removes the largest score edge in a ‘z’-configuration from A^t . Finally, the set A^{t*} containing temporal correspondences over all iterations, the sets containing topological events per iteration, and the global containers for arcs and events are updated (lines 22-26). TEMPORALARCS computes the global temporal arc set of a TVEG in quadratic time $\mathcal{O}(n^2)$, where $n = \max |M^t|, p \leq t \leq r$, the largest cardinality of the maxima sets over all time steps in the given input time range $[p, r]$.

Score computation. The subroutine COMPUTESCORES (Algorithm 5) uses Equation 7.2 to compute correspondence scores between two sets of maxima M^0 and M^1 . For each maximum $m^0 \in M^0$, it iterates exhaustively over all maxima in M^1 , computes the objective function from equation 7.2 using the attributes $(pers, \bar{x}, \mathcal{F}(\bar{x}))$, and records them. The distance component is computed as the Euclidean distance between pair of maxima. The neighborhood component is computed by considering the local neighborhood contribution η of the maxima. The two lowest scores are identified for m^0 and finally the output \mathcal{S} containing the correspondences and their scores is returned as a set of tuples (m^0, m^1, s) of length three. In practice, to ensure that the addition of the four components in Equation 7.2 is meaningful, we normalize them so that the values lie within $[0, 1]$.

Score refinement. The set of scores returned by COMPUTESCORES (Algorithm 5) need to be further refined to assess whether a particular correspondence is meaningful and should be retained within \mathcal{S} . The refinement may lead to topological events such as deletion and

Algorithm 4: UPDATEMERGESPLITEDGES Update the set of edges participating in merge/split events

Input : Edge sets $\mathcal{E}^m, \mathcal{E}^s$, maximum score edge w

Output: Modified Edge sets

```

/* Remove edge adjacent to  $w$  in  $\mathcal{E}^s$  and  $\mathcal{E}^m$  */
1  $u \leftarrow$  edge that participates in split with  $w$ 
2  $\mathcal{E}^s \leftarrow \mathcal{E}^s \setminus \{w, u\}$ 
3  $U \leftarrow$  set of edges that participate in merge with  $w$ 
4 if  $U == \{u\}$  then
5   |  $\mathcal{E}^m \leftarrow \mathcal{E}^m \setminus u$ 
6  $\mathcal{E}^m \leftarrow \mathcal{E}^m \setminus w$ 
7 return  $\mathcal{E}^m, \mathcal{E}^s$ 

```

generation. FILTERSCORES (Algorithm 6) processes the input set \mathcal{S} and refine the set based on a threshold. The procedure records all scores in a list \mathcal{Y} , computes the mean(μ) and the standard deviation(σ) of scores, and sets the threshold τ to $\mu + \sigma$. Finally, all tuples corresponding to scores greater than or equal to τ are removed from \mathcal{S} .

7.4 Applications

We demonstrate the utility of TVEG on a synthetic sum of 3D Gaussians dataset consisting of spatio-temporal movement of the centres of eight 3D Gaussians, a viscous fingers dataset [108] that simulates the temporal mixing of salt and water resulting in finger like spatial structures, and a 3D Bénard-von Kármán vortex street dataset [101, 51]. The above-mentioned datasets are 3D time-varying scalar fields, so we use PYMS3D [110] to compute the MS complex and extract the extremum graph as a substructure for all time steps. It is likely that multiple saddles are adjacent to a given pair of maxima. In all experiments, we retain the saddle with the highest function value to record the adjacency between the segments associated with the maxima, and discard the other saddles to reduce clutter. The TVEG is computed using the method described in Section 7.3.2.

7.4.1 Moving Gaussians

The first case study consists of experiments on a synthetic dataset called *Gauss8* defined on a $128 \times 128 \times 128$ grid over 50 time steps. It represents the movement of eight 3D Gaussians whose centres move along predetermined paths while being restricted to the $x = 0$ plane. The dataset is defined as the sum of eight 3D Gaussians, whose maxima are located exactly at the Gaussian centres. The trajectories of the centres are designed to

Algorithm 5: COMPUTESCORES Compute all correspondences and scores for a given set of maxima

Input : Two maxima sets M^0, M^1

Output: A set of optimal scores $\mathcal{S} = \{(u, v, s)\}$ s.t. $(u, v) \in M^0 \times M^1$ and $s \in \mathbb{R}$

```

1 Initialization:  $\mathcal{S} \leftarrow \emptyset$ 
2 foreach  $m^0 \in M^0$  do
3    $Q \leftarrow \emptyset$ 
4   foreach  $m^1 \in M^1$  do
5     /* Compute score for  $(m^0, m^1)$  */
6      $s \leftarrow |m^0.\text{pers} - m^1.\text{pers}| + |\mathcal{F}(m^0.\bar{x}) - \mathcal{F}(m^1.\bar{x})| + |m^0.\bar{x} - m^1.\bar{x}|_2$ 
7      $Q \leftarrow Q \cup s$ 
8   end
9   /* Insert two lowest scores to  $\mathcal{S}$  */
10   $s_1 \leftarrow \min(Q), \mathcal{S} \leftarrow \mathcal{S} \cup (m^0, m^1, s_1)$ 
11   $Q \leftarrow Q \setminus s_1$ 
12   $s_2 \leftarrow \min(Q), \mathcal{S} \leftarrow \mathcal{S} \cup (m^0, m^1, s_2)$ 
13 end
14 return  $\mathcal{S}$ 

```

Algorithm 6: FILTERSCORES Filter the input set of scores based on a threshold

Input : A list of scores \mathcal{S} from Algorithm 5

Output: A filtered version of \mathcal{S}

```

1 Initialization:  $\mathcal{Y} \leftarrow \emptyset$ 
2 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
3    $\mathcal{Y} \leftarrow \mathcal{Y} \cup s$ 
4 end
5 /* Compute the mean and standard deviation of all scores */
6  $\mu \leftarrow \text{MEAN}(\mathcal{Y}), \sigma \leftarrow \text{STD}(\mathcal{Y})$ 
7 /* Refine  $\mathcal{S}$  using the threshold  $\tau$  */
8  $\tau \leftarrow \mu + \sigma$ 
9 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
10  if  $s \geq \tau$  then
11     $\mathcal{S} \leftarrow \mathcal{S} \setminus (m^0, m^1, s)$ 
12 end
13 return  $\mathcal{S}$ 

```

Algorithm 7: DETECTMERGE Detect merge events between two consecutive time steps

Input : A list of scores \mathcal{S} from Algorithm 5;
Time step t
Output: Set of merges \mathcal{E}^m in \mathcal{S} between time t and $t + 1$

```

1 Initialization:  $\mathcal{E}^m \leftarrow \emptyset$ 
2 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
3    $e \leftarrow m^1$ 
4   /* Count correspondences mapped to  $e$  */
5    $c \leftarrow 0; \mathcal{K} \leftarrow \emptyset$ 
6   foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
7     if  $m^1 == e$  then
8        $\mathcal{K} \leftarrow \mathcal{K} \cup (m^0, t); c \leftarrow c + 1$ 
9     end
10    /* Merge detected. Update  $\mathcal{E}^m$  */
11    if  $c > 1$  then
12       $\mathcal{E}^m \leftarrow \mathcal{E}^m \cup (e, t + 1) \cup \mathcal{K}$ 
13       $c \leftarrow 0; \mathcal{K} \leftarrow \emptyset$ 
14    end
15 return  $\mathcal{E}^m$ 

```

Algorithm 8: DETECTSPLIT Detect split events between two consecutive time steps

Input : A list of scores \mathcal{S} from Algorithm 5;
Time step t
Output: Set of splits \mathcal{E}^s in \mathcal{S} between time t and $t + 1$

```

1 Initialization:  $\mathcal{E}^s \leftarrow \emptyset$ 
2 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
3    $e \leftarrow m^0$ 
4   /* Count correspondences mapped from  $e$  */
5    $c \leftarrow 0; \mathcal{K} \leftarrow \emptyset$ 
6   foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
7     if  $m^0 == e$  then
8        $\mathcal{K} \leftarrow \mathcal{K} \cup (m^1, t + 1); c \leftarrow c + 1$ 
9     end
10    /* Split detected. Update  $\mathcal{E}^s$  */
11    if  $c > 1$  then
12       $\mathcal{E}^s \leftarrow \mathcal{E}^s \cup (e, t) \cup \mathcal{K}$ 
13       $c \leftarrow 0; \mathcal{K} \leftarrow \emptyset$ 
14    end
15 return  $\mathcal{E}^s$ 

```

Algorithm 9: DETECTDEL Detect deletion events between two consecutive time steps

Input : A list of scores \mathcal{S} from Algorithm 5;
A list of maxima M^t for time step t ;
Time step t

Output: Set \mathcal{E}^d with all the deletion events detected from \mathcal{S} between time t and $t + 1$

```
1 Initialization:  $\mathcal{E}^d \leftarrow \emptyset; \mathcal{K} \leftarrow M^t$ 
  /* Record maxima with no edges to  $t + 1$  */
2 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
3   |  $\mathcal{K} \leftarrow \mathcal{K} \setminus m^0$ 
4 end
  /* Add time step information */
5 foreach  $e \in \mathcal{K}$  do
6   |  $\mathcal{E}^d \leftarrow \mathcal{E}^d \cup (e, t)$ 
7 end
8 return  $\mathcal{E}^d$ 
```

Algorithm 10: DETECTGEN Detect generation events between two consecutive time steps

Input : A list of scores \mathcal{S} from Algorithm 5;
A list of maxima M^{t+1} for time step $t + 1$;
Time step t

Output: Set \mathcal{E}^g with all the generation events detected from \mathcal{S} between time t and $t + 1$

```
1 Initialization:  $\mathcal{E}^g \leftarrow \emptyset; \mathcal{K} \leftarrow M^{t+1}$ 
  /* Record maxima with no edges to  $t + 1$  */
2 foreach  $(m^0, m^1, s) \in \mathcal{S}$  do
3   |  $\mathcal{K} \leftarrow \mathcal{K} \setminus m^1$ 
4 end
  /* Add time step information */
5 foreach  $e \in \mathcal{K}$  do
6   |  $\mathcal{E}^g \leftarrow \mathcal{E}^g \cup (e, t + 1)$ 
7 end
8 return  $\mathcal{E}^g$ 
```

Algorithm 11: EXGRAPH3D Compute extremum graph

Input : A scalar field $\mathcal{F}(t)$ at time step t
A persistence threshold ϵ
Output: Extremum Graph $\mathcal{G}^t = (V^t, E^t)$

```
1 Initialization:  $V^t \leftarrow \emptyset; E^t \leftarrow \emptyset$ 
2 Compute MS complex:  $\mathcal{C} \leftarrow \text{MS3D}(\mathcal{F}(t), \epsilon)$ 
  /* Store neighborhood maxima and saddles */
3 foreach  $c \in \mathcal{C}$  do
4   if  $c.\text{index} == 2$  then
5      $M \leftarrow c.\text{ascmfold}$ 
6     foreach  $m \in M$  do
7        $m^t \leftarrow m, V^t \leftarrow V^t \cup m^t$ 
8        $E^t \leftarrow E^t \cup (c.\text{id}, m.\text{id})$ 
9     end
10     $c^t \leftarrow c, V^t \leftarrow V^t \cup c^t$ 
11 end
12  $V^t \leftarrow \text{set}(V^t)$ 
13 return  $\mathcal{G}^t = (V^t, E^t)$ 
```

induce multiple topological events such as merges and splits. Figure 7.3 shows three time steps that exhibit splits and merges together with a change in the numbers and locations of maxima. We demonstrate the utility of TVEG tracks by emphasizing upon two aspects of *Gauss8* as observed from the tracks, see Figure 7.4. The temporal tracks in the figure are a collective representation of the temporal arcs from TVEG and represents correspondence between maxima along time. The 3D domain of the scalar field is scaled along the z-axis, multiple instances of the domain are stacked to represent the space-time domain. Figure 7.4 shows one such scaled domain in brown together with the extremum graph at that time step. All maxima (red) in the extremum graph lie on a plane as expected and continue to be restricted to the plane over time, as is evident from the flat appearance of the temporal TVEG tracks outlined by the cyan box. We study two tasks that rely on the TVEG tracks.

Topological event detection. The merges and splits among the Gaussians can be visually identified from the TVEG tracks in Figure 7.4. From the overview of the temporal tracks (Figure 7.4, top-right) computed over the entire time range of 50 steps, we observe multiple merges followed by splits. We note an increasing number of cross correspondences as the tracks approach a merge event or subsequent to a split event. The existence of such cross temporal arcs between corresponding maxima over a time range can indicate gradually decreasing spatial distance between global components such as super-level sets eventually leading to topological merges/splits. A specific track segment of interest, **A**, is highlighted.

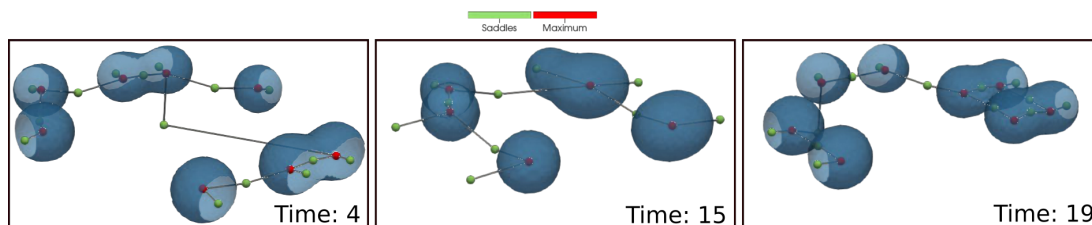


Figure 7.3: Dynamics in a synthetic sum of Gaussians dataset *Gauss8*. The data is visualized by displaying the intersection of descending 3-manifolds of maxima with volume enclosed by the isosurface at scalar value 21. The resulting blobs merge over time to form larger components and subsequently split into multiple components. The merge and split behavior is also observable from the extremum graphs.

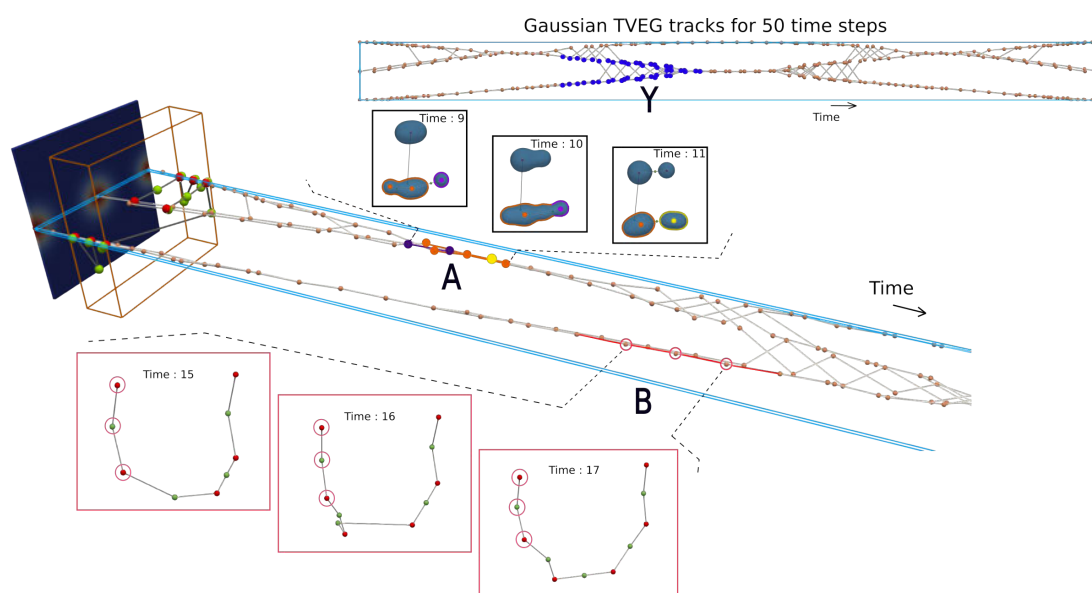


Figure 7.4: Temporal tracks from the TVEG of *Gauss8*. (top-right) A view from top of the stacked domains shows the tracks for all time steps. A subset of TVEG tracks (Y) that exhibits symmetry along time is highlighted in blue. (middle) Track A is a subset of a longer track, consisting of time steps 9-11 and includes merge and split events. Inset depicts the blobs in the corresponding time steps 9, 10, and 11. Track B (red), a subset of Y, is selected to showcase the structural similarity between extremum graphs (inset) sampled at time steps 15, 16, and 17.

The maxima merge, die, and are born at time steps 9,10,11. We choose an isosurface extracted at scalar value 21 to visualize the field. Each component of the isosurface associated with a maximum is identified by computing the intersection with the descending 3-manifold of the maximum. The blobs merge and subsequently split corresponding to the topological events in the extremum graph. Along track **A**, the maxima are colored to indicate their temporal correspondence. The corresponding maxima in the scalar domain and their associated isosurface components are also highlighted with the same color. The purple maximum continues between time steps 9 and 10, whereas the orange maxima in time step 9 merge in time step 10. As a result of this interaction one component of the isosurface in time step 10 is jointly represented by the purple and orange maxima. The purple maximum vanishes between time steps 10 and 11 and a new yellow maximum appears, resulting in one of the isosurface components splitting into two. The TVEG temporal tracks help interpret the topological events similar to a tracking graph.

Similarity pattern within extremum graphs. The tracks of the Gaussian centres in *Gauss8* are designed to be symmetrical as can be seen in the overview of the tracks over 50 time steps (Figure 7.4, top-right). One such temporally symmetric region is labelled as **Y** (blue) within the overview. We studied structural changes within the extremum graphs from the time range when the TVEG tracks exhibit symmetry. One such segment, track **B** (red), is highlighted to demonstrate the result of the study and three time steps (15, 16, and 17) from this segment are shown. Maxima that belong to track **B** are highlighted within the extremum graph using a red enclosing circle. One neighboring saddle and a maximum in the neighborhood of the saddle are also highlighted using a red circle. We observe a structural similarity between the highlighted regions of the extremum graphs in these time steps. This observation suggests that temporal geometric patterns within TVEG tracks, if any, are indicative of a repetitive local structural pattern within the corresponding extremum graphs over time.

Comparison with the Lifted Wasserstein Matcher (LWM) [116] implemented in TTK [128]. We use the *Gauss8* dataset in this comparison. We first generate tracks using TVEG and then we use LWM provided by the TTK to compare our results as shown in Figure 7.5. For the LWM tracks, we use the same parameters as specified by the authors - extrema weight = 1.0, saddle weight 0.1, x,y,z weights = 1.0,1.0,1.0. To facilitate comparison we embed the tracks in the domain like it is done in TTK. We note that while LWM is provided as a ttkfilter with an easy interface, the current implementation does not provide an option to do the post processing. So in case of LWM without post processing, split/merge events are identified as birth/death events. TVEG handles the merge/split events directly.

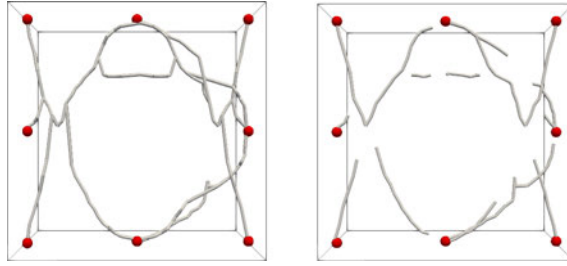


Figure 7.5: Comparison of tracks obtained by TVEG (left) and LWM (right) for *Gauss8* dataset. All maxima in the first timestep are highlighted. The apparent breaks in LWM tracks are due to a missing post-processing step in the current implementation.

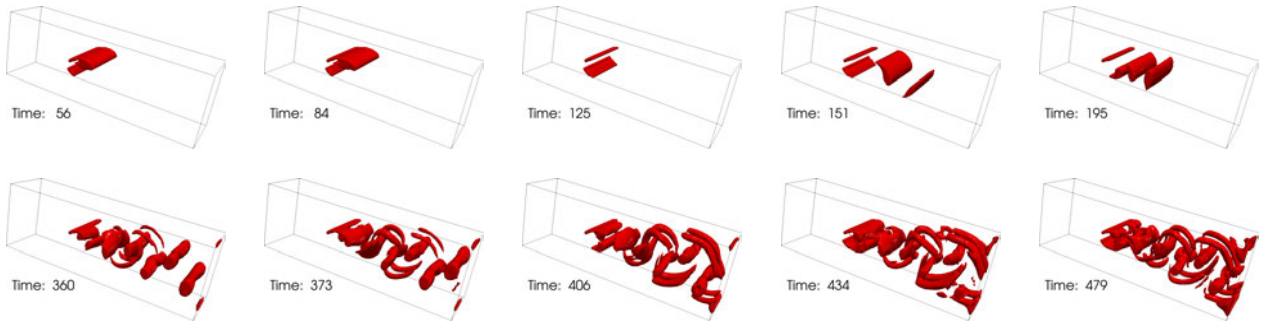


Figure 7.6: Visualizing the top 20% tracks in the 3D von Kármán vortex street data. The tracks are generated based on TVEG and spatial overlaps, and the regions associated with maxima in the tracks are displayed. The top tracks include the temporal evolution of many of the primary and secondary vortices.

7.4.2 Vortex Street

We demonstrate the utility of TVEG in feature tracking by presenting two use cases – computing a summary view by tracking features that are automatically computed, and tracking features that are specified as a user query. We demonstrate both use cases on a 3D Bénard-von Kármán vortex street dataset. The Okubo-Weiss criterion, which indicates regions of high vorticity, is a scalar field sampled on a regular grid [101]. It is available on a $192 \times 64 \times 48$ grid over 508 time steps. Previous results [101, 117] identify two classes of vortices, primary

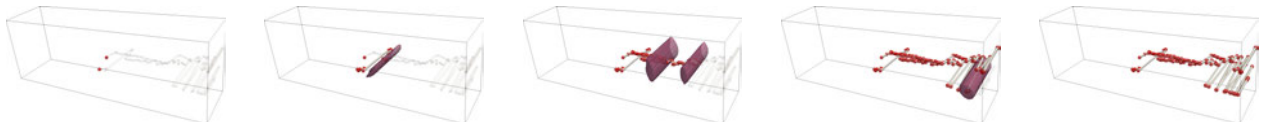


Figure 7.7: Visualizing tracks computed in response to a query consisting of two maxima (left). The spatial location of the maxima varies substantially even though they represent the same feature over time. (left to right) The tracks are shown using high transparency for earlier time steps. As the maxima and the corresponding features evolve over time, the opacity of the track is increased and the maxima are also displayed.

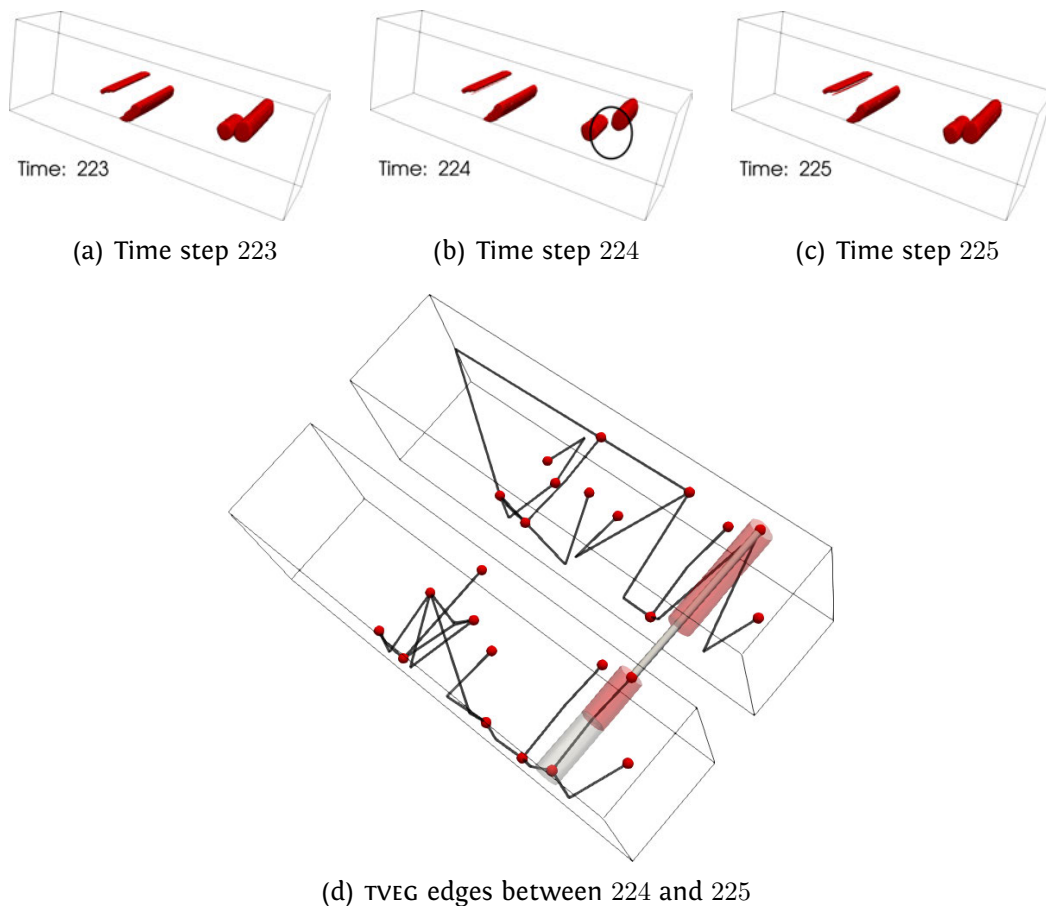


Figure 7.8: Illustrating challenges in tracking vortices in the 3D von Kármán vortex street data. (a)-(c) Vortices in time steps 223-225. (d) Temporal arcs between a particular feature in time 224 (left) and time 225 (right). The extremum graphs within each time step are displayed with black arcs to provide context. The white feature in time 224 should have ideally been mapped to the red feature in time 225. Instead, there is no temporal arc connecting it to any feature in 225. The red feature in 224 is connected with the red feature in 225 via the temporal arc shown in gray with higher thickness.

and secondary, and track them across time.

Overview of features. In order to identify and track all topological features that can be captured using an extremum graph, we first compute the TVEG and consider all temporal arcs. Among the multiple possible correspondences between a maximum in time step t and a maximum in time step $t + 1$, TVEG provides the top two correspondences based on a score. Each maximum has an associated region, which is spanned by its descending manifold clipped by the isosurface at scalar value 0.1. The threshold value of 0.1 represents high vorticity regions. For the two correspondences, which are stored as temporal arcs, we calculate the spatial overlaps between the regions associated with the maxima. We pick the arc with the higher spatial overlap between the regions. Arcs where the spatial overlap is small and short length tracks (< 10 time steps) are removed. We observe that each secondary vortex is represented by a single maximum, but a primary vortex may comprise of regions associated with a collection of maxima. In the latter case, the multiple maxima spanning a primary vortex are connected in the extremum graph via saddles. So, we collate such tracks in a post-processing step. We observe that the top 20% tracks sorted in decreasing order of track length consists of most primary and secondary vortices, see Figure 7.6. The regions associated with maxima in the tracks are displayed at different time steps to visualize the tracks. We observe that none of the temporal arcs connect a primary vortex with a secondary and hence note that the TVEG is able to provide a good summary of the two types of vortices.

Query driven feature tracking. User queries consisting a set of maxima from primary or secondary vortices or both may be used to visualize specific tracks. Figure 7.7 shows one such query. Again, we observe that the temporal arcs either connect two primary vortices or two secondary vortices. One challenge that is typically encountered while analyzing this vortex street data is that, even though the spatial movement of vortices is along a smooth curve, the maxima that represents the vortices follow a tortuous path. The temporal arcs in the TVEG play a crucial role in finding the correspondences between such maxima. It provided a foundation upon which the additional spatial overlap criterion could be applied.

Discussion. Overall, we find that the TVEG serves as a good underlying representation that supports the development of feature tracking methods. The four components of the objective function are sufficient to identify meaningful correspondences in most cases, but there are exceptions.

The simplification threshold which is uniformly applied across all time steps affects the extremum graph and therefore the segmentation. While a maximum may continue to represent the same vortex across different time steps, its spatial location within the vortex

may vary substantially as seen in Figure 5.22(a). These two factors may lead to instances where the top two temporal arcs from a maximum in time t to maxima in time $t + 1$ may not represent the desired feature correspondences. Further, filtering the correspondences in a refinement step may result in zero temporal arcs incident on a maximum and hence discontinuation of tracks.

We show one such case occurring in time steps 223-225 in Figure 7.8. A feature represented by a maximum in 223 splits, resulting in a primary vortex represented by two maxima. One of those maxima, which represents a partial vortex in time 224, contains temporal arcs to features in time 225, but there is no arc for one feature (see Figure 7.8(d)). As a consequence, the track containing the white feature in time 224 terminates in time 225, thereby causing the anomaly highlighted in Figure 7.8(b). Such early terminations lead to smaller length tracks in the 3D vortex street, specifically for primary vortices.

The neighborhood component \mathcal{N} of the score helps balance the effect of the variation in the location of maxima. A time step-specific simplification threshold may have resulted in a single maximum representing the primary vortex, alleviating this anomaly.

7.5 Summary

We introduced TVEG, a time-varying extremum graph to facilitate analysis and visualization of time-varying scalar fields. To the best of our knowledge, TVEG is one of the first time-varying topological structures based on extremum graphs. The structure is easy to compute and interpret. We demonstrate its utility in feature tracking tasks and analysis of synthetic and simulated data.

Chapter 8

Conclusions and future work

The contributions of this thesis are two fold. One, comparison measures for merge trees, global and local. Two, comparison measures for extremum graphs. We have also presented time-varying extremum graph which extends the extremum graph to time-varying data. We have provided some theoretical guarantees and efficient algorithms to compute them. The focus has been on intuitive measures. We have showcased their utility with a lot of applications which handle data many scientific domains.

8.1 Impact

There have been many comparison measures for merge trees that were introduced following our work. We name two such measures which are similar in flavor to our measures, Pont et.al. [94] extend `MTEd` to define Wasserstein distance between merge trees which allows computation of barycenters. Wetzels et.al [136] define branch decomposition independent tree edit distance which is asymptotically slow ($O(n^4)$ since it uses branch decompositions instead of the trees) but independent of the choice of branch decompositions.

8.2 Future work

We discuss potential future directions specifically for the comparison measures which we have presented, followed by some general directions which can be explored.

8.2.1 Merge tree measures

The `OTED` algorithm is limited by the slow running time, but considers a general gap model which might be useful if the model can be adapted for the type of changes merge trees undergo. A definition of a meaningful ordering upon merge trees can also enhance its utility.

Theoretical properties of the MTED needs more work. The MTED, though a metric is not stable. While we use a stabilization parameter to alleviate this effect in practice, a stabilization strategy with theoretical guarantees will make the MTED truly stable. In practice we observe that MTED is more discriminative than bottleneck distance but a theoretical guarantee will strengthen this claim. This also applies to the local measure LMTED. Such guarantees also would help in determining the relationship between these measures and other existing measures in terms of their discriminative power.

A comparative visualization framework including all these measures is an ongoing work which we aim continue in the near future, leading to a publicly available tool.

8.2.2 Extremum graph measures

In case of the extremum graph based measures PDEG and GWEG, stability results are harder, since the graphs themselves are not very stable w.r.t the scalar fields. Analysis of how the measures are related and whether a hierarchy of measures can be built based on their discriminative power is another potential direction which requires further exploration. Another challenge is to extend these measures to Morse-Smale graphs (i.e. 1-skeletons of Morse-Smale complexes) which includes saddle-saddle connections.

The criteria employed to compute temporal correspondences in TVEG may be extended to incorporate other attributes of critical points and extremum graphs. Analyzing the effect of these attributes on the correspondences is an interesting topic for future work.

8.2.3 General directions

Many of the comparison measures for merge trees use the hierarchy but are bottom-up in nature. A measure which can be computed top-down would lead to progressive comparison measures which can be computed for coarser trees first and then finer details can be added if required. This would eliminate the need for simplified trees.

While running times ranging from $O(n^2)$ to $O(n^4)$ sounds great in theory, in practice where interactive visualizations are required, this might still be costly. Parallel algorithms to compute comparison measures would make the measures scalable.

Comparison measures for multi-fields is still nascent stage. Topological descriptors like Jacobi sets and Reeb spaces are less understood which makes designing measures a challenging prospect.

Bibliography

- [1] Aditya Acharya and Vijay Natarajan. A Parallel and Memory Efficient Algorithm for Constructing the Contour Tree. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 271–278, 2015. [8](#)
- [2] Pankaj Agarwal, Kyle Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff Distance for Metric Trees. *ACM Transactions on Algorithms (TALG)*, 14(2):1–20, 2018. [10](#)
- [3] James Ahrens, Berk Geveci, and Charles Law. ParaView: An End-User Tool for Large-Data Visualization. In Charles D. Hansen and Chris R. Johnson, editors, *Visualization Handbook*, pages 717–731. Butterworth-Heinemann, Burlington, 2005. [8](#), [53](#)
- [4] David Alvarez-Melis and Tommi Jaakkola. Gromov-Wasserstein Alignment of Word Embedding Spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, 2018. [14](#)
- [5] Thomas Banchoff. Critical Points and Curvature for Embedded Polyhedral Surfaces. *The American Mathematical Monthly*, 77(5):475–485, 1970. [7](#)
- [6] Ulrich Bauer. Ripser. <https://github.com/Ripser/ripser>, 2021. [8](#)
- [7] Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring Distance between Reeb Graphs. In *Proceedings of the 30th annual symposium on Computational geometry*, pages 464–474. ACM, 2014. [10](#), [15](#), [26](#), [106](#)
- [8] Ulrich Bauer, Michael Kerber, Jan Reininghaus, Hubert Wagner, and Bryn Keller. Persistent Homology Algorithm Toolbox (PHAT). <https://bitbucket.org/phant-code/phant/>, 2021. [8](#)

- [9] Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix Reordering Methods for Table and Network Visualization. *Computer Graphics Forum*, 35(3):693–716, 2016. [88](#)
- [10] Kenes Beketayev, Damir Yeliussizov, Dmitriy Morozov, Gunther Weber, and Bernd Hamann. Measuring the Distance between Merge Trees. In *Topological Methods in Data Analysis and Visualization III*, pages 151–165. Springer, 2014. [10](#), [15](#)
- [11] Paul Bendich, Herbert Edelsbrunner, Dmitriy Morozov, and Amit Patel. Homology and Robustness of Level and Interlevel Sets. *Homology, Homotopy and Applications*, 15:51–72, 2013. [18](#)
- [12] Harsh Bhatia, Attila Gyulassy, Vincenzo Lordi, John Pask, Valerio Pascucci, and Peer-Timo Bremer. TopoMS: Comprehensive Topological Exploration for Molecular and Condensed-matter Systems. *Journal of Computational Chemistry*, 39(16):936–952, 2018. [103](#), [125](#)
- [13] Silvia Biasotti, Andrea Cerri, Alexander Bronstein, and Michael Bronstein. Quantifying 3D Shape Similarity Using Maps: Recent Trends, Applications and Perspectives. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014. [62](#)
- [14] Silvia Biasotti, Andrea Cerri, Masaki Aono, A. Ben Hamza, Valeria Garro, Andrea Giachetti, Daniela Giorgi, Afzal Godil, C. Li, Chika Sanada, Michela Spagnuolo, Atsushi Tatsuma, and Santiago Velasco-Forero. Retrieval and Classification Methods for Textured 3D Models: A Comparative Study. *The Visual Computer*, 32(2):217–241, 2016. [62](#)
- [15] Philip Bille. A Survey on Tree Edit Distance and Related Problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005. [30](#)
- [16] Peer-Timo Bremer, Gunther Weber, Valerio Pascucci, Marc Day, and John B. Bell. Analyzing and Tracking Burning Structures in Lean Premixed Hydrogen Flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010. [12](#), [103](#)
- [17] Peer-Timo Bremer, Gunther Weber, Julien Tierny, Valerio Pascucci, Marc Day, and John Bell. Interactive Exploration and Analysis of Large-scale Simulations using Topology-based Data Segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011. [12](#)

- [18] Stefan Bruckner and Torsten Möller. Isosurface Similarity Maps. *Computer Graphics Forum*, 29(3):773–782, 2010. [9](#)
- [19] Peter Bubenik and Pawel Dlotko. A Persistence Landscapes Toolbox for Topological Statistics. *Journal of Symbolic Computation*, 78:91–114, 2017. [118](#)
- [20] Dmitri Burago, Yuri Burago, and Sergei Ivanov. *A Course in Metric Geometry*, volume 33. American Mathematical Society, 2001. [29](#)
- [21] Gunnar Carlsson and Vin de Silva. Zigzag Persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010. [18](#)
- [22] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing Contour Trees in All Dimensions. *Computational Geometry*, 24(2):75–94, 2003. [1](#), [8](#), [21](#)
- [23] Liqun Chen, Zhe Gan, Yu Cheng, Linjie Li, Lawrence Carin, and Jingjing Liu. Graph Optimal Transport for Cross-domain Alignment. In *International Conference on Machine Learning*, pages 1542–1553. PMLR, 2020. [14](#)
- [24] Samir Chowdhury and Facundo Mémoli. The Gromov-Wasserstein Distance between Networks and Stable Network Invariants. *Information and Inference: A Journal of the IMA*, 8(4):757–787, 2019. [14](#)
- [25] Samir Chowdhury, David Miller, and Tom Needham. Quantized Gromov-Wasserstein. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 811–827. Springer, 2021. [14](#)
- [26] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and Vineyards by Updating Persistence in Linear Time. In *Proceedings of the 22nd annual Symposium on Computational Geometry*, pages 119–126, 2006. [12](#)
- [27] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of Persistence Diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007. [9](#), [15](#), [28](#)
- [28] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz Functions Have L_p -stable Persistence. *Foundations of Computational Mathematics*, 10(2):127–139, 2010. [9](#), [15](#), [29](#)
- [29] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009. [43](#)

- [30] Carlos Correa, Peter Lindstrom, and Peer-Timo Bremer. Topological spines: A Structure-preserving Visual Representation of Scalar Fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1842–1851, 2011. [1](#), [6](#), [8](#), [24](#), [125](#)
- [31] Somenath Das, Raghavendra Sridharamurthy, and Vijay Natarajan. Time-varying Extremum Graphs. In preparation, 2022. [6](#), [7](#), [16](#), [121](#)
- [32] Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian Sheppard. Skeletonization and Partitioning of Digital Images using Discrete Morse Theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):654–666, 2014. [125](#)
- [33] Ismail Demir, Christian Dick, and Rüdiger Westermann. Multi-charts for Comparative 3D Ensemble Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2694–2703, 2014. [2](#)
- [34] Tamal Dey, Dayu Shi, and Yusu Wang. Comparing Graphs via Persistence Distortion. In *Proceedings of the 31st Symposium on Computational Geometry*, pages 491–506, 2015. [5](#), [13](#), [26](#), [29](#), [104](#), [109](#), [110](#)
- [35] Barbara Di Fabio and Claudia Landi. The Edit Distance for Reeb Graphs of Surfaces. *Discrete & Computational Geometry*, 55(2):423–461, 2016. [10](#), [15](#)
- [36] Harish Doraiswamy. Topological Data Analysis using Contour Trees (contour-tree). <https://github.com/harishd10/contour-tree>, 2021. [8](#)
- [37] Harish Doraiswamy and Vijay Natarajan. Computing Reeb Graphs as a Union of Contour Trees. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):249–262, 2013. [8](#), [53](#)
- [38] Soumya Dutta, Jonathan Woodring, Han-Wei Shen, Jen-Ping Chen, and James Ahrens. Homogeneity Guided Probabilistic Data Summaries for Analysis and Visualization of Large-scale Data Sets. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 111–120, 2017. [2](#)
- [39] Herbert Edelsbrunner and John Harer. Jacobi Sets of Multiple Morse Functions. *Foundations of Computational Mathematics, Minneapolis*, 8:35–57, 2002. [7](#)
- [40] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society., 2010. [9](#), [16](#), [18](#), [20](#), [23](#)

- [41] Herbert Edelsbrunner and Ernst Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990. [18](#)
- [42] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological Persistence and Simplification. In *Foundations of Computer Science*, pages 454–463. IEEE, 2000. [1](#), [21](#), [25](#), [125](#)
- [43] Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Morse-Smale Complexes for Piecewise Linear 3-manifolds. In *Proceedings of the 19th annual symposium on Computational geometry*, pages 361–370. ACM, 2003. [1](#), [8](#), [25](#)
- [44] Herbert Edelsbrunner, John Harer, and Afra Zomorodian. Hierarchical Morse-Smale Complexes for Piecewise Linear 2-manifolds. *Discrete & Computational Geometry*, 30(1): 87–107, 2003. [1](#), [8](#), [125](#)
- [45] Herbert Edelsbrunner, John Harer, Ajith Mascarenhas, Valerio Pascucci, and Jack Snoeyink. Time-varying Reeb graphs for Continuous Space-time Data. *Computational Geometry: Theory and Applications*, 41(3):149–166, 2008. [12](#)
- [46] Brittany T Fasy, Jisu Kim, Fabrizio Lecci, Clement Maria, David L. Millman, and Vincent Rouvreau. R-TDA. <https://rdrr.io/cran/TDA/>, 2021. [8](#)
- [47] Pascal Ferraro and Christophe Godin. A Distance Measure between Plant Architectures. *Annals of Forest Science*, 57(5):445–461, 2000. [13](#)
- [48] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, et al. POT: Python Optimal Transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021. URL <https://pythonot.github.io/>. [110](#)
- [49] Robin Forman. A User’s Guide to Discrete Morse Theory. *Séminaire Lotharingien de Combinatoire*, 48:1–35, 2002. [125](#)
- [50] Daiji Fukagawa, Takeyuki Tamura, Atsuhiko Takasu, Etsuji Tomita, and Tatsuya Akutsu. A Clique-based Method for the Edit Distance between Unordered Trees and its Application to Analysis of Glycan Structures. *BMC Bioinformatics*, 12(1):1–9, 2011. [13](#)
- [51] Wolfram von Funck, Tino Weinkauff, Holger Theisel, and Hans-Peter Seidel. Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow

- Experiments. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1396–1403, 2008. [64](#), [128](#)
- [52] Todd Gillette, Parsa Hosseini, and Giorgio Ascoli. Topological Characterization of Neuronal Arbor Morphology via Sequence Representation: II - Global Alignment. *BMC Bioinformatics*, 16(1):209–226, 2015. [13](#)
- [53] Mikhael Gromov. *Metric structures for Riemannian and non-Riemannian spaces*, volume 152. Springer, 1999. [14](#), [29](#)
- [54] Gudhi Developers. Gudhi 3.4.1. <http://gudhi.gforge.inria.fr/>, 2021. [8](#)
- [55] Attila Gyulassy. MSCEER - Morse Smale Complex Extraction, Exploration, Reasoning. <https://github.com/sci-visus/MSCEER>, 2021. [8](#)
- [56] Attila Gyulassy, Vijay Natarajan, Valerio Pascucci, Peer-Timo Bremer, and Bernd Hamann. A Topological Approach to Simplification of Three-dimensional Scalar Functions. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):474–484, 2006. [25](#)
- [57] Attila Gyulassy, Mark Duchaineau, Vijay Natarajan, Valerio Pascucci, Eduardo Bringa, Andrew Higginbotham, and Bernd Hamann. Topologically Clean Distance Fields. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1432–1439, 2007. [103](#)
- [58] Attila Gyulassy, Vijay Natarajan, Valerio Pascucci, and Bernd Hamann. Efficient Computation of Morse-Smale complexes for Three-dimensional Scalar Functions. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1440–1447, 2007. [8](#)
- [59] Attila Gyulassy, Aaron Knoll, Kah Chun Lau, Bei Wang, Peer-Timo Bremer, Michael Papka, Larry Curtiss, and Valerio Pascucci. Interstitial and Interlayer Ion Diffusion Geometry Extraction in Graphitic Nanosphere Battery Materials. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):916–925, 2015. [103](#)
- [60] Attila Gyulassy, Aaron Knoll, Kah Chun Lau, Bei Wang, Peer-Timo Bremer, Michael Papka, Larry Curtiss, and Valerio Pascucci. Morse-Smale Analysis of Ion Diffusion in ab-initio Battery Materials Simulations. In *Topological Methods in Data Analysis and Visualization IV*, pages 135–149. Springer, 2015. [103](#)
- [61] Attila Gyulassy, Peer-Timo Bremer, and Valerio Pascucci. Shared-Memory Parallel Computation of Morse-Smale Complexes with Improved Accuracy. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1183–1192, 2019. [8](#)

- [62] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Toshiyasu Kunii. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212. ACM, 2001. 2, 61, 64
- [63] Michael Kerber, Dmitriy Morozov, and Arnur Nigmatov. HERA. https://bitbucket.org/grey_narn/hera/, 2021. 8
- [64] Tanguy Kerdoncuff, Rémi Emonet, and Marc Sebban. Sampled Gromov Wasserstein. *Machine Learning*, 110(8):2151–2186, 2021. 14
- [65] Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A Tree-edit-distance Algorithm for Comparing Simple, Closed Shapes. In *Proceedings of the 11th annual ACM-SIAM Symposium On Discrete Algorithms*, pages 696–704, 2000. 13
- [66] Wiebke Köpp and Tino Weinkauff. Temporal Treemaps: Static Visualization of Evolving Trees. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):534–543, 2018. 12
- [67] Harold Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955. 50
- [68] Daniel Laney, Peer-Timo Bremer, Ajit Mascarenhas, Paul Miller, and Valerio Pascucci. Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1053–1060, 2006. 11
- [69] Yongjin Lee, Senja D Barthel, Pawel Dlotko, S Mohamad Moosavi, Kathryn Hess, and Berend Smit. Quantifying Similarity of Pore-geometry in Nanoporous Materials. *Nature Communications*, 8(1):1–8, 2017. 116, 118, 119
- [70] Michael Levandowsky and David Winter. Distance between Sets. *Nature*, 234(5323):34, 1971. 46
- [71] Mingzhe Li, Sourabh Palande, Lin Yan, and Bei Wang. Sketching Merge Trees for Scientific Data Visualization. *arXiv preprint arXiv:2101.03196*, 2021. 14
- [72] Jonas Lukasczyk, Gunther Weber, Ross Maciejewski, Christoph Garth, and Heike Leitte. Nested tracking graphs. *Computer Graphics Forum*, 36(3):12–22, 2017. 11, 12

- [73] Jonas Lukasczyk, Christoph Garth, Gunther Weber, Tim Biedert, Ross Maciejewski, and Heike Leitte. Dynamic Nested Tracking Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):249–258, 2019. [11](#), [12](#)
- [74] Richard Luis Martin, Berend Smit, and Maciej Haranczyk. Addressing Challenges of Identifying Geometrically Diverse Sets of Crystalline Porous Materials. *Journal of chemical information and modeling*, 52(2):308–318, 2012. [118](#)
- [75] Ajith Mascarenhas and Jack Snoeyink. Isocontour based Visualization of Time-varying Scalar Fields. In Torsten Möller, Bernd Hamann, and Robert Russell, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 41–68. Springer, 2009. [11](#)
- [76] Talha Bin Masood, Dilip Mathew Thomas, and Vijay Natarajan. Scalar Field Visualization via Extraction of Symmetric Structures. *The Visual Computer*, 29(6-8):761–771, 2013. [2](#)
- [77] Talha Bin Masood, Signe Sidwall Thygesen, Mathieu Linares, Alexei I. Abrikosov, Vijay Natarajan, and Ingrid Hotz. Visual Analysis of Electronic Densities and Transitions in Molecules. *Computer Graphics Forum*, 40(3):287–298, 2021. [103](#)
- [78] Yukio Matsumoto. *An Introduction to Morse Theory*, volume 208. American Mathematical Society, 2002. [16](#)
- [79] Matt McVicar, Benjamin Sach, Cédric Mesnage, Jeffrey Lijffijt, Eirini Spyropoulou, and Tjil De Bie. SuMoTED: An Intuitive Edit Distance between Rooted Unordered Uniquely-labelled Trees. *Pattern Recognition Letters*, 79:52–59, 2016. [13](#)
- [80] Facundo Mévoli. On the use of Gromov-Hausdorff Distances for Shape Comparison. In Mario Botsch, Renato Pajarola, Baoquan Chen, and Matthias Zwicker, editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007. [14](#), [29](#)
- [81] Facundo Mévoli. Gromov-Wasserstein Distances and the Metric Approach to Object Matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011. [5](#), [14](#), [30](#), [104](#), [109](#)
- [82] John Milnor. *Morse Theory*. Princeton University Press, New Jersey, NY, USA, 1963. [16](#), [18](#)

- [83] Dmitriy Morozov. Dionysus. <https://mrzv.org/software/dionysus2/>, 2021. 8
- [84] Dmitriy Morozov, Kenes Beketayev, and Gunther Weber. Interleaving Distance between Merge Trees. *Discrete & Computational Geometry*, 49(52):22–45, 2013. 8, 9, 15
- [85] Suthambhara Nagaraj, Vijay Natarajan, and Ravi Nanjundiah. A Gradient-Based Comparison Measure for Visual analysis of Multifield Data. *Computer Graphics Forum*, 30(3):1101–1110, 2011. 2
- [86] Vidit Nanda. Perseus, the Persistent Homology Software. <http://www.sas.upenn.edu/~vnanda/perseus>, 2021. 8
- [87] Vidya Narayanan. Complete Extremum Graph. https://bitbucket.org/vgl_iisc/compextgraph/, 2021. 8
- [88] Vidya Narayanan, Dilip Mathew Thomas, and Vijay Natarajan. Distance between Extremum Graphs. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 263–270, 2015. 10, 15, 52, 57, 115
- [89] Patrick Oesterling, Christian Heine, Gunther Weber, Dmitriy Morozov, and Gerik Scheuermann. Computing and Visualizing Time-varying Merge Trees for High-dimensional Data. In Hamish Carr, Christoph Garth, and Tino Weinkauf, editors, *Topological Methods in Data Analysis and Visualization IV*, pages 87–101. Springer, 2017. 2, 12
- [90] Karran Pandey, Talha Bin Masood, Saurabh Singh, Ingrid Hotz, Vijay Natarajan, and Tejas Murthy. Morse Theory-based Segmentation and Fabric Quantification of Granular Materials. *Granular Matter*, 24(1):1–20, 2022. 103
- [91] Valerio Pascucci, Kree Cole-McLaughlin, and Giorgio Scorzelli. Multi-resolution Computation and Presentation of Contour Trees. *Proceedings of the IASTED Conference on Visualization, Imaging, and Image Processing*, pages 452–290, 2004. 23
- [92] Persim Developers. Distances and representations of persistence diagrams (Persim 0.3.0). <https://github.com/scikit-tda/persim>, 2021. 8
- [93] Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-Wasserstein Averaging of Kernel and Distance Matrices. In *International Conference on Machine Learning*, pages 2664–2672. PMLR, 2016. 14, 15, 30, 111

BIBLIOGRAPHY

- [94] Mathieu Pont, Jules Vidal, Julie Delon, and Julien Tierny. Wasserstein Distances, Geodesics and Barycenters of Merge Trees. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):291–301, 2022. [139](#)
- [95] Georges Reeb. Sur les points singuliers d’une forme de Pfaff complètement intégrable ou d’une fonction numérique. *CR Acad. Sci. Paris*, 222(847-849):2, 1946. [1](#), [7](#)
- [96] Jan Reininghaus, Jens Kasten, Tino Weinkauf, and Ingrid Hotz. Efficient Computation of Combinatorial Feature Flow Fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1563–1573, 2011. [12](#)
- [97] Bastian Rieck, Heike Leitte, and Filip Sadlo. Hierarchies and Ranks for Persistence Pairs. In *Topological Methods in Data Analysis and Visualization V*, pages 3–17, 2017. [13](#)
- [98] Vanessa Robins, Peter John Wood, and Adrian P Sheppard. Theory and Algorithms for Constructing Discrete Morse Complexes from Grayscale Digital Images. *IEEE Transactions on pattern analysis and machine intelligence*, 33(8):1646–1658, 2011. [104](#)
- [99] Yusuf Sahillioglu. Recent Advances in Shape Correspondence. *The Visual Computer*, 36(8):1705–1721, 2020. [62](#)
- [100] Himangshu Saikia. A merge tree library (mtlib). <https://github.com/hsaikia/mtlib>, 2021. [8](#)
- [101] Himangshu Saikia and Tino Weinkauf. Global Feature Tracking and Similarity Estimation in Time-Dependent Scalar Fields. *Computer Graphics Forum*, 36(3):1–11, 2017. [2](#), [10](#), [12](#), [101](#), [102](#), [128](#), [135](#)
- [102] Himangshu Saikia, Hans-Peter Seidel, and Tino Weinkauf. Extended Branch Decomposition Graphs: Structural Comparison of Scalar Data. *Computer Graphics Forum*, 33(3):41–50, 2014. [2](#), [10](#), [11](#), [23](#), [24](#), [49](#)
- [103] Himangshu Saikia, Hans-Peter Seidel, and Tino Weinkauf. Fast Similarity Search in Scalar Fields using Merging Histograms. In *Topological Methods in Data Analysis and Visualization IV*, pages 121–134, 2015. [10](#), [11](#), [15](#)
- [104] Natascha Sauber, Holger Theisel, and Hans-Peter Seidel. Multifield-graphs: An Approach to Visualizing Correlations in Multifield Scalar Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):917–924, 2006. [2](#)

BIBLIOGRAPHY

- [105] Stefanie Schirmer and Robert Giegerich. Forest Alignment with Affine Gaps and Anchors, applied in RNA Structure Comparison. *Theoretical Computer Science*, 483:51–67, 2013. 13
- [106] Andrea Schnorr, Dirk Helmrich, Hank Childs, Torsten Kuhlen, and Bernd Hentschel. Feature Tracking Utilizing a Maximum-weight Independent Set Problem. In *Proc. IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 6–15. IEEE, 2019. 12
- [107] Andrea Schnorr, Dirk Helmrich, Dominik Denker, Torsten Kuhlen, and Bernd Hentschel. Feature Tracking by Two-Step Optimization. *IEEE Transactions on Visualization and Computer Graphics*, 26(6):2219–2233, 2020. 12
- [108] Scientific visualization contest. IEEE Vis. scientific visualization contest. <http://www.uni-kl.de/sciviscontest/>, 2016. 128
- [109] Ariel Shamir, Chandrajit Bajaj, and Bong-Soo Sohn. Progressive Tracking of Isosurfaces in Time-varying Scalar Fields. Technical report, CS & TICAM Technical Report TR-02-4, University of Texas, 2002. 11
- [110] Nithin Shivashankar and Vijay Natarajan. Parallel Computation of 3D Morse-Smale Complexes. *Computer Graphics Forum*, 31(3pt1):965–974, 2012. URL <http://doi.wiley.com/10.1111/j.1467-8659.2012.03089.x>. 8, 125, 128
- [111] Nithin Shivashankar and Vijay Natarajan. mscomplex3d. https://bitbucket.org/vgl_iisc/mscomplex-3d/, 2021. 8
- [112] Nithin Shivashankar, Senthilnathan Maadasamy, and Vijay Natarajan. Parallel Computation of 2D Morse-Smale Complexes. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1757–1770, 2012. 8
- [113] Nithin Shivashankar, Pratyush Pranav, Vijay Natarajan, Rien van de Weygaert, EG Patrick Bos, and Steven Rieder. Felix: A Topology based Framework for Visual Exploration of Cosmic Filaments. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1745–1759, 2015. 103
- [114] Bong-Soo Sohn and Chandrajit Bajaj. Time-varying Contour Topology. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14–25, 2005. 11

BIBLIOGRAPHY

- [115] Maxime Soler, Mélanie Plainchault, Bruno Conche, and Julien Tierny. Topologically Controlled Lossy Compression. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 46–55, 2018. [5](#), [95](#), [97](#)
- [116] Maxime Soler, Mélanie Plainchault, Bruno Conche, and Julien Tierny. Lifted Wasserstein Matcher for Fast and Robust Topology Tracking. In *Proc. IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 23–33. IEEE, 2018. [12](#), [15](#), [134](#)
- [117] Raghavendra Sridharamurthy and Vijay Natarajan. Comparative Analysis of Merge Trees using Local Tree Edit Distance. *IEEE Transactions on Visualization and Computer Graphics*, page Preprint, 2022. [4](#), [7](#), [16](#), [70](#), [135](#)
- [118] Raghavendra Sridharamurthy and Vijay Natarajan. Comparing Extremum Graphs. In preparation, 2022. [5](#), [7](#), [16](#), [103](#)
- [119] Raghavendra Sridharamurthy, Adhitya Kamakshidasan, and Vijay Natarajan. Edit Distances for Comparing Merge Trees. In *IEEE SciVis Posters*, 2017. [3](#), [4](#), [7](#), [16](#), [39](#), [40](#), [115](#)
- [120] Raghavendra Sridharamurthy, Talha Bin Masood, Adhitya Kamakshidasan, and Vijay Natarajan. Edit Distance between Merge Trees. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):1518–1531, 2020. [4](#), [7](#), [16](#), [39](#), [40](#), [70](#), [72](#), [76](#), [77](#), [85](#), [86](#), [87](#), [95](#), [96](#), [115](#)
- [121] Jun Tao, Martin Imre, Chaoli Wang, Nitesh Chawla, Hanqi Guo, Gökhan Sever, and Seung Hyun Kim. Exploring Time-varying Multivariate Volume Data using Matrix of Isosurface Similarity Maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1236–1245, 2018. [9](#)
- [122] Holger Theisel and Hans-Peter Seidel. Feature Flow Fields. In *Proceedings of the symposium on Data visualisation*, volume 3, pages 141–148, 2003. [12](#)
- [123] Dilip Thomas and Vijay Natarajan. SymmetryViewer. <https://vgl.csa.iisc.ac.in/symmetryViewer/>, 2021. [8](#)
- [124] Dilip Mathew Thomas and Vijay Natarajan. Symmetry in Scalar Field Topology. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2035–2044, 2011. [2](#), [11](#), [49](#), [61](#), [87](#), [91](#)

- [125] Dilip Mathew Thomas and Vijay Natarajan. Detecting Symmetry in Scalar Fields using Augmented Extremum Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2663–2672, 2013. [2](#), [11](#), [61](#), [87](#), [91](#)
- [126] Dilip Mathew Thomas and Vijay Natarajan. Multiscale Symmetry Detection in Scalar Fields by Clustering Contours. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2427–2436, 2014. [2](#), [11](#), [61](#), [87](#), [91](#)
- [127] Julien Tierny and Hamish Carr. Jacobi Fiber Surfaces for Bivariate Reeb Space Computation. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):960–969, 2017. [16](#), [125](#)
- [128] Julien Tierny, Guillaume Favelier, Joshua Levine, Charles Gueunet, and Michael Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2018. <https://topology-tool-kit.github.io/>. [53](#), [99](#), [134](#)
- [129] Julien Tierny, Guillaume Favelier, Joshua Levine, Charles Gueunet, and Michael Michaux. The Topology ToolKit. <https://topology-tool-kit.github.io/>, 2021. [8](#)
- [130] Vayer Titouan, Nicolas Courty, Romain Tavenard, and Rémi Flamary. Optimal Transport for Structured Data with Application on Graphs. In *International Conference on Machine Learning*, pages 6275–6284. PMLR, 2019. [14](#)
- [131] Vayer Titouan, Rémi Flamary, Nicolas Courty, Romain Tavenard, and Laetitia Chapel. Sliced Gromov-Wasserstein. *Advances in Neural Information Processing Systems*, 32, 2019. [14](#)
- [132] Hélène Touzet. Tree Edit Distance with Gaps. *Information Processing Letters*, 85(3):123–129, 2003. [13](#)
- [133] Cédric Villani. *Optimal Transport: Old and New*. Springer-Verlag Berlin Heidelberg, 2000. [47](#)
- [134] Gunther Weber, Peer-Timo Bremer, Marcus Day, John Bell, and Valerio Pascucci. Feature Tracking Using Reeb Graphs. In Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny, editors, *Topological Methods in Data Analysis and Visualization*, pages 241–253. Springer, 2011. [12](#)

- [135] Tino Weinkauff and Holger Theisel. Streak Lines as Tangent curves of a Derived Vector Field. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1225–1234, 2010. [52](#), [55](#)
- [136] Florian Wetzels, Heike Leitte, and Christoph Garth. Branch Decomposition-Independent Edit Distances for Merge Trees. *Computer Graphics Forum*, 41(3):367–378, 2022. [139](#)
- [137] Wathsala Widanagamaachchi, Cameron Christensen, Valerio Pascucci, and Peer-Timo Bremer. Interactive Exploration of Large-scale Time-varying data using Dynamic Tracking Graphs. In *Proc. IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–17. IEEE, 2012. [12](#)
- [138] Wathsala Widanagamaachchi, Jackie Chen, Pavol Klacansky, Valerio Pascucci, Hemanth Kolla, Ankit Bhagatwala, and Peer-Timo Bremer. Tracking Features in Embedded Surfaces: Understanding Extinction in Turbulent Combustion. In *Proc. IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–16. IEEE, 2015. [12](#)
- [139] Hangjun Xu. An Algorithm for Comparing Similarity Between Two Trees. *arXiv preprint arXiv:1508.03381*, 2015. [3](#), [13](#), [31](#), [36](#), [37](#), [39](#), [40](#), [41](#), [42](#), [50](#)
- [140] Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable Gromov-Wasserstein Learning for Graph Partitioning and Matching. *Advances in Neural Information Processing Systems*, 32, 2019. [14](#)
- [141] Lin Yan, Talha Bin Masood, Raghavendra Sridharamurthy, Farhan Rasheed, Vijay Natarajan, Ingrid Hotz, and Bei Wang. Scalar Field Comparison with Topological Descriptors: Properties and Applications for Scientific Visualization. *Computer Graphics Forum*, 40(3):599–633, 2021. [7](#), [8](#), [9](#), [14](#), [15](#), [16](#), [20](#), [23](#), [26](#), [103](#)
- [142] Yan, Lin. Average Merge Tree (AMT). <https://github.com/tdavislab/amt>, 2021. [8](#)
- [143] Kaizhong Zhang. Algorithms for the Constrained Editing Distance between Ordered Labeled Trees and related Problems. *Pattern Recognition*, 28(3):463–474, 1995. [13](#)
- [144] Kaizhong Zhang. A Constrained Edit Distance Between Unordered Labeled Trees. *Algorithmica*, 15:205–222, 1996. [4](#), [13](#), [31](#), [32](#), [34](#), [36](#), [40](#), [50](#), [51](#), [76](#), [78](#), [79](#), [80](#), [82](#)

BIBLIOGRAPHY

- [145] Kaizhong Zhang and Dennis Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989. [13](#)
- [146] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the Editing Distance between Unordered Labeled Trees. *Information Processing Letters*, 42(3):133–139, 1992. [13](#), [33](#)
- [147] Zhen Zhou, Yongzhen Huang, Liang Wang, and Tieniu Tan. Exploring Generalized Shape Analysis by Topological Representations. *Pattern Recognition Letters*, 87:177–185, 2017. [62](#)
- [148] Afra Zomorodian. *Topology for Computing*. Cambridge University Press, 2005. [16](#)