Stability of geometric algorithms

Citation for published version (APA):
Wulms, J. J. H. M. (2020). *Stability of geometric algorithms*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven.

Document status and date:
Published: 25/08/2020

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

Stability of Geometric Algorithms

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen
op dinsdag 25 augustus 2020 om 16:00 uur

door

Jules Joseph Helena Maria Wulms

geboren te Sittard

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. J.J. Lukkien
1e promotor:	dr. K.A.B. Verbeek
2e promotor:	prof.dr. B. Speckmann
copromotor:	dr. W. Meulemans
leden:	dr. M. Nöllenburg (TU Wien)
	prof.dr. F.C.R. Spieksma
	prof.dr. A. Vilanova

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

The work in this thesis is supported by the Netherlands eScience Center (NLeSC) under grant number 027.015.G02.

The work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming Research and Algorithmics).

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5079-1.

Printed by Ipskamp Printing, Enschede.

This thesis has been typeset using PDFLaTeX with the fonts Linux Biolinum and Linux Libertine. The font used on the cover is called Big John PRO, designed by Ion Lucin. The figures in this thesis and the cover were prepared using Ipe.

© 2020 by Jules Joseph Helena Maria Wulms. All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Nil volentibus arduum.

– J.W.C.M. Heijnens (pediatrician)

Acknowledgments

I cannot believe that about 4 years ago, I was unsure about whether I should do a PhD. Looking back now, I most certainly made the right decision: I got to do research, write papers, and give talks, which is sometimes hard, but very satisfying most of the time. On top of that I got to meet and work with many smart, funny, and inspiring people. I would not have wanted to miss out on this journey, and I am going to take this opportunity to thank everyone that was with me along the way.

First of all, my supervisors Bettina, Kevin and Wouter. Three supervisors meant that there was always someone to talk to, not only about research, but also about video games, working in academia, or whatever our conversations would sidetrack to. I am definitely going to miss our weekly meetings. Throughout my PhD it became more and more apparent to me how important your supervisors are, especially when things do not go as planned. Writing a thesis in the COVID-19 lockdown was not easy, but you were there with elaborate feedback and ready to talk to me whenever I felt demotivated. It was always a pleasure working with you, and I cannot thank you enough for the past four years! I will close with a fun fact: history is repeating itself. Kevin was the first student for which Bettina was first promotor, and I will be the first student for which Kevin is first promotor — we just love recursion, right!?

Next, I would like to thank Martin Nöllenburg, Frits Spieksma and Anna Vilanova for reading and reviewing my thesis. Additional thanks to Martin for inviting me to Vienna for a research visit, during which I got to meet the research group at TU Wien and start a collaboration with Martin, Sujoy and Guangping. During my PhD I got to work with many other people, and I would like to thank Juri, Arthur, Ivor, Irina, Marc, Maarten, Tillmann, Irene, Max and Willem for being inspiring co-authors. Lastly, thank you to Kevin Buchin, Bart Jansen, and Alexander Serebrenik, who along with Bettina sparked my interest in academia and became my first co-authors.

For the majority of my PhD, I was part of the Applied Geometric Algorithms (AGA) research group. This not only meant that I had many cool co-workers, but also that there would be a fun activity right around the corner. From organizing the yearly AGA workshop, to kayaking on the Dommel, and from bike trips where we meet parents, to one of the many excellent dinners. Thanks to Bettina, Kevin V., Wouter, Irina, Arthur, Marcel, Ignaz, Quirijn, Tim, Thom, Willem, Irene, Max S., Bram, Pantea, Riet, and Agnes for being wonderful colleagues. I was not part of the AGA group all the time, because we recently fused with the Algorithms group. Our groups have always been close, and I would therefore like to thank Aleks M., Aleks P., Ali, Astrid, Bart, Dániel, Henk, Herman, Huib, Jari, Kevin B., Mark, Martijn, Max K., Mehran, Michał, Morteza, Sándor, Shivesh, and Sudeshna for the interesing lunch conversations, the occasional (sports)activity and being friendly travel companions.

At the start of my PhD, I got to share an office with Quirijn, Tim and Willem, and eventually Bram joined our office as well. At some point, the big PhD office was officially *my* office for a week, until Max S., Pantea and Thijs moved in. While having an office to yourself might sound like a luxury, I always loved having all the office mates around, to share the struggles of our current research and provide some distraction from time to time. A particular memory that stuck with me was the 10-day SoCG marathon, where Willem and I kept each other motivated to finish our papers. This would also be the beginning of our (near daily) SPAR trips, for snacks and late lunches. I have been missing all of this the past few months because of COVID-19, but I am already looking forward to spending more time with you again.

Of course, life is not only about work and I would therefore like to thank some people that are important to me. First the *Buizerds*, which I consider to be a really special group of friends. We have known each other for a long time, are always there to support each other, and happily celebrate special occasions together like birthdays, vacations, graduations, and recently weddings. I really appreciate having you in my life and I hope it stays that way for a long time! Next up are my computer science *bois*. Some of you I have known since the introduction days at TU/e; others we picked up along the way at OGO/SEP. Thanks for pretending to still understand what I work on nowadays, for the board games, and special thanks to Jasper for getting me into rock climbing — it has quickly become a big part of my life. Lastly, many thanks to this randomly assembled list of people: Bob, Raymond, Jacco, Dick, Sam, Aurélien, Kevin, Daan, Andrea and my Erasmus friends from Sweden. We may spend time only occasionally, but when we do, we can always pick up right where we left off.

Last but definitely not least, I want to thank my family, especially my parents, Anita and Jos, and my brother Stan. The last few years were eventful, but at the end of the day, we are always there for each other, and I am very happy to celebrate my defense with you on *Oma's verjaordaag*!

Jules Wulms
Eindhoven, July 2020

Contents

1 Introduction **1**

1.1 Contributions 9

2 A Framework for Algorithm Stability **14**

2.1 Algorithm stability framework 17

2.1.1 Stability vs. solution quality 20

2.1.2 Algorithmic models 22

2.1.3 Applying the framework 24

2.2 Kinetic Euclidean minimum spanning tree 25

2.3 Event stability 27

2.3.1 Event stability analysis 27

2.3.2 Event stability for EMSTs 28

2.4 Topological stability 32

2.4.1 Topological stability analysis 32

2.4.2 Topological stability of EMSTs 35

2.5 Lipschitz stability 40

2.5.1 Lipschitz stability analysis 40

2.5.2 Lipschitz stability of EMSTs 42

2.6 Conclusion 44

3 Kinetic *k*-Centers **46**

3.1 Bounds on topological stability 51

3.2 Algorithms for kinetic *k*-center 61

3.2.1 Unstable k-center algorithms 62
3.2.2 Topologically stable k-center algorithms 66
3.3 Conclusion 70

4 Kinetic Orientation-Based Shape Descriptors 72
4.1 Stateless algorithms 77
4.2 Topological stability 79
4.3 Lipschitz stability 85
4.3.1 Chasing the diametrical pair 86
4.3.2 Lipschitz stability ratio 94
4.3.3 Lipschitz stability of principal component 96
4.4 Conclusion 98

5 Spatially and Temporally Coherent Visual Summaries 100
5.1 Stable dimensionality reduction 106
5.1.1 Stable principal component analysis 106
5.1.2 Gradient-descent methods 109
5.2 Computing 1D orderings 111
5.2.1 Spatial subdivisions 111
5.2.2 Clustering 113
5.3 Metrics 114
5.3.1 Spatial quality 114
5.3.2 Stability 115
5.4 Experimental evaluation 116
5.4.1 Data 117
5.4.2 Running Time 118
5.4.3 Experimental evaluation fish data set (excerpt 1) 118
5.4.4 Experimental evaluation fish data set (excerpt 2) 130
5.4.5 Experimental evaluation Netlogo data set 136
5.4.6 Experimental evaluation clustered data set 144
5.5 Conclusion 150

6 Conclusion and Future Work 154
6.1 Main results 155
6.2 Future work 157

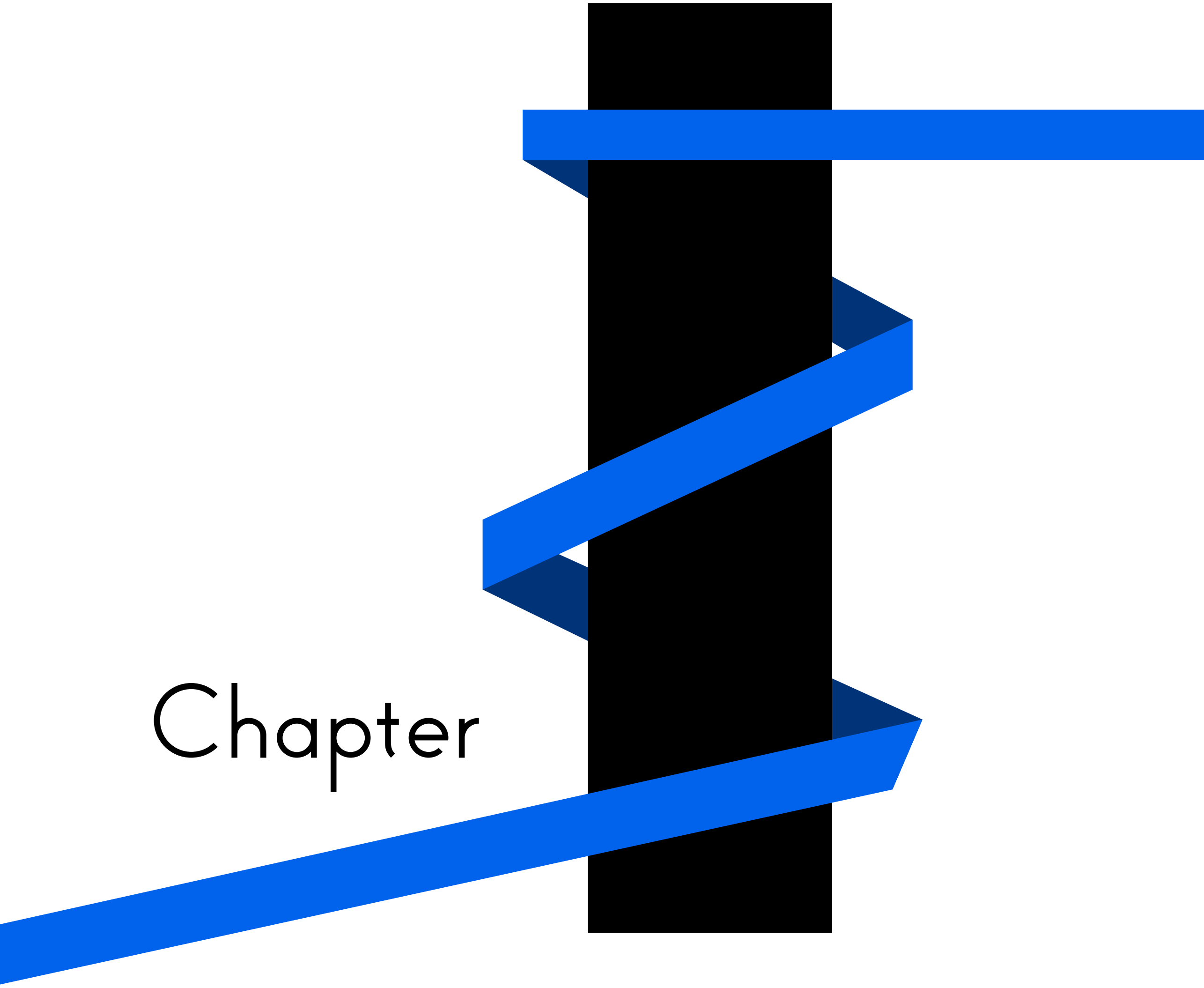
Bibliography **161**

Summary **171**

Samenvatting **175**

Curriculum vitae **179**

Chapter



Introduction

Time-varying data play an important role in our every-day lives. Data on the weather, stock markets, traffic, spreading diseases, power consumption, and public transport, all have attributes that change continuously over time. We use this data all the time: as we get up in the morning, the weather forecast helps us decide what to wear, and over a longer timespan helps us decide what clothes to bring on our vacation. We can quickly check if our flight is delayed before leaving for the airport, or choose the right mode of transport by checking for traffic jams and disruptions in public transport. Maybe we decide to cancel our trip after an outbreak of a contagious disease, which is predicted to affect the destination. Furthermore, as we go by our daily activities, we get notifications of all sorts on smart devices: warnings for low physical activity, suggestions for shows to stream based on recent views, and announcements for new items that are added to an online store we recently ordered from.

Using time-varying data in this way is made possible through technological advances. Examples of these advances are better tracking of changing attributes through GPS and other sensors, mass storage of data in the cloud and global wireless communication allowing for easy access to such storages. As a result, we have multiple ways to use all this data at our disposal. We can easily adapt to live data, such as (air) traffic and stock prices, simply through apps that give access to the (raw) data that is available. While this is sufficient for some kinds of data, in other cases it is beneficial to closely analyze the data to gain insights into the underlying patterns and processes. These insights can then be used to predict future changes. For weather

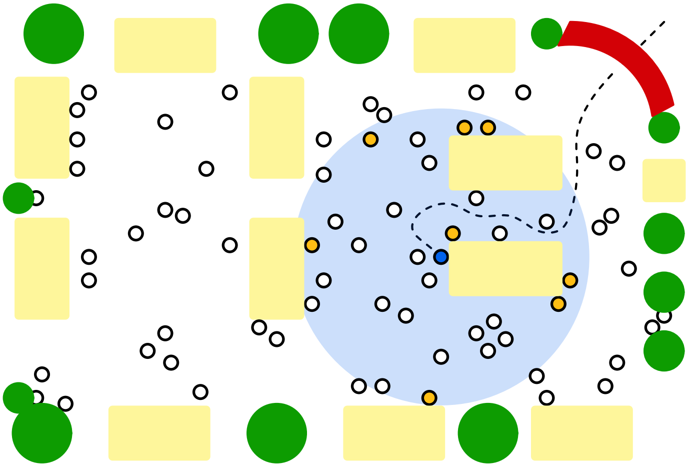


Figure 1.1 The host of a wifi hotspot (blue) tries to maintain a central position to keep their friends (yellow) connected on a bustling market place.

forecasts, such predictions are already done on large scale. Not only is it convenient to have accurate forecasts to plan our activities, but in case of natural disasters such as hurricanes, correctly predicting the trajectory of a hurricane can save many lives. It is therefore important that the analysis of and computation with time-varying data leads to correct insights: these insights can then be used to make educated predictions and decisions.

Stability When working with time-varying data, *stability* can be both a restriction and a requirement. Consider the following examples. You are on a city trip with some of your friends, and you are the only person with internet connectivity, hence you host a hotspot for everyone else. To ensure that a friend stays connected, you have to be in proximity of them at all times. During a visit to the local market, everybody wants to see different stalls and you split up. You find a position for yourself at most a couple of meters away from your friends, to keep them connected to your hotspot as they move through the crowded market area. Your own position on the market, as you try to keep everybody connected, traces a trajectory over time (see Figure 1.1). This trajectory has an important constraint, namely that your position cannot change much in a short amount of time, nor can it change discontinuously: the market is too crowded for you to run and you cannot teleport either. As a result, the temporal pattern must change smoothly and slowly over time.

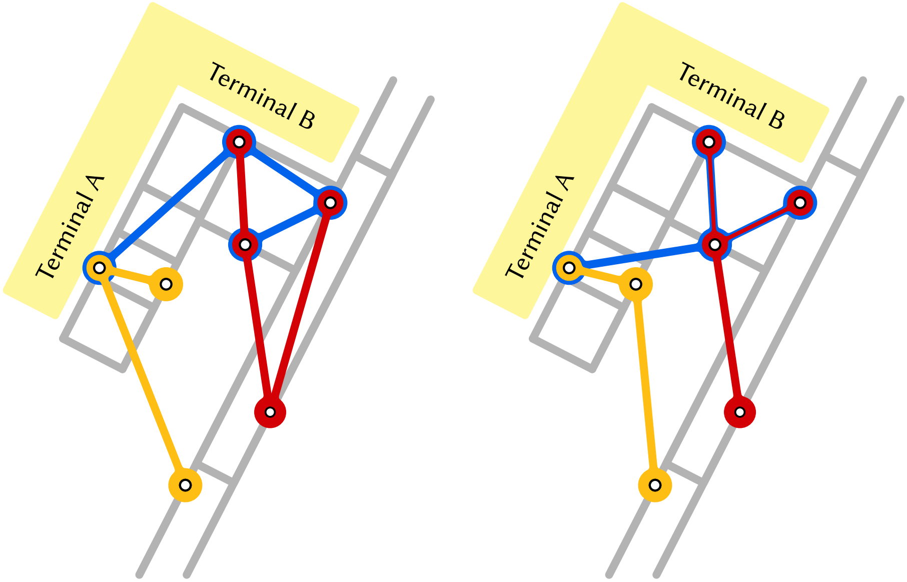


Figure 1.2 Changing the connecting lines in a set visualization communicates change in the data, even when sets and positions are unchanged.

For the next example we want to visualize planes in the vicinity of an airport, by showing their position on a map. We are especially interested in showing which planes are in service to the same airline company, hence we put such planes together in a set. Note that planes are often shared by multiple airlines and hence can be in multiple sets. Sets are visualized by connecting the planes with lines, using a minimal amount of those lines to not clutter the map (see Figure 1.2). Over time, planes land and take off, board at terminals or are being taken to maintenance, which results in stationary, moving, appearing and disappearing planes in the visualization. The positions of the planes are time-varying data, and serve as the input of the visualization. Any changes in the visualization should accurately reflect what happens in the data. Making changes in the visualization while the data is unchanged, even changing only the connecting lines without changing the positions of the planes, would give a user the false impression that the underlying data is changing: The visualizations in Figure 1.2 look quite different, while they show the exact same sets and positions. Thus, for visualizations, there are also constraints on the amount of change that can happen over time.

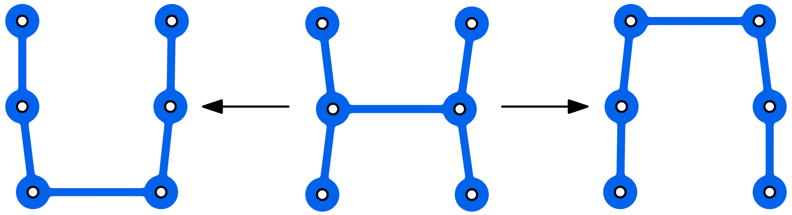


Figure 1.3 Points connected by lines, while minimizing the total line length. Small changes in the positions of points change the layout of optimal lines.

In both examples stability plays an important role: small changes in the time-varying data should lead to small changes in the output of the analysis or visualization. In the first example, we solve a *facility location problem*, in which a facility (the hotspot) should be located close to the input points (the locations of friends). The stability is forced by physical constraints, as it is impossible to quickly move the hotspot on a crowded market. In the second example we considered a visualization of time-varying data. Visualizations are an effective way of communicating data to humans, as humans are very good at identifying visual patterns. A visualization helps the user in forming a *mental map*, a representation of various entities and relations between them in the user's mind. For time-varying data, a visualization should help the user to maintain their mental map as the data changes over time. However, sudden changes in a visualization basically invalidate the established mental map, as we saw in the example: when the visualization changes, but the data does not, then the user can no longer match its own representation of the data to the visualization. This shows that stability is required to make the visualization effective: the visualization must not only truthfully show the data at certain points in time, but should also ensure that temporal patterns are accurately depicted.

Algorithms for time-varying data The problems posed in the two examples are typically solved by an *algorithm*; a procedure or sequence of instructions to solve a (class of) problem(s). Traditionally, algorithms are judged on two important criteria: their solution quality and running time. The solution quality is usually measured by an optimization function f, while the running time is measured by the number of instructions that an algorithm has to execute to solve a problem. Ideally, an algorithm computes an optimal solution efficiently, with few instructions, but in some cases this is impossible. As an example, for so called *NP-hard* problems, no efficient exact algorithms are known. Thus, a trade-off must be made between solution quality and running time. Approximation algorithms show this trade-off clearly, by improving on the running time of exact algorithms, while provably keeping the solution quality close to optimal: An algorithm is an ρ-approximation if the solution is at most a factor ρ worse than an optimal solution, in terms of solution quality.

As we saw in the two examples, stability is a third important quality criterion for algorithms. However, stability often directly contradicts high solution quality: again consider the set visualization example, where we connect the positions of moving planes on a map using lines. In such a visualization, we often want to use minimize ink usage, to prevent clutter on the map. An algorithm producing this kind of visualization would have high solution quality, if it effectively minimized the total line length used. However, as the points in the visualization move, the optimal solution may change drastically even though the input points changed only a little (see Figure 1.3). This shows that a solution with optimal solution quality is not necessarily stable.

On the other hand, if we force stability, then the solution quality may deteriorate, and the result will be useless: in the set visualization we could simply use the same sets throughout the whole visualization. While this is obviously extremely stable, the visualization may eventually clutter the map due to the lines becoming very stretched, and hence the visualization has low quality. This shows that we need to understand the trade-off between stability and solution quality. It is not hard to imagine that there will be more trade-offs, for example between the stability and running time of an algorithm. Figure 1.4 gives an overview of the trade-offs between the three criteria. The ultimate goal is to understand all these trade-offs, between solution quality, running time and stability. However, grasping the trade-off between solution quality and stability is a crucial first step in the development of stable algorithms for time-varying data.

Related work The stability of algorithms or methods has been well-studied in a variety of research areas, such as numerical analysis [60], machine learning [18], control systems [5], and topology [29, 72]. In contrast, the stability of combinatorial algorithms for time-varying data has received little attention in the theoretical computer science community so far. An example where stability has been taken into account is in dynamic map labelling [11, 47, 48, 83], where we label points on a zoomable map. The labels keep the same absolute size and position on the map, while a user can zoom in and out on the map. To prevent labels from overlapping, only a subset of the labels can be shown at a certain zoom level. If we do not carefully choose the labels at each zoom level, some labels may appear and disap-

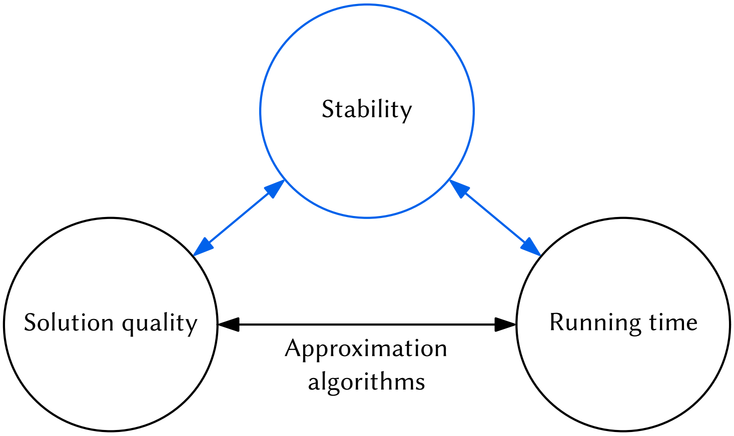


Figure 1.4 Trade-offs between quality criteria for algorithms. Traditional criteria and trade-offs are shown in black, while additional trade-offs for time-varying data are shown in blue.

pear often while zooming, causing inexplicable flickering. The *consistent dynamic labelling* model allows a label to appear and disappear only once. Every label has a single connected zoom range where it is visible, preventing flickering and hence making the labeling much more stable. Furthermore, the so-called simultaneous embeddings [19, 41, 49] ask to create an embedding for multiple graphs on the same set of vertices, such that for each graph no edges intersect. The input of this problem can be interpreted as a time-varying graph where edges appear and disappear over time. Then, the position of the vertices must be stable over time in the solution.

In computational geometry there is a lot of work on algorithms for time-varying data, in the form of *kinetic data structures* (KDSs), introduced by Basch and Guibas [9]. These data structures are used to efficiently maintain optimal solutions to geometric problems with time-varying input. A solution is maintained using a set of certificates, which together prove that the solution is correct. The certificates check properties of a solution, and as long as no certificate fails, the data structure does not need to be updated. For the set visualization on moving points, the certificates in a KDS would prove that a certain combination of lines minimizes the total line length. However, if one of the lines becomes too long, and a different shorter line would also keep the set connected, then a certificate would fail. For each certificate we can compute when it will fail, and hence we can efficiently maintain a solution by jumping to the next certificate failure, and updating the data structure. Such certificate failures are called *events*. An event can coincide with a change in the combinatorial structure of the solution, such as changing the lines in the set visualization, in which case it is called an external event. On the other hand, an event can also be necessary only to update a certificate, and is then named an internal event. When it comes to stability, kinetic data structures offer not much more than an evaluation of the number of events.

Of the few results on stability in computational geometry, most are on the facility location problem [13, 17, 16, 14, 36, 37], and these results come close to how we envision stability analysis: instead of considering stability in isolation, it is of particular interest to understand the trade-offs between solution quality, running time, and stability. The facility location problem is a generalization of the problem we tried to solve in the wifi hotspot example: we choose locations for a set of facilities (hotspots), such that the distance to clients (connected to the hotspot) is minimized. In the time-varying setting the input points are moving and the facilities can move as well to remain close to the points. While for optimal placement of the facilities, the speed at which the facilities move can be unbounded, existing results show that this speed can be brought down, while still approximating the optimal distance between facilities and clients. Thus we can make trade-offs between stability and solution quality for facility location problems.

A framework for stability Although the aforementioned related work considers the stability of algorithms, there is no established theoretical framework to analyze algorithm stability. Such a framework should unify the existing results, and allow us to identify more problems for which a similar analysis works. Furthermore, a framework offers definitions and instructions to respectively describe the stability of algorithms and analyze the trade-offs between stability and solution quality. Even though stability is motivated by real world problems, having a theoretical framework is still very useful, as it allows for provable guarantees on the output of algorithms: visualizations produced by stable algorithms would have the guarantee that they are of high quality and truthfully represent temporal patterns in the input data.

In this thesis, we propose a framework for algorithms stability. The framework distinguishes between various types of stability and characterizes algorithms on traits that influence stability. Since optimal solutions may be unstable, the focus of the framework is on the trade-off between solution quality and stability. As we have seen throughout the introduction, in the areas of (geo)-visualization, graph drawing, and computational geometry, time-varying data is used in various ways. What these areas have in common, is that the time-varying data is often geometric: planes can be

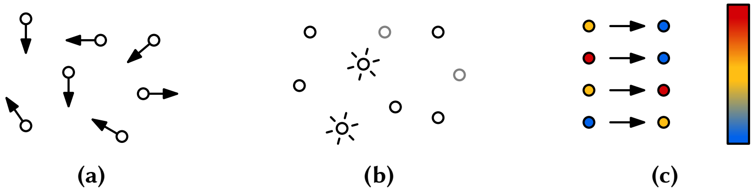


Figure 1.5 Examples of kinetic **(a)**, dynamic **(b)**, and time-varying data **(c)**.

represented as sets of moving points, consistent labels are non-overlapping rectangles, and in facility location problems we cover moving points with disks or squares. For this reason, we focus on geometric algorithms, even though the framework can be applied to other types of algorithms as well. We apply this framework to various theoretical problems from computational geometry, to obtain insights into the trade-offs between solution quality and stability. We then take a more applied point of view and show the effects of improving stability on visualization techniques that summarize data in a single dimension. In the remainder of this section, we describe the distinction between types of time-varying data. Finally, the contributions of this thesis are explained in Section 1.1.

Types of time-varying data As we have seen in the introduction, time-varying data comes in many forms: from moving objects, to changing prices, and from measurements of complex systems such as the weather, to traffic jams that appear and disappear. Depending on which attributes of a dataset are changing over time, we distinguish three types of data. See Figure 1.5 for examples.

kinetic data has a spatial location that changes continuously over time. This data mostly originates from moving objects in the real world, whose movement has been traced over time via geolocation. A school of moving fish is a great example of kinetic data that is retrieved through geolocation. During geolocation the spatial location of the fish is stored as two- or three-dimensional coordinates. For this reason, kinetic data is inherently geometric and forms the basis for many problems in computational geometry. In the next chapters we introduce three geometric problems, which are all kinetic: they take a set of two-dimensional moving points as input.

The aforementioned geolocation usually does not result in continuous data, as it is executed only at (ir)regular intervals. The output of the geolocation is then a set of sampled data points. However, if the granularity is low enough, the difference between two consecutive samples is small. In that case, we would like to abstract from the data samples and interpret the data as continuous: the coordinates of moving objects change continuously over time. In this thesis, whenever we work with kinetic data, we assume the data to be continuous, unless stated otherwise.

dynamic data has data points which over time disappear from the data set, or are (re)introduced. Such changes in the data are respectively called deletions and insertions. For example, consider the group of front runners in a cycle race. As some cyclists catch up to the group, others might fall behind due to fatigue. The newly added front runners are inserted into the group, while the dropouts are deleted. While all problems considered in this thesis have dynamic variants, our focus is on their kinetic counterparts.

time-varying data The last type of data has attributes other than its spatial location or presence, which change continuously over time, and is called time-varying data. The attributes in question are usually of a statistical nature, such as population over time, or income, but can also be general attributes such as the size or color of a data point.

While data can be of a single type, it often falls into multiple categories. For example, we might have data of a flock of birds that migrates from one area to another. The spatial location of the birds changes over time, making this a kinetic dataset. However, during the migration birds might enter the flock, or stay behind to roost, adding a dynamic aspect to the dataset. Furthermore, as we will see in Chapter 5, certain visual summaries are generated for data that is both kinetic and time-varying: the spatial location of the data points is used to compute the structure of a visualization, which can then show various statistical time-varying attributes of the data points.

▸ 1.1 Contributions

In this thesis we study the stability of algorithms from a theoretical point of view, and show how to improve applications that benefit from being stable. We start by introducing a framework that aids in analyzing the stability of algorithms in various ways, each with different advantages and drawbacks. The framework allows us to make trade-offs between solution quality and stability, to create stable algorithms that produce approximations to optimal solutions. By applying the framework to the Euclidean minimum spanning tree problem, we show how such trade-offs can be made. We proceed to use the framework to analyze the stability of both the k-

center and orientation-based shape descriptor problems. Finally, we move from the theoretical problems in computational geometry, to a practical point of view, and improve the stability of visualizations of time-varying data that use orderings.

We end this section with a detailed overview of the contributions in this thesis.

A Framework for Algorithm Stability

We introduce the framework for algorithm stability in Chapter 2. For our framework we subdivide algorithms for time-varying data into stateless, state-aware and clairvoyant algorithms, which respectively have access to data at the current time step, data calculated from previous time steps or data at every time step. Depending on the type of algorithm, different levels of stability can be achieved. Additionally, the framework provides three definitions for measuring stability that each address different aspects. The event stability of a problem counts the number of times the combinatorial structure of the output changes. The topological and Lipschitz stability both enforce the output of an algorithm to change continuously, by imposing a topology or metric on the output space of the algorithm. The Lipschitz stability of an algorithm additionally limits how fast an output can change depending on the change in the input. Because of this additional constraint, Lipschitz stability comes closest to our intuitive definition of stability, namely small changes in the input should lead to small changes in the solution, and is therefore the preferred type of analysis. However, Lipschitz stability analysis is often prohibitively challenging or infeasible. The other types of stability analysis simplify the stability requirements, making the analysis significantly easier. Although these types of stability analysis do not fully capture all aspects of stability, they do offer useful insight into the interplay between problem instances, solutions, and the optimization function. These insights are invaluable for the development of stable algorithms.

We then show how to analyze algorithms using the framework, by proposing stable state-aware algorithms for the Euclidean minimum spanning tree (EMST) problem. To improve the event stability of optimal EMSTs we introduce *k-optimal* solutions, where moving the input at most k units can turn the solution into an optimal solution. We work under the assumption that the trajectories of the input points follow polynomials of bounded degree, and the points stay reasonably spread out during their motion. This results in a constant factor approximation of an optimal EMST, which undergoes at most a linear number of discrete changes. For the topological stability we propose *edge slides* and *edge rotations* to continuously move between spanning trees. These operations move one endpoint of an edge to an adjacent or to any reachable vertex respectively. We then prove upper and lower bounds on the length of topologically stable EMSTs with respect to the length of optimal EMSTs, using edge slides or rotations. Finally, for edge slides we can prove that the solution quality of Lipschitz stable EMSTs can become arbitrarily bad, if the solution cannot change fast enough.

This chapter is based on joint work with Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek [78] which appeared in the Proceedings of the 13th Latin American Symposium on Theoretical Informatics (LATIN 2018).

Kinetic k-Centers

In Chapter 3, we study the k-center problem in a kinetic setting: given a set of continuously moving points P in the plane, determine a set of k (moving) shapes that cover P at every time step, such that the shapes are as small as possible at any point in time. Existing results for this problem require the shapes to move with a bounded speed. Under these conditions it is impossible to approximate optimal solutions for $k > 2$. We therefore study the topological stability of k-centers, which does not bound the speed at which the covering shapes can change.

We consider 4 variants of the k-center problem: in the Euclidean (k-EC) or rectilinear (k-RC) variant of the problem, the k covering shapes are disks or squares, respectively. For each of these variants we distinguish between two different optimization functions: minimizing the radius of the largest shape (-minmax) or minimizing the sum of radii (-minsum). We prove upper and lower bounds on the ratio between the radii of an optimal but unstable solution and the radii of a topologically stable solution—the topological stability ratio. For k-RC-minmax, k-RC-minsum and k-EC-minsum, we prove that the topological stability ratio is exactly 2. The k-EC-minmax problem turns out to be harder: for $k = 2$ we again provide a tight bound of 2, while for $k > 2$ we provide a generic lower bound and tools for investigating the structure of situations where the optimal solution is not unique; these tools allow us to provide a nontrivial upper bound for small k.

Finally, we provide a clairvoyant algorithm to compute an upper bound on the topological stability ratio of an instance to the k-center problem in polynomial time for constant k. The algorithm first computes both an optimal solution and a topologically stable solution, after which the topological stability ratio is found by looking for the maximum ratio between the two solutions over time. Since a full topologically stable solution is computed, we also explore the trade-off between running time and stability for the k-center problem.

This chapter is based on joint work with Ivor van der Hoog, Marc van Kreveld, Wouter Meulemans and Kevin Verbeek [63] and appeared in the Proceedings of the 13th Conference and Workshops on Algorithms and Computation (Best paper WALCOM 2019).

Kinetic Orientation-Based Shape Descriptors

We study three *orientation-based* shape descriptors on a set of continuously moving points in Chapter 4. Specifically, we study the first principal component, the smallest oriented bounding box, and the thinnest strip. Each of these shape descriptors essentially defines a cost capturing the quality of the descriptor and uses the orientation that minimizes the cost. We use these functions to make a trade-off between solution quality and stability.

We first show that there is no *stateless algorithm*, an algorithm that keeps no state over time, that both approximates the minimum cost of a shape descriptor and achieves continuous motion for the shape descriptor. On the other hand, if we turn to state-aware algorithms, we can prove tight bounds on the topological stability ratio for all three shape descriptors. To prove an upper bound on the Lipschitz stabiltiy, we define *chasing* algorithms that attempt to follow the optimal orientation with bounded speed. We show that, under mild conditions, chasing algorithms with sufficient bounded speed approximate the optimal cost at all times for oriented bounding boxes and strips. Since the analysis of such chasing algorithms is challenging and has received little attention in literature, the contributions in this chapter not only include the proved bounds, but also the methods used in the analysis.

This chapter is based on joint work with Wouter Meulemans and Kevin Verbeek [79].

Spatially and Temporally Coherent Visual Summaries

In Chapter 4 we focus on visual summaries using 1D orderings for entities moving in 2D. When exploring large time-varying data sets, visual summaries are a useful tool to identify time intervals of interest for further consideration. A typical approach is to represent the data elements at each time step in a compact one-dimensional form or via a one-dimensional ordering. Such 1D representations can then be placed in temporal order along a time line. As with the output of the theoretical geometric problems, the two main criteria to assess the quality of the resulting visual summaries are the *spatial quality* – how well does the 1D representation capture the structure of the data at each time step, and the *stability* – how coherent are the 1D representations over consecutive time steps or temporal ranges?

We introduce stable techniques, inspired by our work on orientation-based shape descriptors, which are based on well-established dimensionality-reduction techniques: Principle Component Analysis (PCA), Sammon mapping, and t-SNE. These techniques are not inherently stable, hence we adapt them to take stability into account. While for Sammon mapping and t-SNE, we initialize every time step by the solution to the previous time step, for PCA we develop a new clairvoyant algorithm, which we call Stable Principal Component (SPC). The SPC method is explicitly parametrized for stability, allowing a trade-off between the two quality criteria. To take into account that the input data can be clustered, we extend SPC to the Clustered Principal Component (CPC) algorithm, which first finds a clustering on the data, and applies SPC to the different clusters.

We conduct quantitative experiments that compare our stable methods to various state-of-the-art approaches using a set of well-established quality metrics that capture the two main criteria. The quality metrics for spatial quality evaluate for each point whether its k-nearest neighbors in the data are still close in 1D ordering. For stability, we again look for changes in the k-nearest neighbors of each point, but we do so between orderings for consecutive time steps in the output. These experiments demonstrate that stable algorithms outperform existing methods on stability, without sacrificing spatial quality or efficiency. That is, the spatial quality of the best stable algorithm is essentially equivalent to the best spatial quality obtained by any method, while achieving higher stability and often higher efficiency as well.

This chapter is based on joint work with Juri Buchmüller, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek [119].

Other Results

In addition to the contributions in this thesis, the author also worked on grid-based set visualization [51], on movement of point objects by repulsion [52], and on the algorithmic complexity of puzzles in the game *The Witness* [70].

Chapter 2

A Framework for Algorithm Stability

Algorithms play an important role in the analysis and visualization of time-varying data. To understand the temporal patterns in the data, it is important that changes in the output of the algorithm for consecutive time steps correctly reflect changes in the time-varying data. In other words, these algorithms must be stable: small changes in the data lead to small changes in the output. In this chapter we present a theoretical framework for algorithm stability.

As mentioned in the introduction, stability is one of the important criteria in the analysis of algorithms for time-varying data. Traditionally, we care for the solution quality and running time of an algorithm, and these criteria are conflicting: algorithms that find optimal solutions are typically slower than algorithms that are more lenient in solution quality, and compute only approximations to optimal solutions. We expect to find similar trade-offs between stability, and solution quality or running time. For example, a visualization algorithm has high solution quality if at every point in time the produced visual representation represents the time-varying input data well. However, this visualization can be unstable, when there are very sudden changes in the visualization that are not representative of changes in the data. A stable visualization algorithm must therefore ensure that those sudden changes no longer occur, and hence trade some solution quality for stability. The purpose of the theoretical framework for algorithm stability is to formally study this trade-off.

Although stability has often been considered an important criterion for the analysis or visualization of time-varying data, there are currently no suitable generic tools available to formally analyze trade-offs involving stability. The benefit of introducing a theoretical framework for algorithm stability is that it provides a uniform way of analyzing stability and guides in structurally finding stable algorithms. Furthermore, we gain deeper insights in the trade-off between stability and solution quality, and can provide provable bounds on the quality and stability of solutions. An important aspect of this framework is how to measure the stability of an algorithm, and how to analyze trade-offs. In addition, we need to refine the model of an algorithm based on how it interacts with the time-varying data, for which there are several options.

The main focus of the theoretical framework presented in this chapter is to analyze the trade-off between stability and solution quality. While we could simply introduce a measure for stability and analyze the trade-off with solution quality, it turns out that this approach is very cumbersome and unwieldy, and does not often lead to meaningful results. Instead we analyze the solution quality that can be obtained when bounding the stability of the algorithm. Even though the resulting analysis can still be very challenging, we provide different ways to loosen the requirements on stability. Less restrictive stability requirements typically make it easier to analyze the optimal solution quality under these requirements, enabling us to obtain results and insights about the trade-off between stability and solution quality where this would otherwise be prohibitively difficult or impossible.

The framework is set up as follows. We introduce three types of stability analysis of increasing degrees of complexity, along with increasing stability requirements: *event stability*, *topological stability*, and *Lipschitz stability*. While event stability deals only with discrete changes in a solution, topological and Lipschitz stability require the solution to behave continuously. For Lipschitz stability we additionally restrict how fast a solution can change, with respect to changes in the input. Besides these modes of analysis, we distinguish between three models for algorithms on time-varying data: *stateless*, *state-aware* and *clairvoyant* algorithms. The model of an algorithm is based on the availability of time-varying input, and influences how much stability can be achieved. In this chapter, we elaborate further on our stability framework and demonstrate the use of the framework by applying it to the problem of kinetic Euclidean minimum spanning trees (EMSTs). For this problem a set of moving points is connected with line segments at every time step. A solution of high quality ensures that the total length of the line segments is minimized. During the movement of the points, discrete changes to the combinatorial structure of an EMST are necessary to keep a solution optimal. It is therefore an interesting problem for stability analysis.

Related work Even though there are currently no tools available to formally analyze the stability of algorithms, there is still some related work on the topic. In computational geometry, algorithms for time-varying data are extensively studied in the context of *kinetic data structures* (KDSs). These data structures, introduced by Basch and Guibas [9], efficiently maintain solutions to algorithmic problems. Kinetic data structures use a set of certificates, which show that a solution is valid at all times. Whenever a certificate fails, the certificates are updated to again ensure that the solution is valid. Such certificate failures, or *events*, can be counted to analyze the stability of the solution. However, the focus of KDSs is not on stability but on the trade-off between solution quality and running time, and hence we get little insight into the trade-offs involving stability. On the other hand, one of the few times the trade-off between stability and solution quality was examined, was for the facility location problem [13, 17, 16, 14, 36, 37]. The facility location problem in the kinetic setting asks to find locations for a set of (possibly moving) facilities, which are in close proximity to the moving clients they provide to. This is equivalent to covering a set of moving points with covering shapes, such as disks or rectangles. Solutions that use smaller shapes are then considered of higher quality. The authors show that optimal covering shapes can move arbitrarily fast with respect to the movement of the points. However, the speed can be reduced in some cases, without increasing the size of the covering shapes by much. Such provable trade-offs between stability and solution quality are exactly what we want to structurally achieve with our framework. In Chapter 3 we apply our framework to the facility location problem to extend the existing results.

Organization In Section 2.1 we give an overview of our framework for algorithm stability and in Section 2.2 we formally introduce the kinetic EMST problem. In Sections 2.3, 2.4, and 2.5 we describe event stability, topological stability, and Lipschitz stability, respectively. In each of these sections we first describe the respective type of stability analysis in a generic setting, followed by specific results using that type of stability analysis on the kinetic EMST problem. In Section 2.6 we make some concluding remarks on our stability framework.

2.1 Algorithm stability framework

Intuitively, we can say that an algorithm is stable, if small changes in the input lead to small changes in the output. More formally, let Π be an optimization problem that, given an input instance I from a set $\mathcal{I}$, asks for a solution S from a set $\mathcal{S}$ that minimizes (or maximizes) some optimization function $f : \mathcal{I} \times \mathcal{S} \longrightarrow \mathbb{R}$.

An algorithm $\mathcal{A}$ for Π maps an input instance I to a solution S. For an optimal solution $\mathrm{OPT} : \mathcal{I} \longrightarrow \mathcal{S}$ of input instance I, it holds that $f(I, \mathrm{OPT}(I)) = \min_{S \in \mathcal{S}} f(I, S)$ is minimal over all possible solutions $S \in \mathcal{S}$ for I. Note that we did not define $\mathcal{A}$ as a function between input and output. This is intentional and will be further elaborated on in Section 2.1.2, where we distinguish between various algorithmic models that interact differently with the time-varying aspect of the data.

When working with time-varying data, the input instance is not static, and for the purpose of introducing time-varying input, we model time-varying input instances as a sequence of T static inputs $I_1, I_2, \dots, I_T$. In this time-varying setting, algorithm $\mathcal{A}$ maps the input sequence $I_1, I_2, \dots, I_T$ to a sequence of solutions $S_1, S_2, \dots, S_T$. An optimal solution OPT for Π on this time-varying input is then defined by individual static solutions OPT for each input I_i in the sequence, such that $f(I_i, \mathrm{OPT}(I_i))$ is minimized for each input separately. Note that such a solution has optimal solution quality, and completely ignores stability.

To define the *stability* of an algorithm, we need to quantify changes in the input instances and in the solutions. We can do so by imposing a metric on $\mathcal{I}$ and $\mathcal{S}$. Let $d_{\mathcal{I}} : \mathcal{I} \times \mathcal{I} \longrightarrow \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{I}$ and let $d_{\mathcal{S}} : \mathcal{S} \times \mathcal{S} \longrightarrow \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{S}$. Figure 2.1 gives an overview of the above definitions. For an input sequence $\sigma_I = I_1, I_2, \dots, I_T$ and a sequence of solutions $\sigma_S = S_1, S_2, \dots, S_T$ we can define the stability of this solution relative to the input as follows.

$$\mathrm{St}(\sigma_I, \sigma_S) = \max_{i \in [1, T-1]} \frac{d_{\mathcal{S}}(S_i, S_{i+1})}{d_{\mathcal{I}}(I_i, I_{i+1})} \tag{2.1}$$

Finally, let $\Sigma_{\mathcal{I}}$ be the set of all sequences of input instances. We can then define the stability of an algorithm $\mathcal{A}$, which maps every input I_i in sequence σ_I to a solution S_i in a sequence of solutions σ_S, as follows.

$$\mathrm{St}(\mathcal{A}) = \sup_{\sigma_I \in \Sigma_I} \mathrm{St}(\sigma_I, \sigma_S) \tag{2.2}$$

The lower the value for $\mathrm{St}(\mathcal{A})$, the more stable we consider the algorithm $\mathcal{A}$ to be. Note that $\mathrm{St}(\mathcal{A})$ does not depend on f, and hence the stability analysis is not dependent on whether Π is a minimization or maximization problem.

There are many other ways to define the stability of an algorithm given the metrics, and the definition above does not cover all scenarios in which we want to analyze stability. Recall the examples in the introduction: a time-varying set visualization and hosting a wifi hotspot for friends. In the above definition, if there is no change in the input ($I_i = I_{i+1}$) then there should not be any change in the output either

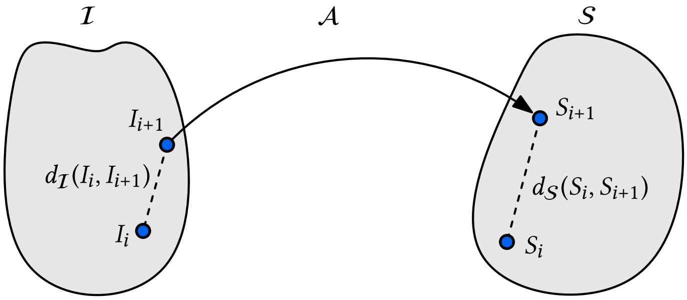


Figure 2.1 Algorithm $\mathcal{A}$ maps input instances from the input space $\mathcal{I}$ to the solution space $\mathcal{S}$. Metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ allow us to reason about how different two instances are in respectively the input and solution space.

($S_i = S_{i+1}$), otherwise St($\mathcal{A}$) is unbounded. This fits the visualization example, where we want to see no changes in the visualization, unless the data is changing. However, when hosting a hotspot for friends, the hotspot can (slowly) move into a better position even though none of the friends are moving. We can adapt Equation 2.1 to express these conditions as well.

$$\mathrm{St}(\sigma_I, \sigma_S) = \frac{\max_{i \in [1,T-1]} d_{\mathcal{S}}(S_i, S_{i+1})}{\max_{i \in [1,T-1]} d_{\mathcal{I}}(I_i, I_{i+1})} \tag{2.3}$$

In this definition, instead of using the amount of change between time steps, we use an upper bound for both the input and solution. Since the upper bound on the changes in the solution can occur between different time steps than the upper bound on the changes in the input, St($\mathcal{A}$) will not be unbounded when the solution changes without any change in the input. Our framework will allow for stability analysis that can adhere to either of these models, whichever fits the problem at hand.

Kinetic time-varying data is typically gathered through geo-location, as already explained in the introduction. Hence, the above setting would suffice to analyze the stability of this kind of time-varying data. However, as the rate at which the movement data is recorded becomes higher, and the difference between consecutive data points becomes smaller, the data closely resembles the continuous motion of real-life objects. In our theoretical framework we would therefore like to abstract from the discrete nature of the data, and work with continuously changing inputs. This approach is very natural for geometric problems and is used abundantly in computational geometry, for example by using continuously moving points as input.

Instead of a sequence of static inputs, we now get a continuous path $I: [0,1] \longrightarrow \mathcal{I}$ through the input space $\mathcal{I}$. For each time $t \in [0,1]$, algorithm $\mathcal{A}$ maps $I(t) \in \mathcal{I}$ to a solution $S(t) \in \mathcal{S}$. Note that the resulting time-varying solution $S: [0,1] \longrightarrow \mathcal{S}$ is not necessarily continuous. An optimal solution OPT for input I minimizes (or maximizes) $f(I(t), \mathrm{OPT}(I(t)))$ for every $t \in [0,1]$, and may have discontinuities as well. The abstraction to (continuous) paths allows us to make a distinction between continuous and discrete changes in the output, which possibly lead to instabilities.

The stability of a solution S relative to I can then be defined similarly to the discrete case, using the metrics on $\mathcal{I}$ and $\mathcal{S}$. The stability is then a generalization of Equation 2.1

$$\mathrm{St}(I,S) = \sup_{t\in[0,1)} \lim_{\epsilon\longrightarrow 0} \frac{d_{\mathcal{S}}(S(t), S(t+\epsilon))}{d_{\mathcal{I}}(I(t), I(t+\epsilon))} \tag{2.4}$$

In this definition of stability, part of the formula is essentially a metric derivative, as we compare the speed along parameterized paths $I(t)$ and $S(t)$ in metric spaces $(\mathcal{I}, d_{\mathcal{I}})$ and $(\mathcal{S}, d_{\mathcal{S}})$, respectively. Let $\mathcal{P}_{\mathcal{I}}$ be the set of continuous paths through $\mathcal{I}$. The stability of an algorithm $\mathcal{A}$ can now also be generalized from Equation 2.2.

$$\mathrm{St}(\mathcal{A}) = \sup_{I\in\mathcal{P}_{\mathcal{I}}} \mathrm{St}(I,S) \tag{2.5}$$

Again, this definition assumes that no change in the input should result in no change in the output, otherwise $\mathrm{St}(\mathcal{A})$ is unbounded. If we want the stability to be less strict as in Equation 2.3, we can also generalize this to the continuous setting:

$$\mathrm{St}(I,S) = \frac{\sup_{t\in[0,1)} \lim_{\epsilon\longrightarrow 0} d_{\mathcal{S}}(S(t), S(t+\epsilon))}{\sup_{t\in[0,1)} \lim_{\epsilon\longrightarrow 0} d_{\mathcal{I}}(I(t), I(t+\epsilon))} \tag{2.6}$$

2.1.1 Stability vs. solution quality

As we saw in the set visualization example in Chapter 1, optimal solutions are not necessarily stable, since small changes in the input can drastically change the visualization. In the above definities this scenario leads to instability as well: the stability of optimal solutions St(OPT) is influenced by how far apart consecutive solutions are in the solution space, with respect to the distance between the corresponding inputs in the input space. For many optimization problems, the optimal solutions produced by OPT may be very unstable. The instabilities can be the result of OPT changing very fast, albeit continuously, or due to OPT having discontinuities. This suggests an interesting trade-off between the stability of an algorithm and the solution quality: since OPT is unstable, stable solutions cannot have optimal solution quality either.

Although the definitions described above (in Equations 2.2 and 2.5) nicely capture the stability of an algorithm, they do not directly allow a feasible analysis of the trade-off between stability and solution quality: it is hard to compute the stability of an algorithm (as it depends on all input sequences/paths), let alone reason about all possible algorithms. Furthermore, it requires the definition of suitable metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, and it is not always clear how to choose these metrics so that we can obtain meaningful results. Additionally, it is not always clear how to handle optimization problems with continuous input and discrete solutions: when using Equation 2.4 to define stability, any change in solution would result in unbounded stability. However, we would still like to distinguish between solutions that change often from solutions that change rarely, which can be considered more stable.

As we have seen above, the trade-off between stability and solution quality is mainly influenced by the following three aspects: (1) how often the combinatorial structure of a solution changes, (2) whether there are discrete changes in a solution or only continuous changes, and (3) how far apart different solutions are, or how fast one solution can transform into another. In our framework we aim to overcome the drawbacks of the described formulation of stability, by tackling the three aspects influencing the trade-off in isolation. Hence to analyze the trade-off between stability and solution quality, we propose to measure how the solution quality is affected by imposing various requirements on the stability of an algorithm. These requirements are formalized in the following three types of stability.

Event stability follows the setting of kinetic data structures. That is, the input (a set of moving objects) changes continuously as a function over time. However, contrary to typical KDSs where a constraint is imposed on the solution quality, we aim to enforce the stability of the algorithm. For event stability we disallow the algorithm to change the solution too often. Doing so directly is problematic, but we formalize this approach using the concept of k-optimal solutions. We can then obtain a trade-off between stability and quality that can be tuned by the parameter k. Note that event stability captures only *how often* a solution changes, but not *how much* a solution changes at each event.

Topological stability takes a first step towards the definition of stability described in Equation 2.5. However, instead of measuring the amount of change using a metric, we merely require the solution to behave continuously. To do so we need to define only a topology on the solution space $\mathcal{S}$ that captures stable behavior. We work under the assumption that the input follows a continuous path through the input space over time, and thus there is also a topology on the input space defining the stable behavior of the input. Surprisingly, even though we ignore the amount of change in a single time step, this type of analysis still provides meaningful information on the trade-off between solution quality and stability. In fact, the resulting trade-off can be seen as a lower bound for any analysis involving metrics that follow the used topology.

Lipschitz stability fully captures the definition in Equation 2.5. As the name suggests, it is inspired by Lipschitz continuity: if we see an algorithm as a function from input to output, then this function should be Lipschitz continuous. We provide an upper bound on the Lipschitz constant K, which restricts the speed at which the solution can change, relative to the change in the input. This further restricts the stability compared to topological stability, where there is no bound on the speed at which the solution could change. We are then again interested in the quality of the solutions that can be obtained with any K-Lipschitz stable algorithm. Given the complexity of this type of analysis, a complete trade-off for any value of the Lipschitz constant K is typically out of reach, but results for sufficiently small or large values can be of interest.

Lipschitz stability analysis follows the algorithm stability definition introduced earlier and is thus the preferred type of analysis. However, its reliance on metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ and speed parameter K often makes Lipschitz stability analysis prohibitively challenging or infeasible. The other types of stability analysis simplify the stability requirement, making the analysis significantly easier. Specifically, topological stability analysis only relies on topologies on input and solution space, and for event stability analysis we need to count only the number of changes in a solution. Although these types of analysis do not fully capture all aspects of stability, they do offer useful insight into the interplay between problem instances, solutions, and the optimization function, which is invaluable for the development of stable algorithms.

2.1.2 Algorithmic models

When applying either of the above definitions to analyze stability, we should always be aware of the algorithm at hand. Algorithms for time-varying input can adhere to different models, which are differentiated by the availability of the input. The model of an algorithm $\mathcal{A}$ influences the results of the stability analysis. For input $I(t)$ depending on time t, we distinguish the following models.

Stateless algorithms depend only on the input $I(t)$ at a particular point in time, and no other information of earlier times. This in particular means that if $I(t_1) = I(t_2)$, then $\mathcal{A}$ produces the same output at time t_1 and at time t_2. These algorithms are essentially functions from input to output.

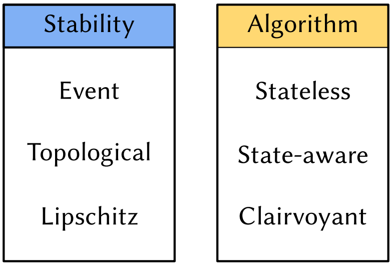


Figure 2.2 Overview of the stability types and algorithmic models defined by our stability framework.

State-aware algorithms have access not only to the input $I(t)$ at a particular time, but also maintain a state over time; in practice this is typically the output at the previous point in time. Thus, even if $I(t_1) = I(t_2)$, then $\mathcal{A}$ may produce different results at time t_1 and t_2 if the states at those times are different.

Clairvoyant algorithms have access to the complete function $I(t)$ and can adapt to future inputs. Thus, the complete output can be computed offline.

While this distinction between types of algorithms resembles characterizations made in other areas of algorithmic research, we chose these names to signify how stability is influenced by the type of algorithm. For example, the state of an algorithm can be seen as knowing its *history*, which is a term used in kinetic data structures to explain how the motion of the input up to a certain point in time influences the current state of the data structure. However, for stability we are interested only in recent history, as we want to ensure that the current solution does not change too quickly. Furthermore, clairvoyant algorithms are often called *offline* algorithms in terms of streaming algorithms and dynamic data, since the complete output over time can be computed offline. However, for stability the clairvoyant aspect of this model is most important: adapting solutions to future inputs to minimize instabilities. In addition, stateless and state-aware algorithms are often called *online* algorithms for dynamic data, or algorithms in the *black-box* model for kinetic data structures, as the complete input over time is not known in advance. While this property indeed influences stability, we choose to make an additional distinction between stateless and state-aware: the stability between these models can differ, as shown in Chapter 4.

Figure 2.2 gives an overview of all parts of the framework. In this chapter we focus only on state-aware algorithms, while in Chapters 3 and 4 we also consider clairvoyant and stateless algorithms. State-aware algorithms are arguably the most interesting of the three models for the following reasons. For many applications, for example online route planning, the input data is not available in full beforehand, rendering clairvoyant algorithms infeasible in such cases. Furthermore, a stateless algorithm cannot utilize the history to ensure stability, which sometimes results is worse stability compared to state-aware algorithms (see Chapter 4 for an example). We conclude this section with a discussion on how to use our framework.

▸ 2.1.3 Applying the framework

As described earlier in this chapter, the most intuitive way to analyze the trade-off between stability and solution quality is as follows. We enforce stability on the solutions to an optimization problem Π, and measure how good the solution quality can still be under this restriction. In our framework, this is done by choosing a type of stability along with a model for the algorithm solving the problem, and analyzing the solution quality that can be achieved. Preferably, we can still use optimal solutions, but in case those are unstable, we want to approximate an optimal solution as well as possible.

In general there are two approaches to find stable solutions with good quality: (1) we can start from an optimal solution OPT and use optimization function f of Π to look for solutions close to OPT that are stable, or (2) we can try to define a structure that is inherently stable according to our chosen type of stability, and analyze its solution quality using optimization function f. For stateless algorithms, the first approach is not a viable strategy: guided by f, a stateless algorithm will always find OPT, since it can access only the data at a particular point in time and use f to look for an optimal solution. Furthermore, using the latter approach, stateless algorithms should always output continuous functions when analyzing topological or Lipschitz stable solutions, as a stateless algorithm is always a function and discontinuities are not allowed for these types of stability.

The stable structures in the second approach are often inherently stateless. An example of this can be found in the stability analysis of the kinetic Euclidean 2-center problem by Durocher and Kirkpatrick [37]. In the kinetic Euclidean 2-center problem, a set of moving points should be covered by two disks of minimum radius. They define reflection-based 2-centers, as a stable alternative to optimal Euclidean 2-centers. To find a reflection-based 2-center, we place a single disk and reflect its position through a reflection center, a central location in the point set. This can be computed on a static input, and hence a stateless algorithm can find the same solution on a time-varying input. To do so, the solution space $\mathcal{S}$ must be restricted to solution space $\mathcal{S}'$, consisting only of stable solutions, such as reflection-based 2-centers. A stateless algorithm can then find a stable optimal solution OPT′ in $\mathcal{S}'$, and ideally OPT′ should approximate the solution quality of OPT. Note that in the other algorithmic models, this stateless computation can be mimicked, since those models are less restrictive in their access to data.

In practice, we often use a combination of the two approaches: the first approach, guided by the optimization function f of Π, requires analysis of optimal and approximate solutions, and hence usually results in the most theoretical insights into the stability of those solutions. Even though the second approach is less straightforward and guided, the (often stateless) nature of this approach can simplify the stability analysis, making it a viable alternative when results are otherwise hard to achieve.

2.2 Kinetic Euclidean minimum spanning tree

A *Euclidean minimum spanning tree* (EMST) on a set of n points in the plane connects all points with a set of edges, such that the total edge length is minimized. The EMST problem is a classic problem in computational geometry, and is strongly related to other geometric structures: the edges of an EMST are also edges of the Gabriel graph, which in turn is a subgraph of a Delaunay triangulation of the same point set. EMSTs can be efficiently computed by first computing a Delaunay triangulation in $O(n \log n)$ time, followed by running a textbook algorithm such as Prim's or Kruskal's algorithm. In the kinetic setting, when the input points are moving continuously through the plane, an EMST can be maintained by growing and shrinking the edges. In certain scenarios, an EMST can undergo discrete changes to its combinatorial structure called *edge flips*. During an edge flip, one of the edges of the EMST is replaced by a different edge, to ensure that the total edge length remains minimal (see Figure 2.3).

Kinetic Euclidean minimum spanning trees and related structures have been studied extensively, albeit mostly in the setting of kinetic data structures. Katoh *et al.* [66] proved an upper bound of $O(n^3 2^{\alpha(n)})$ for the number of external events of d-dimensional EMSTs of n linearly moving points, where $\alpha(n)$ is the inverse Ackermann function. Rahmati *et al.* [89] present a KDS for EMSTs in the plane that processes $O(n^3 \beta^2_{2s+2}(n) \log n)$ events in the worst case, where $\beta_s(n) = \lambda_s(n)/n$ is an extremely slow-growing function. In this formula, $\lambda_s(n)$ represents the maximum length of an order s Davenport-Schinzel sequence [101] on n distinct values, and parameter s measures the complexity of the point trajectories. The best known lower bound for external events of EMSTs in d dimensions is $\Omega(n^d)$ [81]. Approxi-

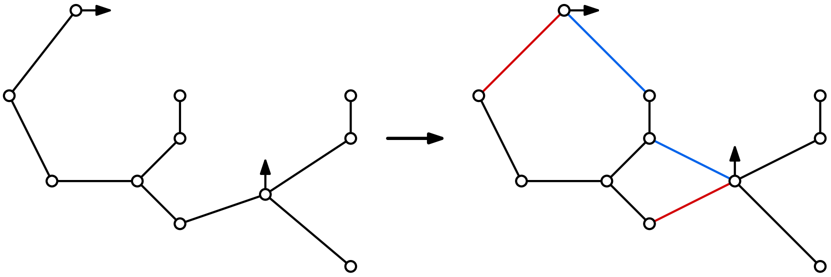


Figure 2.3 The points move to a position where there are multiple optimal EMSTs. As the movement proceeds, the red edges are replaced by blue edges.

mations of EMSTs and related structures such as Delaunay triangulations are also a highly researched topic, but the number of discrete changes still remains at least roughly $\Omega(n^2)$. Since the EMST is a subset of the Delaunay triangulation, we can also consider to kinetically maintain the Delaunay triangulation in a KDS to find a bound on the number of combinatorial changes in an EMST. Fu and Lee [44], and Guibas *et al.* [56] show that the Delaunay triangulation undergoes $O(n^2\lambda_{s+2}(n))$ external events (near-cubic), where $\lambda_s(n)$ is again the maximum length of (n, s)-Davenport-Schinzel sequence [101]. On the other hand, the best known lower bound for external events of the Delaunay triangulation is only $\Omega(n^2)$ [101]. Rubin improves the upper bound to $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, if the number of degenerate events (four points co-circular/three points collinear) is limited [97], or if each point moves along a straight line with unit speed [96]). Agarwal *et al.* [1] also consider a more stable version of the Delaunay triangulation, which undergoes at most a nearly quadratic number of external events. However, external events for EMSTs do not necessarily coincide with external events of the Delaunay triangulation [90]. To further reduce the number of external events, we can consider approximations of the EMST, for example via spanners or well-separated pair decompositions [4]. However, kinetic t-spanners already undergo $\Omega(\frac{n^2}{t^2})$ external events [46].

In the next section, we analyze the event stability of EMSTs. We further reduce the number of external events, as a first restriction on the stability of EMSTs. Using k-optimal EMSTs, we prove that the resulting solution approximates a EMST well.

2.3 Event stability

The least restricting form of stability is event stability. Like the number of external events in KDSs, event stability captures only how often the solution changes.

2.3.1 Event stability analysis

Let Π be an optimization problem with a set of input instances $\mathcal{I}$, a set of solutions $\mathcal{S}$, and optimization function $f : \mathcal{I} \times \mathcal{S} \longrightarrow \mathbb{R}$. Following the framework of kinetic data structures, we assume that the input instances include certain parameters that change as a function of time, such as point coordinates. To apply event stability analysis, we require that all solutions have a combinatorial description, that is, the solution description does not use the time-varying parameters of the input instance. We further require that every solution $S \in \mathcal{S}$ is feasible for every input instance $I \in \mathcal{I}$. Insertions or deletions of elements can break this assumption: a spanning tree on n points is not a feasible solution for $n + 1$ point, since one edge is missing. Note that an insertion or a deletion would typically force an event, and a kinetic data structure would be allowed to recompute. Thus it makes sense to apply our event stability analysis only between such insertions and deletions.

For example, in the setting of kinetic EMSTs, the input instances would consist of a fixed set of points. The coordinates of these points then change as a function over time. A solution of the kinetic EMST problem consists of the combinatorial description of a tree on the set of input points. Note that every tree describes a feasible solution for any input instance, if we do not insist on any additional restrictions like, e.g., planarity. The minimization function f then simply measures the total length of the tree, which does depend on the time-varying parameters of the problem instance.

We want to restrict how often a solution changes, in such a way that the solution is still close to an optimal solution in solution quality. Instead of doing so directly, we introduce the concept of k-optimal solutions. Let $d_{\mathcal{I}}$ be a metric on the input instances, and let $\mathrm{OPT} : \mathcal{I} \longrightarrow \mathcal{S}$ describe the optimal solutions. We say that a solution $S \in \mathcal{S}$ is *k-optimal* for an instance $I \in \mathcal{I}$ if there exists an input instance $I' \in \mathcal{I}$ such that $f(I', S) = f(I', OPT(I'))$ and $d_{\mathcal{I}}(I, I') \leq k$. Any optimal solution is therefore always 0-optimal. We need to point out that the above definition requires a form of normalization on the metric $d_{\mathcal{I}}$, similar to that of, e.g., smoothed analysis [103]. We therefore require that there exists a constant c such that every solution $S \in \mathcal{S}$ is c-optimal for every instance $I \in \mathcal{I}$. For technical reasons we require the latter condition to hold only for some time interval $[0, T]$ of interest. Note that the concept of k-optimal solutions is closely related to *backward error analysis* in numerical analysis [117].

Following the framework of kinetic data structures, we typically require the functions of the time-varying parameters to be well-behaved (e.g., polynomial functions), for otherwise we cannot derive meaningful bounds. The event stability analysis then considers two aspects. First, we analyze how often the solution needs to change to maintain a k-optimal solution for every point in time. Second, we analyze how well a k-optimal solution approximates an optimal solution. Intuitively, k-optimal solutions approximate optimal solutions well, as the input did not change much since the solution was actually optimal. We can then enforce stability by ignoring all events that happen, required to keep the solution optimal, as long as a solution stays k-optimal. Once the solution is no longer k-optimal, we recompute to find the optimal solution, and repeat the process. This reduces the number of events, while the solution quality becomes only moderately worse. Typically we are not able to directly obtain good bounds on the approximation ratio, but given certain reasonable assumptions, good approximation bounds as a function of k can be provided.

2.3.2 Event stability for EMSTs

Our input consists of a set of points $P = \{p_1, \ldots, p_n\}$ where each point p_i describes a trajectory by the function $p_i : [0, T] \longrightarrow \mathbb{R}^d$. The goal is to maintain a combinatorial description of a short spanning tree on P that does not change often. We assume that the functions x_i are polynomials with bounded degree s.

To use the concept of k-optimal solutions, we first need to normalize the coordinates. We simply assume that $p_i(t) \in [0, 1]^d$ for $t \in [0, T]$. This assumption may seem overly restrictive for kinetic point sets, but note that we are interested only in relative positions, and thus the frame of reference may move with the points. Next, we define the metric $d_{\mathcal{I}}$ along the trajectory of all points as follows.

$$d_{\mathcal{I}}(I, I') = \max_i \|p_i - p'_i\| \tag{2.7}$$

Note that this metric, and the resulting definition of k-optimal solutions, is not specific to EMSTs and can be used in general for problems with kinetic point sets as input. In our case $\|a - b\|$ denotes the distance between a and b in the (Euclidean) ℓ_2 norm. Now let $S(t) = OPT(t)$ be the EMST at time t. Then, by definition, $S(t)$ is k-optimal at time t' if $d_{\mathcal{I}}(I(t), I(t')) \leq k$. As explained before, our approach is now very simple: we compute the EMST and keep that solution as long as it is k-optimal, after which we compute the new EMST, and so forth. Below we analyze this approach.

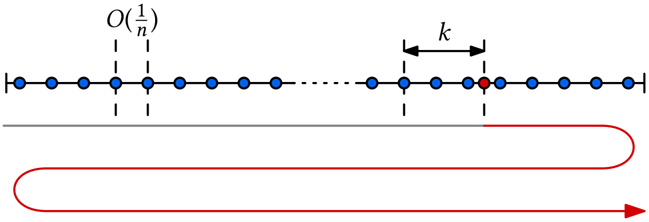


Figure 2.4 The blue points are stationary, while the red point moves along a trajectory of degree 3, triggering $\Omega(\frac{s}{k})$ events. The trajectory is shown as an arrow along the 1D space; the gray part has already been traversed.

Number of events To bound the number of events, we first need to bound the speed of any point with a polynomial trajectory and bounded coordinates. For this we can use a classic result known as the *Markov Brothers' inequality.*

2.3.1 Lemma ([75]). *Let* $h(t)$ *be a polynomial with degree at most* s *such that* $h(t) \in [0, 1]$ *for* $t \in [0, T]$, *then* $|\mathrm{d}h(t)/\mathrm{d}t| \le s^2/T$ *for all* $t \in [0, T]$.

2.3.2 Lemma. *For a kinetic point set* P *with degree-*s *polynomial trajectories* $p_i(t) \in [0, 1]^d$ *(*$t \in [0, T]$*) we need* $O(\frac{s^2}{k})$ *changes to maintain a* k*-optimal solution for constant* d.

Proof. By Lemma 2.3.1 the velocity of any point is at most s^2/T in one dimension, and thus at most $\sqrt{d}\, s^2/T = O(s^2/T)$ in d dimensions, assuming d is constant. Now assume that we have computed an optimal solution S for some time t. The solution S remains k-optimal until one of the points has moved at least k units. Since the velocity of the points is bounded, this takes at least $\Delta t = \Omega(kT/s^2)$ time, at which point we can recompute the optimal solution. Since the total time interval is of length T, this can happen at most $T/\Delta t = O(s^2/k)$ times. □

Next we show that this upper bound is tight up to a factor of s, using *Chebyshev polynomials* of the first kind [94]. A Chebyshev polynomial of degree s with range $[0, 1]$ and domain $[0, T]$ passes through the entire range exactly s times.

2.3.3 Lemma. *For a kinetic point set* P *of* n *points with degree-*s *polynomial trajectories* $p_i(t) \in [0, 1]^d$ *(*$t \in [0, T]$*) we need* $\Omega(\min(\frac{s}{k}, sn))$ *changes in the worst case to maintain a* k*-optimal solution.*

Proof. We can restrict ourselves to $d = 1$. Let p_1 move along a Chebyshev polynomial of degree s, and let the remaining points be stationary and placed equidistantly along the interval $[0, 1]$. As soon as p_1 meets one of the other points, then p_1 can travel at most k units before the solution is no longer k-optimal (see Figure 2.4). Therefore, p_1 moving through the entire interval requires $\Omega(\min(1/k, n))$ changes to the solution. Doing so s times gives the desired bound. □

It is important to notice that this behavior is fairly specific for polynomial trajectories. If we allow more general trajectories, then we can prove a stronger bound.

2.3.4 Lemma. *For a kinetic point set P of n points with degree-s pseudo-algebraic trajectories $p_i(t) \in [0,1]^d$ ($t \in [0,T]$) we need $\Omega(\min(\frac{sn}{k}, sn^2))$ changes in the worst case to maintain a k-optimal solution.*

Proof. We can restrict ourselves to $d = 1$. Any two pseudo-algebraic trajectories of degree at most s can cross each other at most s times. We make $n/2$ points stationary and place them equidistantly along the interval $[0,1]$. The other $n/2$ points follow trajectories that take them through the entire interval s times, in such a way that every point moves through the entire interval completely before another point does so. The resulting trajectories are clearly pseudo-algebraic, and each time a point moves through the entire interval it requires $\Omega(\min(1/k, n))$ changes to the solution. As a result, the total number of changes is $\Omega(\min(\frac{sn}{k}, sn^2))$. □

We can show the same lower bound for algebraic trajectories of degree at most s, but this is slightly more involved.

2.3.5 Lemma. *For a kinetic point set P of n points with degree-s algebraic trajectories $p_i(t) \in [0,1]^d$ ($t \in [0,T]$) we need $\Omega(\min(\frac{sn}{k}, sn^2))$ changes in the worst case to maintain a k-optimal solution.*

Proof. We can restrict ourselves to $d = 1$. We make $n/2$ points stationary and place them equidistantly along the interval $[0,1]$. The other $n/2$ points follow trajectories that take them through the entire interval $s/4$ times, in such a way that every point moves through the entire interval completely before another point does so. The trajectory of a non-stationary point is $p_i(t) = \sum_{j=0}^{s/4} \frac{1}{(t-10\cdot j-10\cdot i\cdot s/4)^4+1}$. The trajectory consists of $s/4$ moves through the stationary points, one such move every 10 time units (see Figure 2.5). The i-th point will be finished $10 * s/4$ time units after it starts its first move through the stationary points, while the $(i+1)$-st point starts 10 units after the i-th point finishes. The resulting trajectories are clearly algebraic, and each time a point moves through the entire interval it requires $\Omega(\min(1/k, n))$ changes to the solution. As a result, the total number of changes is $\Omega(\min(\frac{sn}{k}, sn^2))$. □

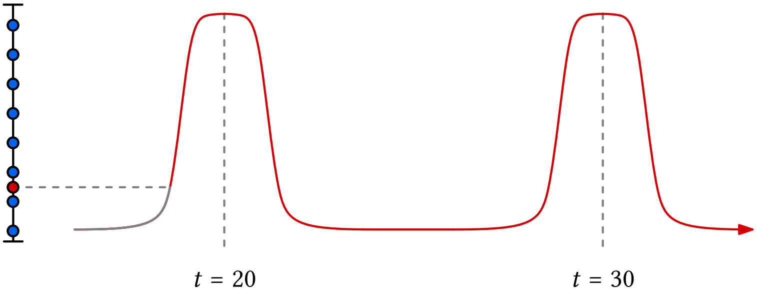


Figure 2.5 The blue points are stationary, while the red point moves along a trajectory of degree 8 and is the second point to start moving through the blue points ($p_1(t) = \sum_{j=0}^{2} \frac{1}{(t-10\cdot j-20)^4+1}$). The trajectory is shown as an arrow along the 1D space; the gray part has already been traversed.

Solution quality To analyze the solution quality of k-optimal solutions, we prove a bound on the ratio between the length of k-optimal solutions and the length of optimal EMSTs. In general, we cannot expect k-optimal solutions to b a good approximation of an optimal EMST's length: if all points are within distance k from each other, then all solutions are k-optimal. We therefore need to make the assumption that the points are spread out reasonably throughout the motion. To quantify this, we use a measure inspired by the *order-l spread*, as defined by Erickson [38]. Let $\textsc{mindist}_l(P)$ be the smallest distance in P between a point and its l-th nearest neighbor. We assume that $\textsc{mindist}_l(P) \geq 1/\Delta_l$ throughout the motion, for some value of Δ_l. We can use this assumption to give a lower bound on the length of the EMST. Pick an arbitrary point and remove all points from P that are within distance $1/\Delta_l$, and repeat this process until the smallest distance is at least $1/\Delta_l$. By our assumption, we remove at most $l-1$ points for each chosen point, so at least n/l points are left. The distance between each pair of points is now at least $1/\Delta_l$, hence an EMST on the remaining points has length $\Omega(\frac{n}{l\Delta_l})$. This readily forms a lower bound on the length of the EMST on P: even if all removed points were Steiner points, adding back those points can improve the length of an EMST only by a constant factor [28].

2.3.6 Lemma. *A k-optimal solution of the EMST problem on a set of n points P is an $O(1+kl\Delta_l)$-approximation of the EMST, under the assumption that $\textsc{mindist}_l(P) \geq 1/\Delta_l$.*

Proof. Let S be a k-optimal solution of P and let OPT be an optimal solution of P. By definition there is a point set P', with $d_{\mathcal{I}}(P,P') \leq k$, for which the length of solution S is at most that of the optimal solution OPT′ of P'. Since $d_{\mathcal{I}}(P,P') \leq k$, the length of each edge can grow or shrink by at most $2k$ when moving from P' to P. Therefore we can state that $f(P,S) \leq f(P,\mathrm{OPT}) + 4kn$, as in the worst case every edge in the k-optimal solution can have grown by $2k$ while in an optimal solution every edge can have shrunk by $2k$. Now, using the lower bound of $\Omega(\frac{n}{l\Delta_l})$ on the length of an EMST, we obtain the following.

$$\begin{aligned} f(P,\mathrm{OPT}) + 4kn &\leq f(P,\mathrm{OPT}) + 4kO(f(P,\mathrm{OPT})l\Delta_l) \\ &= O(1 + kl\Delta_l) \cdot f(P,\mathrm{OPT}) \end{aligned} \qquad \square$$

Note that there is a clear trade-off between the approximation ratio and how restrictive the assumption on the spread is. Regardless, we can obtain a decent approximation, while processing only a small number of events. Choosing reasonable values $k = O(1/n)$, $l = O(1)$, and $\Delta_l = O(n)$, then, under the assumptions, a constant-factor approximation of the EMST can be maintained while processing only $O(n)$ events.

▸ 2.4 Topological stability

The event stability analysis has two major drawbacks: (1) it is applicable only to problems for which the solutions are always feasible and described combinatorially, and (2) it does not distinguish between small and large structural changes. Topological stability analysis is applicable to a wide variety of problems and enforces continuous changes to the solution.

▸ 2.4.1 Topological stability analysis

Let Π be an optimization problem with input instances $\mathcal{I}$, solutions $\mathcal{S}$, and optimization function f. An algorithm $\mathcal{A}$ is *topologically stable* if, for any continuous path $I : [0,1] \longrightarrow \mathcal{I}$ in $\mathcal{I}$, $\mathcal{A}$ maps it to a continuous path S in $\mathcal{S}$. To properly define a continuous path in $\mathcal{I}$ and $\mathcal{S}$, we need to specify a topology $\mathcal{T}_\mathcal{I}$ on $\mathcal{I}$ and a topology $\mathcal{T}_\mathcal{S}$ on $\mathcal{S}$. An overview of this model can be found in Figure 2.6. Alternatively, we could specify metrics $d_\mathcal{I}$ and $d_\mathcal{S}$, but this is typically more involved. Let $\mathcal{P}_\mathcal{I}$ be the set of continuous paths through $\mathcal{I}$. We then want to analyze the approximation ratio ρ_{TS} of any topologically stable algorithm with respect to OPT, which we will call the *topological stability ratio*. That is, we are interested in the ratio

$$\rho_{\mathrm{TS}}(\Pi, \mathcal{T}_\mathcal{I}, \mathcal{T}_\mathcal{S}) = \inf_{\mathcal{A}} \sup_{I \in \mathcal{P}_\mathcal{I}} \sup_{t \in [0,1]} \frac{f(I(t), S(t))}{f(I(t), \mathrm{OPT}(I(t)))} \tag{2.8}$$

where the infimum is taken over all topologically stable algorithms. Naturally, if OPT is already topologically stable, then this type of analysis does not provide any insight

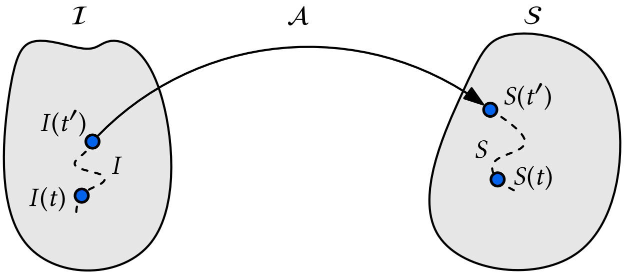


Figure 2.6 Algorithm $\mathcal{A}$ maps input instances from the input space $\mathcal{I}$ to the solution space $\mathcal{S}$. Continuous path I in the space defined by topology $\mathcal{T}_{\mathcal{I}}$, is mapped to continuous path S in the space defined by topology $\mathcal{T}_{\mathcal{S}}$.

and the ratio is simply 1. However, OPT is not topologically stable if it undergoes discrete changes, and in that case topological stability allows you to measure what solution quality can be achieved by requiring continuity.

The above analysis can even be applied when the solution space (or the input space) is discrete. In such cases, continuity can often be defined using the graph topology of so-called flip graphs, for example, based on edge flips for triangulations or rotations in rooted binary trees. The vertices of such a graph each represent a solution (or input) with a different combinatorial structure, while the edges represent the possible transitions between solutions. To create a continuous solution space, we still represent the discrete space using the vertices of the flip graph, but we create continuity on the edges: we define a (continuous) topological space by representing vertices by points, and representing every edge of the graph by a copy of the unit interval $[0, 1]$. These intervals are glued together at the vertices. In other words, we consider the corresponding simplicial 1-complex.

For EMSTs we can do exactly what we just described, since the solution space is discrete and the vertices of a flip graph represent spanning trees. Although the points in the interior of the edges of this topological space do not represent proper spanning trees, we can still use this topological space in Equation 2.8 by extending f over the edges via linear interpolation. This ensures that the value of f for one of the vertices incident to an edge is as least as high as the value of f anywhere on the edge. We therefore need to consider only the vertices of the flip graph (which represent proper spanning trees) to compute the topological stability ratio. Figure 2.7 shows an example of such a topological space for spanning trees.

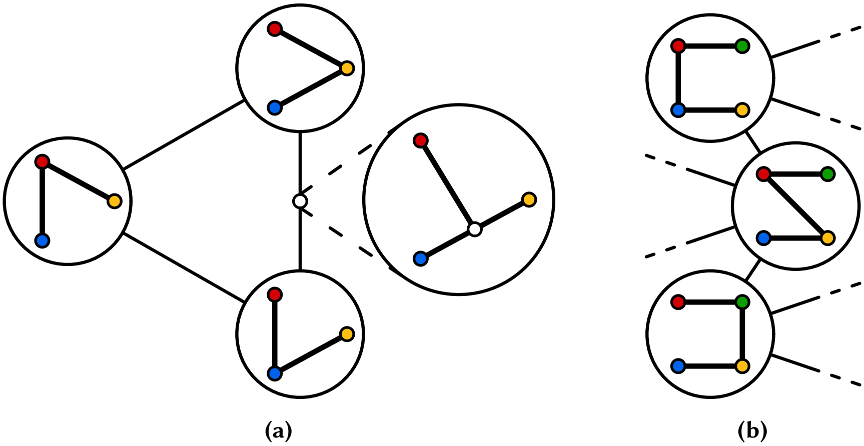


Figure 2.7 Topological spaces defined by flip graphs for edge slides/rotations. **(a)** The complete solution space for three vertices, along with an intermediate solution. **(b)** A partially drawn solution space for four vertices.

Proving bounds on ρ_{TS} To prove an upper bound on the topological stability ratio, we have to describe a state-aware algorithm that produces a topologically stable solution. If this algorithm computes an r-approximation of the optimal solution, then we have found an upper bound of r on ρ_{TS}. While such an algorithm works on time-varying data, we usually consider static inputs that allow multiple optimal solutions, when proving an upper bound. The optimal solution would undergo a discrete change when such an input was encountered during motion. We define a continuous transformation that works for any such static input, and transforms one optimal solution of the static input into another. Note that the continuous transformation should follow the chosen topology $\mathcal{T}_{\mathcal{S}}$. If during this transformation, the solution is at most a factor r worse than OPT according to f, then we immediately obtain an upper bound of r on ρ_{TS}: an algorithm can wait until a discrete change happens and then apply the transformation to produce an r-approximation. This approximation is topologically stable, since the transformation is continuous, and topological stability does not bound the speed at which the solution can change. Hence, at the point in time where the discrete change would happen, the algorithm may "freeze time" and apply the transformation to swap between optimal solutions.

For a lower bound on the topological stability ratio, we should consider a full time-varying input, for which an approximation ratio of r is always necessary. However, we can again use static inputs that allow multiple optimal solutions, to simplify the analysis. We first construct a static input I, where every continuous transformation from one optimal solution to another, requires a solution that is at least a factor r worse than OPT. Thus, any algorithm computing a continuous transformation for I produces at least an r-approximation. Finally, we find a motion of the input points in which I occurs at some time t, and this motion should force the continuous transformation to happen somewhere during the motion. This is achieved by ensuring that keeping the same solution during the complete motion, or attempting a continuous transformation before/after t results in a solution that is even worse. The best any algorithm can do, is transforming exactly at t, but this requires a solution that is at least a factor r worse than OPT. Thus every topologically stable algorithm computes at least an r-approximation on this time-varying input: ρ_{TS} is lower bounded by r.

Note that since we constructed a state-aware algorithm to prove an upper bound on the topological stability ratio ρ_{TS}, we also prove a bound on the topological stability ratio of clairvoyant algorithms. A clairvoyant algorithm can simply emulate a state-aware algorithm, by only looking ahead in time. On the other hand, the method we describe to prove a lower bound on ρ_{TS} for state-aware algorithms, shows that *every* topologically stable algorithm requires an r-approximation. Thus we prove a stronger statement than required: even a clairvoyant algorithm cannot do better.

2.4.2 Topological stability of EMSTs

We use the same setting of the kinetic EMST problem as in Section 2.3.2, except that we do not restrict the trajectories of the points and we do not normalize the coordinates. We merely require that the trajectories are continuous. To define this properly, we need to define a topology on the input space, but for a kinetic point set with n points in d dimensions we can simply use the standard topology on $\mathbb{R}^{dn}$ as $\mathcal{T}_{\mathcal{I}}$. To apply topological stability analysis, we also need to specify a topology on the (discrete) solution space. As the points move, the minimum spanning tree may have to change at some point in time by removing one edge and inserting another edge. We do not consider this operation to be stable, since the edge can reinserted anywhere in the tree. Instead we define the topology of $\mathcal{S}$ using a flip graph, where the operations are either *edge slides* or *edge rotations* [3, 50, 82] (see Figure 2.7). The optimization function f, measuring the quality of the EMST, is naturally defined for the vertices of the flip graph as the length of the spanning tree, and we use linear interpolation to define f on the edges of the flip graph. For edge slides and rotations we provide upper and lower bounds on $\rho_{\text{TS}}(\text{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}})$.

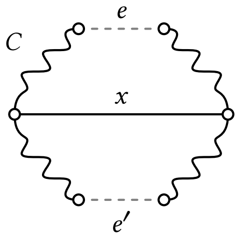


Figure 2.8 A configuration where x is the longest edge when sliding from e to e'.

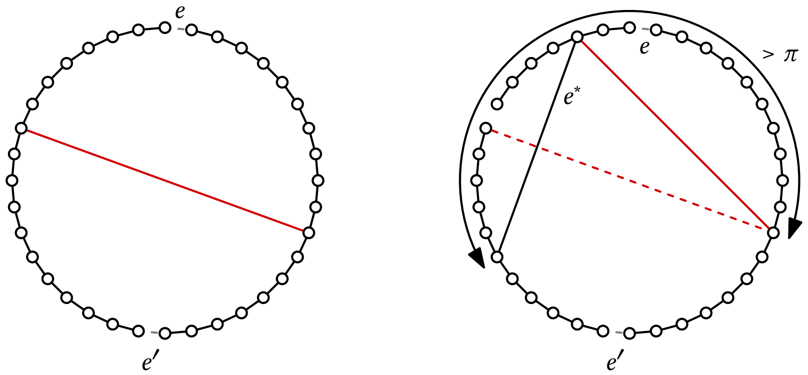
Edge slides An edge slide is defined as the operation of moving one endpoint of an edge to one of its neighboring vertices along the edge to that neighbor. More formally, an edge (u, v) in the tree can be replaced by (u, w) if w is a neighbor of v and $w \neq u$. Since this operation is very local, we consider it to be stable. Note that a tree stays connected after edge slides.

2.4.1 Lemma. *If $\mathcal{T}_S$ is defined by edge slides, then $\rho_{\text{TS}}(EMST, \mathcal{T}_I, \mathcal{T}_S) \leq \frac{3}{2}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now slide edge e to edge e' by sliding its endpoints along the edges of C. Let x be the longest intermediate edge when sliding from e to e' (see Figure 2.8). To allow x to be as long as possible with respect to the length of the EMST, the EMST should be fully contained in C. By the triangle inequality we get that $2|x| \leq |C|$. Since the length of the EMST is $\text{OPT} = |C| - |e|$, we get that $|x| \leq \text{OPT}/2 + |e|/2$. Thus, the length of the intermediate tree is $|C| - 2|e| + |x| = \text{OPT} - |e| + |x| \leq \frac{3}{2}\,\text{OPT}$. □

2.4.2 Lemma. *If $\mathcal{T}_S$ is defined by edge slides, then $\rho_{\text{TS}}(EMST, \mathcal{T}_I, \mathcal{T}_S) \geq \frac{\pi+1}{\pi} \approx 1.318$.*

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e| = |e'|$ is very small. Let the remaining points be arranged in a circle with diameter d, as shown in Figure 2.9a. Furthermore, let OPT be the length of the EMST, then we get that $\text{OPT} < d\pi$, since OPT cannot form a cycle around the circle. Simply using edge slides to move e toward e' will always grow e to be length at least $d - \varepsilon$. We can make this construction for any $\varepsilon > 0$ by using sufficiently many points around the circle and consequently making e and e' arbitrarily short. Alternatively, e can take a shortcut by sliding over another edge e^* as a chord (see Figure 2.9b). Doing so would require e^* to first slide into this position. The shortcut is only beneficial if $|e| + |e^*| < d - \varepsilon$. However, if e^*



(a) Edge e slides to e' and becomes the diameter of the circular configuration.

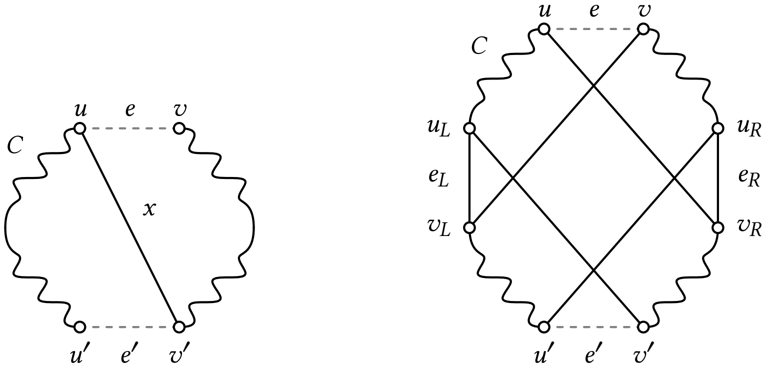
(b) Sliding edge e^* to form a chord creates an even longer spanning tree.

Figure 2.9 This configuration is a $(\frac{\pi+1}{\pi} - \varepsilon)$-approximation of the EMST.

helps e to avoid becoming a diameter of the circle, then e and e^*, as chords, must span an angle larger than π together. As a result, for a circle of diameter d, we get $|e| + |e^*| \geq d > d - \varepsilon$ by triangle inequality.

A motion of the points that forces e to slide to e' in this particular configuration looks as follows. The points start at e and move at constant speed along the circle, half of the points clockwise and the other half counter clockwise. The speeds are assigned in such a way that at some point all points are evenly spread along the circle. Once all points are evenly spread, they start moving towards e', again along the circle. Any edge sliding from e to e' during the motion must have length $d - \varepsilon$ at some point throughout the motion. On the other hand, OPT is largest when the points are evenly spread along the circle. Let the circle have diameter d, then OPT has length at most $d\pi - |e|$, and equivalently $d \geq \text{OPT}/\pi + |e|$. Since we argued that the sliding edge will always have length $d - \varepsilon$ at some point, the largest intermediate solution has length at most $\text{OPT} - |e| + d - \varepsilon$. Thus, for any small constant $\varepsilon > 0$, we show that $\rho_{\text{TS}}(\text{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{\text{OPT} - |e| + d - \varepsilon}{\text{OPT}} \geq \frac{\text{OPT} + \text{OPT}/\pi - \varepsilon}{\text{OPT}} \geq \frac{\pi+1}{\pi} - \varepsilon \approx 1.318 - \varepsilon$. □

Edge rotations Edge rotations are a generalization of edge slides, that allow one endpoint of an edge to move to any other vertex. These operations are clearly not as stable as edge slides, but they are still more stable than the deletion and insertion of arbitrary edges.



(a) If one part of C is small enough, then we can rotate one endpoint of e directly to one endpoint of e'.

(b) Potential intermediate edges when rotating edge e to e' using two edge rotations.

Figure 2.10 Potential intermediate edges when rotating e to e'.

2.4.3 Lemma. *If $\mathcal{T}_S$ is defined by edge rotations, then $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_I, \mathcal{T}_S) \leq \frac{4}{3}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge $e = (u, v)$ and inserting an edge $e' = (u', v')$, where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now rotate edge e to edge e' along some of the vertices of C. Let x be the longest intermediate edge when optimally rotating from e to e'. To allow x to be as long as possible with respect to the length of the EMST, the EMST should be fully contained in C. We argue that $|x| \leq \mathrm{OPT}/3 + |e|$, where OPT is the length of the EMST. Removing e and e' from C splits C into two parts, where we assume that u and u' (v and v') are in the left (right) part. First assume without loss of generality that the left part has length at most OPT/3. Then we can rotate e to (u, v'), and then to e', which implies that $|x| = |(u, v')| \leq \mathrm{OPT}/3 + |e|$ by the triangle inequality (see Figure 2.10(a)).

Now assume that both parts have length at least OPT/3. Let $e_L = (u_L, v_L)$ be the edge in the left part that contains the midpoint of that part, and let $e_R = (u_R, v_R)$ be the edge in the right part that contains the midpoint of that part, where u_L and u_R are closest to e (see Figure 2.10(b)). Furthermore, let Z be the length of $C \setminus \{e, e', e_L, e_R\}$. Now consider the potential edges (u, v_R), (v, v_L), (u', u_R), and (v', u_L). By the triangle inequality, the sum of the lengths of these edges is at most $4|e| + 2|e_L| + 2|e_R| + Z$. Thus, one of these potential edges has length at most $|e| + |e_L|/2 + |e_R|/2 + Z/4$. Without loss of generality let (u, v_R) be that edge (the construction is fully symmetric). We can

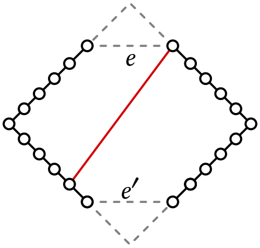


Figure 2.11 Lower bound construction for edge rotations.

now rotate e to (u, v_R), then to (u', v_R), and finally to e'. As each part of C has length at most $2\,\mathrm{OPT}/3$, we get that $|(u', v_R)| \leq \mathrm{OPT}/3 + |e|$ by construction. Furthermore we have that $\mathrm{OPT} = |e| + |e_L| + |e_R| + Z$. Thus, $|(u, v_R)| \leq |e| + |e_L|/2 + |e_R|/2 + Z/4 = \mathrm{OPT}/3 + 2|e|/3 + |e_L|/6 + |e_R|/6 - Z/12$. Since e needs to be removed to update the EMST, it must be the longest edge in C. Therefore $|(u, v_R)| \leq \mathrm{OPT}/3 + |e|$, which shows that $|x| \leq \mathrm{OPT}/3 + |e|$. Since the length of the intermediate tree is $\mathrm{OPT} - |e| + |x| \leq \frac{4}{3}\,\mathrm{OPT}$, we obtain that $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T_I}, \mathcal{T_S}) \leq \frac{4}{3}$. □

2.4.4 Lemma. *If $\mathcal{T_S}$ is defined by edge rotations, then, $\rho_{\mathrm{TS}}(EMST, \mathcal{T_I}, \mathcal{T_S}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}} \approx 1.162$.*

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e'. Let the remaining points be arranged in a diamond shape as shown in Figure 2.11, where the side length of the diamond is 2, and $|e| = |e'| = 1$. As a result, the distance between an endpoint of e and the left or right corner of the diamond is $2 - \frac{1}{2}\sqrt{2}$. Now we define a *top-connector* as an edge that intersects the vertical diagonal of the diamond, but is completely above the horizontal diagonal of the diamond. A *bottom-connector* is defined analogously, but must be completely below the horizontal diagonal. Finally, a *cross-connector* is an edge that crosses or touches both diagonals of the diamond. Note that a cross-connector has length at least 2, and a top- or bottom-connector has length at least $|e| = 1$. In the considered update, we start with a top-connector and end with a bottom-connector. Since we cannot rotate from a top-connector to a bottom-connector in one step, we must reach a state that either has both a top-connector and a bottom-connector, or a single cross-connector. In both options the length of the spanning tree is equal to four times $2 - \frac{1}{2}\sqrt{2}$, plus the connector(s) of length at least 2. This gives a total length of $10 - 2\sqrt{2}$, while the minimum spanning tree has no cross-connector, but a single top- or bottom-connector and hence a total length of $9 - 2\sqrt{2}$.

To force the update from e to e' in this configuration, we can use the following motion. The points start at the endpoints of e and move with constant speeds to a position where the points are evenly spread around the left and right corner of the diamond. Then the points move with constant speeds to the endpoints of e'. The argument above still implies that we need edges of total length at least 2 intersecting the vertical diagonal of the diamond at some point during the motion. On the other hand, $\mathrm{OPT} \leq 9 - 2\sqrt{2}$ throughout the motion. Thus $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}} \approx 1.162$. □

2.5 Lipschitz stability

The major drawback of topological stability analysis is that it still does not fully capture stable behavior; the algorithm must be continuous, but we can still make many changes to the solution in an arbitrarily small time frame. In Lipschitz stability analysis we additionally limit how fast the solution can change.

2.5.1 Lipschitz stability analysis

To formally define Lipschitz stability, we return to the continuous setting in Section 2.1. Let Π be an optimization problem with input instances from the set $\mathcal{I}$, solutions from the set $\mathcal{S}$, and optimization function f. An input $I : [0,1] \longrightarrow \mathcal{I}$ is a continuous path through input space $\mathcal{I}$ and an algorithm $\mathcal{A}$ maps $I(t) \in \mathcal{I}$ to a solution $S(t) \in \mathcal{S}$ for each time $t \in [0,1]$. To quantify how fast a solution changes as the input changes, we need to specify metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ on $\mathcal{I}$ and $\mathcal{S}$, respectively. An algorithm $\mathcal{A}$ is *K-Lipschitz stable* if $\mathrm{St}(\mathcal{A}) \leq K$, where $\mathrm{St}(\mathcal{A})$ is defined as in Equations 2.4 and 2.5. For an output S of a K-Lipschitz stable algorithm, this means that we bound how quickly S can change relative to input I. In particular, if we consider two times $t, t' \in [0,1]$, then for this output S it holds that $\Delta_{\mathcal{S}}(S(t), S(t')) \leq K\Delta_{\mathcal{I}}(I(t), I(t'))$, where $\Delta_{\mathcal{S}}$ and $\Delta_{\mathcal{I}}$ measure the distance traveled in solution and input space, respectively. Figure 2.12 gives an overview the above.

Note that by defining K-Lipschitz stability using Equation 2.4, we require that, for bounded K, a solution S does not change unless input I also changes. However, we can instead use the definition of stability in Equation 2.6. Using that definition with the assumption that the input changes with at most unit speed (e.g., points moving with unit speed), allows the solution S to change with speed at most K. This assumption is common in computational geometry, and we use it in Chapter 4 to perform K-Lipschitz stability analysis.

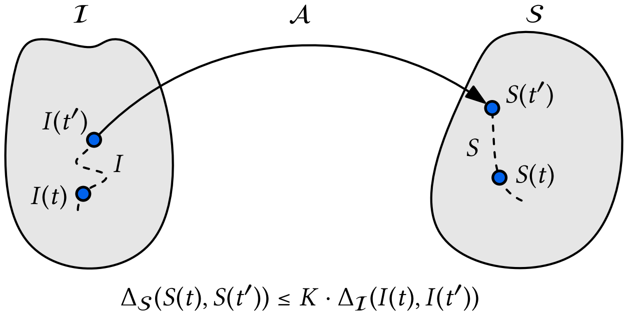

$\mathcal{I}$
$\mathcal{A}$
$\mathcal{S}$
$I(t')$
I
$I(t)$
$S(t')$
S
$S(t)$
$\Delta_{\mathcal{S}}(S(t), S(t')) \leq K \cdot \Delta_{\mathcal{I}}(I(t), I(t'))$

Figure 2.12 Algorithm $\mathcal{A}$ maps input instances from input space $\mathcal{I}$ to solution space $\mathcal{S}$. The metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ allow us measure the distance traveled in input ($\Delta_{\mathcal{I}}$) and output ($\Delta_{\mathcal{S}}$).

Let $\mathcal{P}_{\mathcal{I}}$ be the set of continuous paths through $\mathcal{I}$. We are again interested in the approximation ratio ρ_{LS} of any K-Lipschitz stable algorithm with respect to OPT. We call this ratio the *Lipschitz stability ratio*, which is defined as

$$\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}}) = \inf_{\mathcal{A}} \sup_{I \in \mathcal{P}_{\mathcal{I}}} \sup_{t \in [0,1]} \frac{f(I(t), S(t))}{f(I(t), \mathrm{OPT}(I(t)))} \tag{2.9}$$

where the infimum is taken over all K-Lipschitz stable algorithms. It is easy to see that $\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}})$ is lower bounded by $\rho_{\mathrm{TS}}(\Pi, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}})$ for the corresponding topologies $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{S}}$ of $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, respectively.

As mentioned in Section 2.1, analyses of this type are often difficult. First, we need to be very careful when choosing metrics the $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, as they should behave similarly with respect to scale. For example, let the input consist of a set of points in the plane and let cI for $I \in \mathcal{I}$ be the instance obtained by scaling all coordinates of the points in I by the factor c. Now $d_{\mathcal{I}}$ depends linearly on scale, that is $d_{\mathcal{I}}(cI, cI') \sim c d_{\mathcal{I}}(I, I')$. Moreover, assume that $d_{\mathcal{S}}$ is independent of scale. If we now scale the input instance, we change the relative speed between changes in the input and changes in the solution. Thus without scale invariance, the analysis is rendered meaningless.

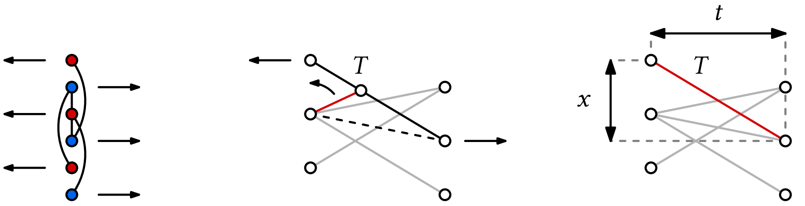
Second, we need to be careful with discrete solution spaces. Using the flip graphs as mentioned in Section 2.4 we can extend a discrete solution space to a continuous space by including the edges. However, it is not always straightforward to extend $d_{\mathcal{S}}$ over the edges of the flip graph: the distance between (combinatorial) solutions, which are represented by vertices in the flip graph, and between intermediate solutions, on the edges of the flip graph, can change as the input moves. In Section 2.5.2, we show how to overcome this problem for EMSTs, by carefully defining $d_{\mathcal{S}}$.

Typically we expect it to be hard to fully describe $\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}})$ as a function of K. However, it may be possible to obtain interesting results for certain values of K. One value of interest is the value of K for which the approximation ratio equals or approaches the approximation ratio of the corresponding topological stability analysis. Another potential value of interest is the value of K below which any K-Lipschitz stable algorithm performs asymptotically as poorly as a constant algorithm, always computing the same solution regardless of instance.

2.5.2 Lipschitz stability of EMSTs

We use the same setting of the kinetic EMST problem as in Section 2.4.2, except that, instead of topologies, we specify metrics for $\mathcal{I}$ and $\mathcal{S}$. For $d_{\mathcal{I}}$ we can simply use the metric in Equation 2.7, which implies that points move with a bounded speed. For $d_{\mathcal{S}}$ we use a metric inspired by the edge slides of Section 2.4.2. To that end, we need to define how long a particular edge slide takes, or equivalently, how "far" an edge slide is. To make sure that $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ behave similarly with respect to scale, we let $d_{\mathcal{S}}$ measure the distance the sliding endpoint has traveled during an edge slide. However, this creates an interesting problem: the edge on which the endpoint is sliding may be moving and stretching/shrinking during the operation. This influences how long it takes to perform the edge slide. We need to be more specific to make this approach work: we define that (1) as the points are moving, the relative position (between 0 and 1 from starting endpoint to finishing endpoint) of a sliding endpoint is maintained without cost in $d_{\mathcal{S}}$, and (2) $d_{\mathcal{S}}$ measures the difference in relative position multiplied by the length $L(t)$ of the edge on which the endpoint is sliding. More tangibly, an edge slide performed by a K-Lipschitz stable algorithm can be performed in t^* time such that $\int_0^{t^*} \frac{K}{L(t)} \mathrm{d}t = 1$, where $L(t)$ describes the length of the edge on which the endpoint slides as a function of time. Finally, the optimization function f simply computes a linear interpolation of the cost on the edges of the flip graph defined by edge slides.

We now give an upper bound on K below which any K-Lipschitz stable algorithm for kinetic EMST performs asymptotically as bad as any fixed tree. Given the complexity of the problem, our bound is fairly crude. We state it anyway to demonstrate the use of our framework, but we believe that a stronger bound exists. Before we go into the details, we first show the asymptotic approximation ratio of any spanning tree.



(a) Red and blue points move in opposite directions. **(b)** The red edge slides over stretching edge T. **(c)** The length of edge T at time t is $\sqrt{x^2+t^2}$.

Figure 2.13 An instance where any $\frac{c}{\log n}$-Lipschitz stable algorithm performs poorly, for small enough constant $c > 0$.

2.5.1 Lemma. *Any spanning tree on a set of n points is an $O(n)$-approximation of the EMST.*

Proof. Let T be an EMST on point set P with total edge length OPT. Additionally let $u, v \in P$, and observe that the path along T from u to v is at least the Euclidean distance between u and v, $\|u, v\| \leq \textsc{path}_T(u, v)$. Furthermore, any path along an EMST is at most as long as the total length of an EMST, $\textsc{path}_T(u, v) \leq$ OPT. If we now take an arbitrary spanning tree T' on the same point set P, then we know that each edge (u', v') in this spanning tree has at most length $\|u', v'\| \leq \textsc{path}_T(u', v') \leq$ OPT. Since T' has $n - 1$ edges, its total length is $O(n) \cdot$ OPT. □

2.5.2 Lemma. *Let $d_{\mathcal{S}}$ be the metric for edge slides, then $\rho_{\text{LS}}(EMST, \frac{c}{\log n}, d_{\mathcal{I}}, d_{\mathcal{S}}) = \Omega(n)$ for a small enough constant $c > 0$, where n is the number of points.*

Proof. Consider the instance where n points are placed equidistantly vertically above each other with distance $1/n$ between two consecutive points. Now let $\mathcal{A}$ be any $(c/\log n)$-Lipschitz stable algorithm for the kinetic EMST problem and let T be the tree computed by $\mathcal{A}$ on this point set. We now color the points red and blue based on a 2-coloring of T. We then move the red points to the left by $\frac{1}{2}$ and the blue points to the right by $\frac{1}{2}$ in the time interval $[0, 1]$ (see Figure 2.13a). This way every edge of T will be stretched to a length of $\Omega(1)$ and thus the length of T will be $\Omega(n)$. On the other hand, the length of the EMST in the final configuration is OPT $= O(1)$. Therefore, we must perform an edge slide (see Figure 2.13b). However, we show that $\mathcal{A}$ cannot complete any edge slide. Consider any edge of T and let x be the initial (vertical) distance between its endpoints. Then the length of this edge can be described as $L(t) = \sqrt{x^2 + t^2}$ (see Figure 2.13c). Now assume that we want to slide an endpoint over this edge. To finish this edge slide before $t = 1$, we require that $\int_0^1 \frac{c}{\log n\sqrt{x^2+t^2}}\mathrm{d}t \geq 1$. This solves to $c\log(1/x + \sqrt{1 + 1/x^2}) \geq \log n$. However, since $x \geq$

$1/n$, we get that $c \log(1/x + \sqrt{1 + 1/x^2}) \leq c \log(n + \sqrt{1 + n^2}) < \log n$ for c small enough. Finally, since one edge slide can reduce the length of only one edge to $o(1)$, the cost of the solution at $t = 1$ computed by $\mathcal{A}$ is $\Omega(n)$. Thus, $\rho_{\mathrm{LS}}(\mathrm{EMST}, \frac{c}{\log n}, d_{\mathcal{I}}, d_{\mathcal{S}}) = \Omega(n)$ for a small enough constant $c > 0$. □

2.6 Conclusion

We presented a framework for algorithm stability, which includes three types of stability analysis: event stability, topological stability, and Lipschitz stability. The framework also distinguishes between three models for algorithms on time-varying data: stateless, state-aware and clairvoyant algorithms. We also demonstrated the use of this framework by applying the different types of analysis to state-aware algorithms for the kinetic EMST problem, deriving new interesting results. By providing different types of stability analysis with increasing degrees of complexity, we make stability analysis for algorithms more accessible. Chapters 3 and 4 show this by presenting more results on the stability of geometric algorithms, for both the k-center and orientation-based shape descriptor problems.

However, the framework that we presented does not (yet) give a complete picture: we do not consider the algorithmic aspect of stability, which gives insights into the trade-offs with running time. For example, can we develop a clairvoyant algorithm to efficiently compute a stable function of solutions over time that is optimal with regard to solution quality? Or, in a more restricted sense, can we efficiently compute one solution that is best for all inputs over time? Even for state-aware algorithms we can consider designing efficient algorithms that are K-Lipschitz stable and perform well with regard to solution quality. The latter question is further explored in Chapter 4, we analyze K-Lipschitz stable state-aware algorithms for orientation-based shape descriptors. We develop a clairvoyant algorithm to compute a topological stable solution to the k-center problem in Chapter 3.

Additionally, there are still many problems left open for the kinetic EMST problem. Future work could focus on the algorithmic models other than state-aware algorithms. However, for state-aware algorithms we could strive to tighten the bounds on the topological stability ratio for both edge slides and rotations. We would also like to extend our results on the Lipschitz stability of kinetic EMSTs, by considering K-Lipschitz stable solutions for different values of K. A description of the Lipschitz stability ratio as a function of K would be the ultimate goal.

Chapter 3

Kinetic k-Centers

In the previous chapter, we introduced a framework for algorithm stability. We will apply our framework to *facility location* problems in this chapter, specifically, to variations of the kinetic k-*center* problem. The k-center problem asks for a set of k shapes (e.g. disks) that cover a given set of n points, such that the radii of the shapes are as small as possible. The problem can be interpreted as placing a set of k facilities (e.g. stores) such that the distance from every point (e.g. client) to the closest facility is minimized. While this usually means that the maximum distance between point and facility is minimized, we also consider variations in which the sum of distances between points and the closest facilities are minimized.

Since the introduction of the k-center problem by Sylvester [105], the problem has been widely studied and has found many applications in practice, for example, in urban planning where a limited number of facilities should be available to all citizens in a reasonable radius around their homes. The complexity of the k-center problem varies with the function used to measure the distance between facilities and input points. The most commonly used variants of the problem are the Euclidean and rectilinear k-center problem, which measure distances under the ℓ_2 norm and the ℓ_∞ norm, respectively. In these variants, the shapes in a solution are disks and axis-aligned squares, respectively. Although the k-center problem is NP-hard in both settings, if k is part of the input [76], efficient algorithms have been developed for small k. Using rectilinear distance, the problem can be solved in $O(n)$ time [34, 62, 102] for $k = 2, 3$ and in $O(n \log n)$ time [84, 99] for $k = 4, 5$. The problem becomes

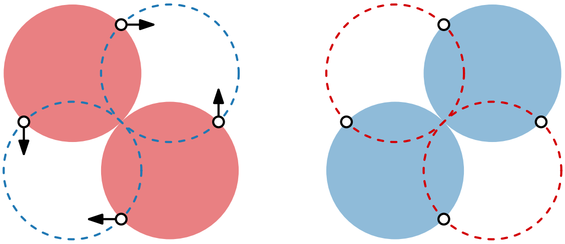


Figure 3.1 A small change in point positions causes a large change in the smallest 2 covering disks.

harder when using Euclidean distance, and the currently best known algorithm for Euclidean 2-centers runs in $O(n \log^2 n (\log \log n)^2)$ time [26].

In the kinetic version of the k-center problem, the input points are moving. This problem finds a lot of practical applications in, for example, mobile networks and robotics. A number of kinetic data structures have been developed for maintaining (approximate) k-centers [31, 43, 45, 46]. In many practical applications, the disks in the solution should move smoothly as the input points are moving smoothly. For example, this is the case when the disks in the solution represent objects in the physical world such as the mobile hotspots considered in Chapter 1, or if the shapes are used to represent groups of animals in a time-varying visualization. Stability is therefore of utmost importance for the k-center problem, as small changes in the input should lead to small changes to the disks. The optimal k-center may exhibit discontinuous changes as points move: the disks in an optimal solution do not follow continuous trajectories, as can be seen in Figure 3.1. This makes the k-center problem a perfect candidate for finding trade-offs between solution quality and stability.

In computational geometry there are a few results on the trade-off between solution quality and stability, and almost all of them concern the facility location problems. Specifically, the stability of kinetic 1-center and 1-median problems has been studied by Bespamyatnikh *et al.* [13, 17, 16]. These problems ask to find the mean/median position for a set of moving points. They showed that the speed at which the center/median point moves is higher than the speed of the input points. Futhermore, they developed various approximations by fixing the speed of the center/median point and provide results on the trade-off between solution quality and speed. A simple example of such an approximation is the rectilinear 1-center, which approximates the Euclidean 1-center, and moves with speed at most $\sqrt{2}$ when the input moves at unit speed. Durocher and Kirkpatrick [36, 37] studied the stability of the

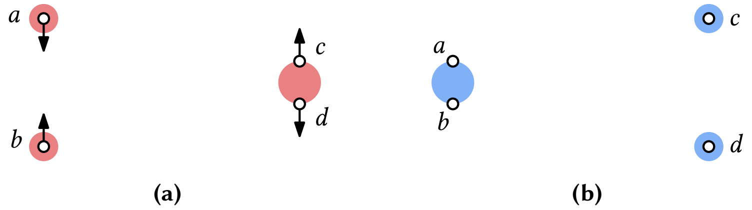


Figure 3.2 An instance for $k = 3$ with unbounded Lipschitz stability.

kinetic Euclidean 2-center problem, for which the center points can undergo discontinuous movement. They showed that an approximation ratio of $8/\pi \approx 2.55$ can be maintained when the disks can move with speed $8/\pi + 1 \approx 3.55$. Similarly, in the black-box KDS model, de Berg *et al.* [14] showed an approximation ratio of 2.29 for Euclidean 2-centers with maximum speed $4\sqrt{2}$. In our framework, these existing results would be classified as analyzing the Lipschitz stability of state-aware algorithms for the k-center problem with $k = 1, 2$.

In this chapter, we continue this line of research and analyze the stability of the k-center problem for $k > 2$. For k-centers with $k > 2$, no approximation factor can be guaranteed with disks of any bounded speed, as shown by Durocher and Kirkpatrick [35]. Figure 3.2 shows an instance that requires infinite speed of the disk centers for $k = 3$. In Figure 3.2a we see point a and b moving closer, and c and d moving apart. When the distance between a and b becomes less than the distance between c and d, one of the disks covering a or b has to move to c or d, to keep the radius of the disks minimal. The resulting solution can be seen in Figure 3.2b. Since the distance between a, b and c, d can be arbitrarily large, either a disk has to move with arbitrarily high speed, or, given bounded speed, the radius of the disk has to be arbitrarily large to ensure that all points are covered at all times. This configuration can occur in any instance of the k-center problem for $k \geq 3$, hence the K-Lipschitz stability ratio of the k-center problem may be unbounded for any bounded K.

The previous example shows that it is impossible to improve the Lipschitz stability of the k-center problem for $k > 2$. However, in our framework we defined *topological stability* as a less restrictive version of Lipschitz stability, to still be able to analyze stability whenever Lipschitz stability is unachievable: in contrast to Lipschitz stability, there is no bound on the speed at which the output can change for topological stability. Before we can analyze the topological stability of the k-center problem, we must first define an optimization function. In this chapter, we consider four variants of the k-center problem, each with a unique optimization function.

k-center variants An instance of the k-center problem is characterized by three choices: the number k of covering shapes, the geometry of the covering shapes and the criterion that measures solution quality. In this chapter, we consider two types of covering shapes: (a) in the *Euclidean* model, the covering shapes are disks; (b) in the *rectilinear* model, the shapes are axis-aligned squares. The radius of a covering shape is the distance from its center to its boundary, under the ℓ_2 norm for the Euclidean model and the ℓ_∞ norm for the rectilinear model. Furthermore, we distinguish two criteria: (a) in the *minmax* model, the quality of a solution is the maximum radius of its covering shapes, and the optimization criterion is to minimize this maximum radius; (b) in the *minsum* model, the quality of a solution is the sum of radii of all k covering shapes, and the optimization criterion is to minimize this sum of radii.

The above results in four variants of the problem that can be defined for any $k \geq 2$. We use the notation k-EC and k-RC to denote the Euclidean and rectilinear k-centers problem, and append either -minmax or -minsum to indicate the quality criterion.

Topological stability In the context of the k-centers problem, topological stability is determined by the following parameters. Let $\mathcal{I}$ denote the input space of n (stationary) points in $\mathbb{R}^2$ and $\mathcal{S}^k$ the solution space of all configurations of k disks or squares of varying radii. Let Π denote the k-center problem with criterion $f : \mathcal{I} \times \mathcal{S}^k \longrightarrow \mathbb{R}$ (minmax or minsum). We call a solution in $\mathcal{S}^k$ valid for an instance in $\mathcal{I}$ if it covers all points of the instance. An optimal algorithm OPT maps an instance of $\mathcal{I}$ to a solution in $\mathcal{S}^k$ that is valid and minimizes f.

We capture the continuous motion of points in a topology $\mathcal{T}_\mathcal{I}$, here chosen as the standard topology on $\mathbb{R}^{2n}$, and we capture the continuity of solutions in a topology $\mathcal{T}_\mathcal{S}^k$, of k disks or squares with continuously moving centers and radii. Remember that a topologically stable algorithm $\mathcal{A}$ maps a path I in $\mathcal{T}_\mathcal{I}$ to a path S in $\mathcal{T}_\mathcal{S}^k$.

Results and organization In Section 3.1, we prove various bounds on the topological stability of state-aware algorithms for the k-center problem. Starting with k-EC-minmax, the topological stability ratio is $\sqrt{2}$ for $k = 2$, while for arbitrary k, we prove an upper bound of 2 and a lower bound that converges to 2 as k tends to infinity. For small k, we show an upper bound strictly below 2 as well. For the other three variants, the stability ratio is exactly 2 for any $k \geq 2$. In Section 3.2, we provide a clairvoyant algorithm to compute a topologically stable solution for an instance of the kinetic k-center problem in polynomial time for constant k. The approximation ratio of such a solution is generally better than the generic bounds we prove in Section 3.1.

▸ 3.1 Bounds on topological stability

As illustrated in Figure 3.1, some point sets have more than one optimal solution. Recall from Chapter 2 that if we can transform an optimal solution into another, by growing the covering disks or squares at most (or at least) a factor r, we immediately obtain an upper bound (or respectively a lower bound) of r on the topological stability. To analyze the topological stability of the k-center problem, we therefore start with an input instance for which there is more than one optimal solution, and continuously transform one optimal solution into another. This transformation allows the centers to move along a continuous path, while their radii can grow and shrink continuously. At any point during this transformation, the intermediate solution should cover all points of the input. The maximum approximation ratio r that we need for such a transformation, gives a bound on the topological stability ratio. We can simply consider the input to be static during the transformation, since for topological stability the solution can move arbitrarily fast.

Before analyzing topological stability, we first introduce some tools to help us model and reason about these transformations. We then focus on the Euclidean minmax case, and finally, we consider the minsum and rectilinear cases.

2-colored intersection graphs Consider a point set P and two sets of k convex shapes (disks, squares, ...), such that each set covers all points in P: we use R to denote the one set (red) and B to denote the other set (blue). We now define the *2-colored intersection graph* $G_{R,B} = (V, E)$: each vertex represents a shape ($V = R \cup B$) and is either red or blue; E contains an edge for each pair of differently colored, intersecting shapes. A 2-colored intersection graph always contains equally many red nodes as blue nodes, as there are k shapes of each color. The shapes of both colors in a 2-colored intersection graph must cover all points: points may only be in the area of intersection between a blue and red shape, otherwise a point is not covered by one of the two colors. In the remainder, we use *intersection graph* to refer to 2-colored intersection graphs. Furthermore, we refer to the intersection graph as an *intersection forest*, if the underlying graph is a forest.

3.1.1 Lemma. *An intersection forest has at least one node of degree at most* 1 *of each color.*

Proof. Let F be an intersection forest. We prove that F has at least one red node of degree at most 1; the blue case is symmetric. Since F contains equally many blue and red nodes, there must be a tree T in F having at least as many red nodes as blue nodes. To arrive at a contradiction, assume that T has only blue leaves.

We decompose T into paths as follows. Pick an arbitrary leaf as a root. Partition the nodes of T into paths such that each path starts at a nonroot leaf, e.g. by running a BFS from each such leaf simultaneously following edges towards the root or using a heavy-path decomposition. Because T is part of an intersection graph, each path alternates between red and blue nodes. Hence, the path ending at the root, starting and ending at a blue leaf, has one more blue node than red nodes; the other paths cannot have more red nodes than blue nodes, since at least one endpoint is a leaf and thus blue. Now, T has more blue than red nodes, which contradicts that T has at least as many red as blue nodes. Thus, T cannot have only blue leaves. □

3.1.2 Lemma. *Consider two sets R and B of k convex translates each covering a point set P. If intersection graph $G_{R,B}$ is a forest, then R can morph onto B without increasing the shape size, while covering all points in P.*

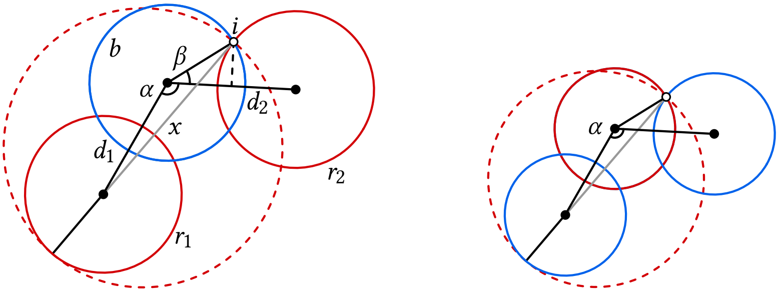
Proof. We prove this lemma by induction on k. For the base case, $k = 0$, intersection graph $G_{R,B}$ is empty and thus morphing all red shapes onto blue shapes is trivial.

For $k > 0$, we reason as follows. Since $G_{R,B}$ is a forest, Lemma 3.1.1 tells us that there is a red node r with degree at most 1. If r has degree 1, then its one neighbor b must be a blue node; if r has degree 0, then we pick any blue node b. We morph r onto b by linearly moving the center of r to the center of b. Since r and b are convex translates, this covers their intersection at all times. Now, the new position of the red shape covers any point originally covered by r or b. Consider $R' = R \setminus \{r\}$ and $B' = B \setminus \{b\}$. These sets have size $k - 1$ and define an intersection forest $G_{R',B'}$ with $k - 1$ shapes. The induction hypothesis readily tells us that there is a morph from R' into B' without increasing their size. The morph of r onto b, followed by the morph of the smaller instance yields us a morph from R to B. □

Euclidean minmax case We are now ready to analyze the Euclidean minmax case. Without loss of generality, we assume here that the disks all have the same radius. We first need a few results on (static) intersection graphs, to argue later about topological stability.

3.1.3 Lemma. *Let R and B to be optimal solutions to a point set P for k-EC-minmax. Assume the intersection graph $G_{R,B}$ has a 4-cycle with a red degree-2 vertex. To transform R in such a way that $G_{R,B}$ misses one edge of the 4-cycle, while covering the area initially covered by both sets, it suffices to increase the disk radius of a red disk by a factor $\sqrt{2}$.*

Proof. To morph from R to B, a red disk r_1 has to grow to cover the intersection of an adjacent blue disk b with the other (red) neighbor r_2 of b. Once r_1 has grown to overlap the intersection between a blue disk and r_2, r_2 no longer has to cover this



(a) Angle α occurs at a blue disk b. Red disk r_1 can grow to radius $x + 1$ to also cover the overlap between b and r_2.

(b) Angle α occurs at a red disk.

Figure 3.3 The smallest angle α occurs in a cycle of overlapping disks.

intersection and can be treated as a degree-1 vertex in $G_{R,B}$. The intersection graph no longer has the 4-cycle now.

As we have a 4-cycle of intersections, a, b, c, d, we either have to cover both a and c while covering d or b, or we have to cover b and d while covering either a or c. Let $p_a \in a$ and $p_c \in c$ be the pair of points whose distance is the longest of any pair from a and c, and similarly $p_b \in b$ and $p_d \in d$ for b and d. We claim that distance (p_a, p_c) or (p_b, p_d) is shorter than $2\sqrt{2}$. Assume that distance $(p_a, p_c) > 2\sqrt{2}$, otherwise we are done. Since the disks have radius 1, distances $(p_a, p_b), (p_b, p_c), (p_c, p_d), (p_d, p_a)$ are at most 2. Our assumption on (p_a, p_c) now implies that the distance from either p_b or p_d to the middle of the line between p_a and p_c is shorter than $\sqrt{2}$. By the triangle inequality, distance (p_b, p_d) is now shorter than $2\sqrt{2}$.

Assume without loss of generality that (p_a, p_c) is shorter than $2\sqrt{2}$ and that p_a and p_b are covered by a red disk with only two overlaps. Combining this with the fact that $(p_a, p_b), (p_b, p_c)$ are at most 2, we can conclude that triple (p_a, p_b, p_c) can be covered by growing the red disk with only two overlaps to radius $\sqrt{2}$. □

3.1.4 Lemma. *Let R and B to be optimal solutions to a point set P for k-EC-minmax. Assume the intersection graph $G_{R,B}$ has only degree-2 vertices. To transform the disks of R onto B, while covering the area initially covered by both sets, it is sufficient to increase the disk radius by a factor* $\left(1 + \sqrt{1 + 8\cos^2(\frac{\pi}{2k})}\right)/2$.

Proof. As the problem is invariant under scaling, we assume without loss of generality that the radii of disks are 1. To morph from R to B, a red disk r_1 has to grow to cover the intersection of an adjacent blue disk b with the other (red) neighbor r_2 of b (see dashed red disk in Figure 3.3). Instead of growing r_1 to tightly cover the two intersections b has with its neighbors, we grow r_1 to fully cover its initial disk and the intersection between b and r_2. As a result, we now have to consider only r_1, b, r_2 without concerning ourselves with the other neighbor of r_1 or r_2.

Let r_1 be the red disk that has to grow the least, of all red disks in our instance. Let $0 \leq d_1, d_2 \leq 2$ be the distance between the centers of r_1 and b and between r_2 and b respectively. We know that $d_1 \leq d_2$, as otherwise r_2 has to grow less than r_1 to cover the other intersection of b. However, if d_2 is smaller, the intersection between b and r_2 is larger so r_1 may have to grow more. We can therefore conclude that in the worst case $d_1 = d_2 = d$.

We use α to denote the angle at the center point of b (see Figure 3.3a). Larger values of α readily lead to a higher maximum radius for stretching r_1. Since $G_{R,B}$ is a cycle, the $2k$ disk centers thus form the vertices of a simple polygon and we find that $\alpha \leq \frac{\pi(k-1)}{k}$. The boundaries of b and r_2 intersect in at least one point; we are interested in the point i that is the furthest away from the center point of r_1. Let β denote the angle at the center of b between rays towards i and the center of r_2. We know that $\cos(\beta) = d/2$. The distance x between i and the center of r_1 can be found using the Law of Cosines: $x^2 = d^2 + 1^2 - 2d\cos(\alpha + \beta)$. The diameter of r_1 when overlapping its initial area and the intersection between b and r_2 is $1 + x$. If the described configuration occurs only with the smallest angle α at a red disk (instead of at a blue disk as in our construction), the red disk can grow to overlap both its intersections and fully cover one of the blue disks adjacent to it. This also results in a disk with diameter $1 + x$ that allows us to break the cycle. Figure 3.3b shows this configuration.

Since $d_1 = d_2 = d$, the problem is fully symmetric and r_1 has an equivalent of point i under the same angle β. Since all points covered by b are in the intersections with r_1 and r_2, the angle $\alpha + 2\beta$ must enclose a diametrical pair for b to be an optimal disk, thus we find $\alpha + 2\beta \geq \pi$. As spanning more than the diametrical pair only forces d to become smaller, we find that in the worst case, this is in fact an equality and we get $\beta = \pi\frac{1}{2k}$. Hence, $\alpha + \beta = \pi - \beta$ and we can derive that $\cos(\alpha + \beta) = \cos(\pi - \beta) = -\cos(\beta)$. Since. $d = 2\cos(\beta)$ we can conclude that $1 + x \leq 1 + \sqrt{1 + d^2 - 2d\cos(\alpha + \beta)} \leq 1 + \sqrt{1 + 8\cos^2(\frac{\pi}{2k})}$. The diameter of r_1 after growing is exactly $1 + x$, thus its radius is exactly half this expression. □

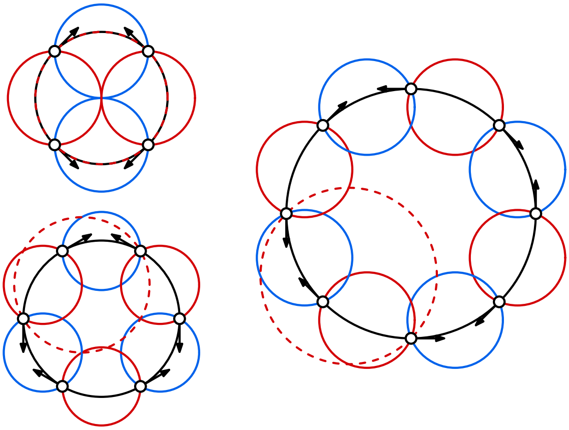


Figure 3.4 Lower bound construction for the stable minmax Euclidean k-center problem, shown for $k = 2, 3, 4$. The optimal solution changes from the red to the blue solution. To break the cycle one of the red disks has to grow to the dashed red disk.

3.1.5 Lemma. *Let R and B be optimal solutions to a point set P for k-EC-minmax. Assume the intersection graph $G_{R,B}$ has only degree-2 vertices. To transform the disks of R onto B, while covering the area initially covered by both sets, it may be necessary to increase the disk radius by a factor $2\sin(\frac{\pi(k-1)}{2k})$.*

Proof. Consider a point set of $2k$ points, positioned such that they are the corners of a regular $2k$-gon with unit radius, i.e., equidistantly spread along the boundary of a unit circle. There are exactly two optimal solutions for these points (see Figure 3.4). To morph from R to B, one of the red disks r_1 has to grow to cover the intersection of an adjacent blue disk b with the other (red) neighbor r_2 of b (see dashed red disk in Figure 3.4). Since the points are all at equal distance from each other on a unit circle, they are the vertices of a regular $2k$-gon. The diameter of the disks in our optimal solution equals the length of a side of this regular $2k$-gon. This means that a red disk has to grow such that its diameter is equal to the distance between a vertex of the $2k$-gon and a second-order neighbor. Hence, the radius of r_1 has to grow to with a factor $2\sin(\frac{\pi(k-1)}{2k})$. Once r_1 has grown to overlap the intersection between b and r_2, r_2 no longer has to cover the points in the intersection and can be treated as a degree-1 vertex in $G_{R,B}$. Since that makes the intersection graph a tree, we can morph R into B without further increasing the radii using Lemma 3.1.2. □

We are now ready to prove bounds on the topological stability of the k-EC-minmax problem for moving points. The upcoming lemmata establish the following theorem.

3.1.6 Theorem. *For k-EC-minmax, we obtain the following bounds:*

- $\rho_{\mathrm{TS}}(2\text{-}\mathit{EC}\text{-}\mathit{minmax}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{2}) = \sqrt{2}$
- $\sqrt{3} \leq \rho_{\mathrm{TS}}(3\text{-}\mathit{EC}\text{-}\mathit{minmax}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{3}) \leq \left(1 + \sqrt{7}\right)/2$
- $2\sin(\frac{\pi(k-1)}{2k}) \leq \rho_{\mathrm{TS}}(k\text{-}\mathit{EC}\text{-}\mathit{minmax}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{k}) \leq 2$ *for* $k > 3$.

3.1.7 Lemma. $\rho_{\mathrm{TS}}(k\text{-}\mathit{EC}\text{-}\mathit{minmax}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{k}) \leq 2$ *for* $k \geq 2$.

Proof. Consider a point in time t where there are two optimal solutions; let R denote the solution that matches the optimal solution at $t - \varepsilon$ and B the solution at $t + \varepsilon$ for arbitrarily small $\varepsilon > 0$. Let C be the maximum radius of the disks in R and in B. Furthermore, let intersection graph $G_{R,B}$ describe the above situation. First we make a maximal matching between red and blue vertices that are adjacent in $G_{R,B}$, implying a matching between a number of red and blue disks. The intersection graph of the remaining red and blue disks has no edges, and we match these red and blue disks in any way.

We find a bound on the topological stability as follows. All the red disks that are matched to blue disks they already intersect, grow to overlap their initial disk and the matched blue disk. Now the remaining red disks can safely move to the blue disks they are matched to, and adjust their radii to fully cover the blue disks. Each blue disk is now fully covered by the red disk that it was matched to. Finally, all red disks shrink to match the size of the matched blue disk to finish the morph. When all the red disks are overlapping blue disks, the maximum of their radii is at most $2C$, since each red disk grows by at most the radius of the matched blue disk. □

3.1.8 Lemma. $\rho_{\mathrm{TS}}(k\text{-}\mathit{EC}\text{-}\mathit{minmax}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{k}) \geq 2\sin(\frac{\pi(k-1)}{2k})$ *for* $k \geq 2$.

Proof. The bound readily follows from Lemma 3.1.5, if we can show that a set of *moving* points can force the exact same swap to happen. To this end, consider the $2k$ points in the construction to move at unit speed along tangents of the unit circle, which define the point placement. The direction of the points is alternately clockwise and counterclockwise with respect to the circle. We use t to indicate the time at which the points are in the positions needed for Lemma 3.1.5 (see Figure 3.4).

To see that this morph has to happen at time t, consider the following. At some time t' before t, the pairs of points covered by the red disks in the construction at time t, are all coinciding: hence the optimal solution then has maximum radius

0; to not violate our bound, we must have this solution at that time. Morphing R into B between t' and t requires a red disk to grow its radius more than a factor $2\sin(\frac{\pi(k-1)}{2k})$, since the red disks are still smaller than the blue disks, and the next point to cover is further away. Analogously, we can argue that we must morph to the blue solution, before a time t'' at which the pairs covered by blue disks in the construction coincide. Additionally, the morph cannot happen between t and t'', since this requires a radius more than a factor $2\sin(\frac{\pi(k-1)}{2k})$ larger than optimal. We conclude that the morph has to happen at time t and thus requires the maximum radius to grow by a factor $2\sin(\frac{\pi(k-1)}{2k})$. □

3.1.9 Lemma. $\rho_{\mathrm{TS}}(2\text{-}EC\text{-}minmax, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^2) = \sqrt{2}$.

Proof. The lower bound follows directly from Lemma 3.1.8 by using $k = 2$. For the upper bound, consider a point in time t where there are two optimal solutions; let R denote the solution that matches the optimal solution at $t - \varepsilon$ and B the solution at $t + \varepsilon$ for arbitrarily small $\varepsilon > 0$. If $G_{R,B}$ is a forest, Lemma 3.1.2 applies and we do not need to increase the maximum radius during the morph. If $G_{R,B}$ contains a cycle, the entire graph must be a 4-cycle. Lemma 3.1.3 gives an upper bound of $\sqrt{2}$ for transforming $G_{R,B}$ from a 4-cycle into a tree. Finally, we can morph R into B without further increasing the maximum radius using Lemma 3.1.2. □

3.1.10 Lemma. $\sqrt{3} \le \rho_{\mathrm{TS}}(3\text{-}EC\text{-}minmax, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^3) \le \left(1 + \sqrt{7}\right)/2$.

Proof. Consider a point in time t where there are two optimal solutions; let R denote the solution that matches the optimal solution at $t - \varepsilon$ and B the solution at $t + \varepsilon$ for arbitrarily small $\varepsilon > 0$. If intersection graph $G_{R,B}$ is a forest, then Lemma 3.1.2 applies and we do not need to increase the maximum radius during the morph. If $G_{R,B}$ contains a cycle, then either the entire graph is a 6-cycle, or there are smaller cycles. If the entire graph is a 6-cycle, the upper bound follows from Lemma 3.1.4 and the lower bound from Lemma 3.1.8.

Consider the case where $G_{R,B}$ contains a cycle, but no 6-cycle. There is at least one 4-cycle. As $k = 3$, every vertex has degree at most 3. Note that two overlapping disks can be covered by a single disk without increasing the maximum radius beyond $\left(1 + \sqrt{7}\right)/2$, if $1 + d/2 \le \left(1 + \sqrt{7}\right)/2$, where d is the distance between the centers of the disks. We now distinguish the following cases:

- If there is at most one red degree-3 vertex, every 4-cycle contains at least one degree-2 red vertex. Therefore, Lemma 3.1.3 can be used to break one of the 4-cycles by increasing the radius of a red disk by at most $\sqrt{2}$. If $G_{R,B}$ has another 4-cycle after breaking the first one, we can apply Lemma 3.1.3

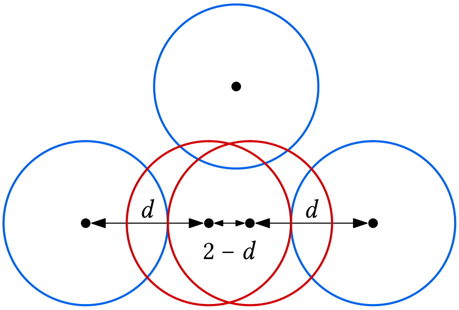
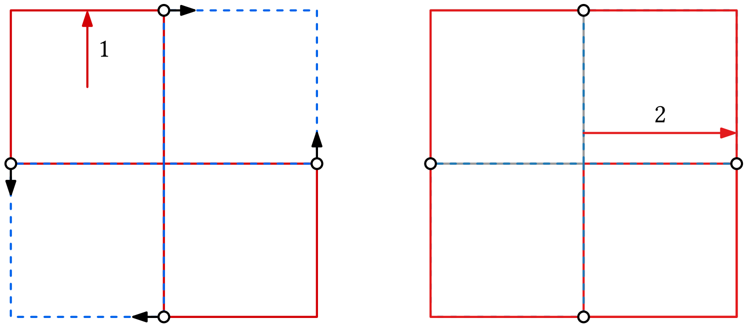


Figure 3.5 Multiple red disks overlap three blue disks. The distance between the center points of the red disks is at most $2 - d$, when the distance between the center points of the red and blue pairs is at least d.

again. However, if breaking the first 4-cycle resulted in a 6-cycle, then the distance between the center points of two adjacent disks in the 4-cycle was less than $\sqrt{2} < \sqrt{7} - 1$. We can therefore fully cover this pair of adjacent disks with the red disk instead of breaking the 4-cycle. We need a radius of at most $1 + \sqrt{2}/2 < 1 + (\sqrt{7} - 1)/2 = \left(1 + \sqrt{7}\right)/2$ for this. Since the red disk now covers all intersections of the blue disk, the resulting intersection graph is a tree.

- If there are two or more red degree-3 vertices, we look at the distances d between the center points of the overlapping disks. If there is a pair of red and blue disks for which $1 + d/2 \leq \left(1 + \sqrt{7}\right)/2$, we can fully cover the blue disk with the red disk. The remainder of the red disks can now be seen as vertices of degree-2 or less in the intersection graph. If there is still a 4-cycle, then Lemma 3.1.3 can be used to break the cycle. If for every pair of red and blue disks $1 + d/2 > \left(1 + \sqrt{7}\right)/2$ holds, then the centers of the red disks that overlap the three blue disks can be at most $2 - d$ away from each other (see Figure 3.5). We can cover two red disks r_1, r_2 with a single red disk r_1 of radius $1 + (2 - d)/2 < 1 - \left(1 + \sqrt{7}\right)/4 < \left(1 + \sqrt{7}\right)/2$. We can then freely transform red disk r_2 to a blue disk. Again the remainder of the red disks can now be seen as vertices of degree-2 or less in the intersection graph. If a 4-cycle remains, then Lemma 3.1.3 can be used to break the cycle.

In both cases $G_{R,B}$ consists of trees after the changes that were made, thus R can morph into B without further increasing the maximum radius by Lemma 3.1.2. In all the above cases we need to grow the maximum radius during the transformation from R to B by at most a factor $\left(1 + \sqrt{7}\right)/2$, in some it is necessary to grow to $\sqrt{3}$. □



(a) The optimal solution (of radius 1) changes from the solid red to the dashed blue solution.

(b) A red square of radius 2 is required to cover at least three points.

Figure 3.6 Lower bound construction for kinetic minmax rectilinear 2-center.

The above proof shows the strengths and weaknesses of the earlier lemmata. While in many cases we can get close to tight bounds, dealing with high degree vertices in the intersection graph requires additional analysis. Furthermore, in general we cannot upper bound the Lipschitz stability ratio [35], but the bounds in Theorem 3.1.6 act as lower bounds for such bounded speed solutions.

Rectilinear and minsum cases We now turn to the remaining cases, for which the stability ratio is 2 when $k \geq 2$. This is captured in the following theorems.

3.1.11 Theorem. $\rho_{\mathrm{TS}}(k\text{-}RC\text{-}minmax, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}^{k}) = 2$ *for* $k \geq 2$.

Proof. The upper bound readily follows from the argument of Lemma 3.1.7. We prove the lower bound for $k = 2$, using that higher values of k cannot lead to a weaker lower bound.

Consider an instance consisting of four points, two points move with unit speed over the lines $y = 2$ and $y = -2$ respectively, in opposite directions, while the other two points move with unit speed along the lines $x = -2$ and $x = 2$ in opposite directions. Assume that at some time t the points are in the positions $(0, 2), (2, 0), (0, -2), (-2, 0)$. There is exactly one optimum solution for this instance before t and exactly one after t. However at time t there are two possible optimum solutions (see Figure 3.6a).

To ensure that the squares together cover all points at all times, and that the centers of the squares move in continuous fashion, one of the squares has to grow to cover at least three of the points. After this has happened the second square can move in position of the other optimum solution, followed by shrinking the square that covers three points. To cover at least three points, one of the squares has to grow its radius ($r = 2$) to two times the size of the maximum radius of any of the optimum solutions ($r = 2$) (see Figure 3.6b).

Strictly before or after t it is impossible to achieve the same ratio: The radius of the largest square in the optimum solution is smaller before and after t, hence the difference between the necessary radius and the optimum radius increases. □

3.1.12 Lemma. *$\rho_{\mathrm{TS}}(k\text{-}EC\text{-}minsum, \mathcal{T}_{\mathcal{I}}, \mathcal{T}^k_{\mathcal{S}}) \leq 2$ for $k \geq 2$.*

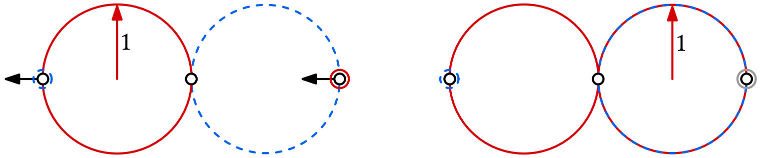
Proof. Consider a point in time t where there are two optimal solutions; let R denote the solution that matches the optimal solution at $t - \varepsilon$ and B the optimal solution at $t + \varepsilon$ for arbitrarily small $\varepsilon > 0$. Let C be the sum of radii of R, and equivalently the sum of radii of B. We morph R onto B while covering all the points with a sum of radii of at most $2C$.

First make a matching between disks in R and disks in B. If we look at the intersection graph $G_{R,B}$, we want to create a matching between red and blue vertices that are adjacent. Finding such a matching is easy for cycles since they have as many red disks as blue disks. However in some cases we might not be able to find a matching between overlapping disks. When $G_{R,B}$ is a forest there can be trees that have more red disks than blue disks and vice versa. In these cases we map the remaining red disks (leaves in $G_{R,B}$) to the remaining blue disks (also leaves in $G_{R,B}$) arbitrarily.

We find a bound on the topological stability as follows. All the red disks that are matched to blue disks they already intersect grow to overlap their initial disk and the blue disk they are matched to. Now the remaining red disks can safely move to the blue disks they are matched to, and adjust their radius to fully cover the blue disks. Finally, all red disks shrink to match the size of the blue disk they overlap to finish the morph. When all the red disks are overlapping blue disks, the sum of radii is at most $2C$, since the radius C_r of each red disk r grows by at most the radius C_b of the blue disk b it is matched to. □

3.1.13 Theorem. *$\rho_{\mathrm{TS}}(k\text{-}EC\text{-}minsum, \mathcal{T}_{\mathcal{I}}, \mathcal{T}^k_{\mathcal{S}}) = \rho_{\mathrm{TS}}(k\text{-}RC\text{-}minsum, \mathcal{T}_{\mathcal{I}}, \mathcal{T}^k_{\mathcal{S}}) = 2$ for $k \geq 2$.*

Proof. The upper bound for the Euclidean case follows from Lemma 3.1.12. The proof only use the triangle inequality and therefore work for general metrics, in particular they work for ℓ_2 and ℓ_∞.



(a) The optimal solution changes from the solid red to the dashed blue; the smaller disks have radius 0, thus the total radius is 1.

(b) Doubly covering the center point is required (as illustrated) or one disk should cover all points (not illustrated); the total radius is 2.

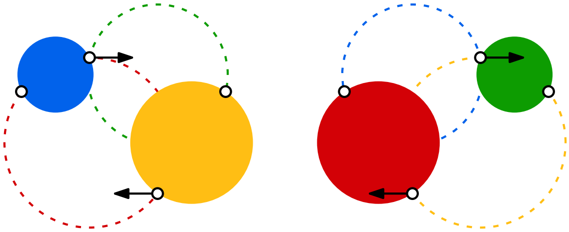
Figure 3.7 Lower bound construction for kinetic minsum Euclidean 2-center.

The lower bound construction for $k = 2$ uses three points: $(0, -2), (0, 0), (0, 2)$, admitting two optimal solutions, both with a disk of radius 1 and a disk of radius 0 (see Figure 3.7a). Morphing between these requires an intermediate state that double-covers $(0, 0)$ (see Figure 3.7b), or one disk covering all three: the total radius is then 2. Since the lower bound construction is essentially one dimensional, the disks can be traded for squares, and thus works for both the Euclidean and rectilinear case. □

3.2 Algorithms for kinetic *k*-center

Topological stability captures the worst-case penalty that arises from making transitions in a solution continuous. In this section we are interested in the corresponding algorithmic problems that generally result in smaller penalties for a specific instance: how efficiently can we compute an (unstable) k-center for an instance with n moving points, and how efficiently can we compute a topologically stable k-center? When we combine these two algorithms, we can determine for any instance how large the penalty is, when solving the given instance in a topologically stable way. We examine these questions for all four k-center variants.

The second algorithm gives us a topologically stable solution to a particular instance of k-center. This solution can be used in a practical application requiring stability, such as the wifi hotspot scenario sketched in the introduction: if k people are hosting wifi hotspots for n of their friends, for example at a festival, then we may assume that the friends stay stationary or move slowly most of the time. A solution to the k-center problem, which corresponds to the position of the k wifi hotspots, may change with arbitrary speed, but since the friends are stationary, there is plenty of time to move the hotspots to their new positions.



(a) A point set is shown at two points in time. Pairs of dashed disks form valid candidate k-centers, while pairs of filled disks form valid candidate k-centers with minimal maximum radius. Disks of the same color are defined by the same points. Disks not drawn are either larger than drawn disks, or require a larger disk to form a valid candidate k-center.

Figure 3.8 Visualization of the unstable k-EC-minmax algorithm (for $k = 2$).

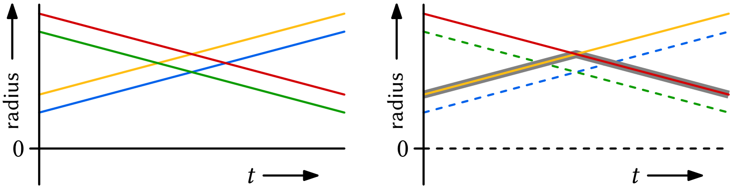
▸ 3.2.1 Unstable k-center algorithms

Let P be a set of n points moving in the plane, each represented by a constant-degree algebraic function that maps time to the plane. We denote the point set at time t as $P(t)$ and will develop an algorithm that computes the smallest maximum radius r needed to cover all points with k disks at any point in time.

Observe that the minimum covering disks of a point set $P(t)$, denoted $S^*(t)$, is a set of k disks where each disk is defined by three points in $P(t)$, two points as a diametrical pair, or a singleton point. In other words, we can define S^* as the Cartesian product of k triples, pairs, and singletons of distinct points from the set $P(t)$. Not every triple is always relevant: if the circumcircle of the three points is not the boundary of the smallest covering disk, then the triple is irrelevant at that time. Each triple is relevant on $O(1)$ time intervals. On the other hand, pairs and singletons always define relevant disks. This allows us to define what we call *candidate* k-centers.

3.2.1 Definition (Candidate k-centers). Any set of k disks $D_1, \dots, D_k$ where each disk is the minimum covering disk of one, two or three points in $P(t)$ is called a *candidate k-center* and is denoted $S(t)$. A candidate k-center is *valid* if the union of its disks cover all points of $P(t)$.

This definition allows us to rephrase the goal of the algorithm: for each time t we want to compute the smallest value $C(t)$, such that there exists a valid candidate k-center $S(t)$ where the disks in $S(t)$ have at most radius $C(t)$ or where their radii sum to $C(t)$ for the minmax and minsum model respectively.



(b) The arrangement obtained by plotting the radii of the drawn disks and disks on singleton points. In the right arrangement, curves that are not maxcurves are dashed, while the maximum radii of the optimal solution are marked in grey.

Unstable k-EC-minmax For each singleton, pair, or triple in P we can find the minimum covering disk. Let the radius of such a disk be r. As the points move along their trajectories, the radius of the minimum covering disk changes over time (unless defined by a singleton point). The function over time which gives this radius is continuous, because the points that define this radius move continuously. We refer to the images of these functions as curves, for brevity. Taking every singleton, pair, and triple of points, we get $O(n^3)$ curves that represent the radii of the minimum covering disks. Any pair of these curves intersects $O(1)$ times, since the points corresponding to each curve follow constant degree algebraic functions. This implies that the curves form an arrangement of complexity $O(n^6)$.

3.2.2 **Observation.** Each of the $O(n^3)$ curves can be split into $O(n)$ pieces where the same subset of points of $P(t)$ are inside the disk corresponding to the curve.

We are not interested in the arrangement as a whole, but only in the parts where the curves show the maximum radius of a valid k-center: we want to know when a minimum covering disk is the largest of the k disks that cover all the points. The curve of such a pair or triple may define a part of the solution in the minmax model. For ease of description we will now first continue with the algorithm for the 2-center case, and then show how to extend it to larger values of k.

Assume that a pair a, b or triple $a, b, c \in P$ has a minimum covering disk D_1 with radius r_1 at time t. Let $P_1 \subseteq P$ be all the points covered by D_1. To solve the 2-center problem, we need to cover all other points with another disk. Let the minimum covering disk of $P_2 = P \setminus P_1$ be D_2 with radius r_2. We say that the curve for pair a, b or triple a, b, c is a *maxcurve* at time t if $r_1 \geq r_2$ at time t.

A curve can only become a maxcurve at intersections of the curves, since the radii of two covering disks will be equal at an intersection. We refer to a piece of curve between intersections as an *arc*. The arrangement of all maxcurves still has complexity $O(n^6)$. It takes $O(n^7)$ time to compute this arrangement, since we need to check for every arc whether it is a maxcurve. Note that this has to be done only once for every arc in the arrangement. As we take all maxcurve arcs, we know that we keep only the parts of the initial arrangement where the maximum radius of a solution is represented. The lower envelope of this arrangement will therefore show the maximum radius of an optimum solution at any point in time (see Figure 3.8).

Finding the lower envelope of this arrangement takes $O(\lambda_{s+1}(n^6)\log(n))$ time when every pair of curves intersects at most s times [59], where $\lambda_s(n)$ is the maximum length of a Davenport–Schinzel sequence of order s with n distinct values [101]. The time needed to compute the lower envelope is dominated by the $O(n^7)$ time we spend on computing the arrangement of maxcurves itself.

To extend this algorithm to the Euclidean k-center problem, we observe that we can start with the same set of $O(n^3)$ curves for all singletons, pairs, and triples of moving points. We define a curve to be *maxcurve* if the not yet covered points can be covered by at most $k-1$ disks of no larger radius. For each of the $O(n^6)$ arcs of the arrangement this implies solving a static $(k-1)$-center problem, which takes $O(n^{2k-1})$ [33] or $n^{O(\sqrt{k})}$ time [64] and dominates computing the lower envelope.

3.2.3 Theorem. *Given a set of n points in the plane moving along constant-degree algebraic functions, the 2-EC-minmax problem can be solved in $O(n^7)$ time. The k-EC minmax problem can be solved in $O(n^{2k+5})$ or $n^{O(\sqrt{k})}$ time.*

Unstable k-EC-minsum We continue with the minsum version of the Euclidean k-center problem. In this variant we can no longer use maxcurves, since a solution should minimize the sum of radii, instead of the maximum radius. Instead, choose k curves and their corresponding k-center, and trace it over time to determine when the k-center covers all points. The number of times when a point enters or leaves any of the k disks of the k-center is $O(kn) = O(n)$. Hence, any choice of k curves gives $O(n)$ time intervals where the corresponding disks form a valid solution. We sum the k curves (the radii of the disks) on these intervals to get candidate k-center values for k-EC-minsum. In total, there are $O(n^{3k+1})$ new curves that are the sum of k original curves, and their lower envelope represents the desired solution $S^*(t)$.

3.2.4 Theorem. *Given a set of n points in the plane moving along constant-degree algebraic functions, the k-EC-minsum problem can be solved in $O(\lambda_s(n^{3k+1})\log n)$ time for some constant s, where $\lambda_s(n)$ is the maximum length of a (n, s)-Davenport–Schinzel sequence.*

Unstable k-RC-minmax The rectilinear version of the k-center problem in the minmax model is solved by similar methods as the Euclidean variant, albeit simpler and more efficient. We use pairs of points to define curves that plot the radius of a smallest covering square over time. The points should be on opposite sides of the covering square. This square is not unique, since there are $O(n)$ different subsets of points that can be covered by such a square. We first consider the 2-center case, where it is sufficient to take the two extreme squares: if the second square must cover points that lie beyond two opposite sides of the first square, then that second square must be larger. Therefore, in comparison the solution using two extreme squares has a smaller or equal maximum radius.

In total we have $O(n^2)$ curves that form an arrangement of complexity $O(n^4)$. We again use the concept of maxcurves: we are interested in those arcs of the arrangement, for which the not yet covered points can be covered by a square of no larger size. While we can test this easily in linear time for each of the $O(n^4)$ arcs, we can use the arrangement to do this faster. For each curve C corresponding to a square R, we process the points moving in and out of R and maintain the leftmost, rightmost, bottommost, and topmost uncovered points, for example using heaps. As the points move, each square will start and stop covering other points $O(n)$ times. The uncovered points can swap places $O(n^2)$ times in an ordering of their x- or y-coordinates. Thus we can maintain the left-, right, top-, and bottommost points in $O(n^2 \log n)$ for each curve C, and determine in constant time whether each of the $O(n^2)$ arcs along the curve is a maxcurve. Finally, we compute the lower envelop of the maxcurves in time linear in the number of maxcurves, since the maxcurves are disjoint arcs.

For k-centers, the observation that one can use the two extreme squares no longer holds. We therefore define squares by triples as in the Euclidean case, the main difference being that the $O(n^3)$ curves we get, have considerable overlap: there must still be two defining points on opposite sides of the covering squares defined by triples, hence all $O(n)$ squares that have these exact points on opposite sides must have the same area. The arrangement of the $O(n^3)$ curves is therefore equivalent to the arrangement for the $O(n^2)$ curves defined above, and each arc of the arrangement can be part of $O(n)$ curves. Since the $O(n^2)$ curves form an arrangement of complexity $O(n^4)$, we need to test $O(n^5)$ arcs for being maxcurves. Following the analysis as in the Euclidean case, we obtain:

3.2.5 Theorem. *Given a set of n points in the plane moving along constant-degree algebraic functions, the* 2*-RC-minmax problem can be solved in* $O(n^4 \log n)$ *time. The k-RC-minmax problem can be solved in* $O(n^{2k+4})$ *or* $n^{O(\sqrt{k})}$ *time.*

Unstable k-RC-minsum We can use exactly the same approach as in the Euclidean case, and obtain the following result:

3.2.6 Theorem. *Given a set of n points in the plane moving along constant-degree algebraic functions, the k-RC-minsum problem can be solved in* $O(\lambda_s(n^{3k+1}) \log n)$ *time for some constant s, where* $\lambda_s(n)$ *is the maximum length of a* (n, s)*-Davenport–Schinzel sequence.*

▸ 3.2.2 Topologically stable k-center algorithms

In this section we describe an algorithm to compute a topologically stable solution for the Euclidean k-center variants. We use only combinatorial properties of solutions, which also hold for the rectilinear k-center variants. Hence the same algorithm, replacing disks by squares, also solves the rectilinear variants.

Intuitively, the unstable algorithm finds the lower envelope of all the *valid* radii by traversing the arrangement of all valid radii over time. At each time t a minimal enclosing disk D_1 (defined by a set of at most three points) in the set of optimal disks $S^*(t)$ needs to be replaced with another disk D_2, we "hop" from our previous curve to the curve corresponding to the new disk D_2. If we require that the algorithm is topologically stable these hops have a cost associated with them.

We first show how to model and compute the cost $C(t)$ of a topologically stable transition between any two k-centers at a fixed time t. We then extend this approach to work over time. Let t be a fixed moment in time where we want to go from one k-center S_1 to another candidate k-center S_2. The transition can happen at infinite speed but must be continuous. We denote the infinitesimal time frame around t in which we do the transition as $[0, T]$. We extend the concept of a k-center with a corresponding partition of P over the disks in the k-center:

3.2.7 Definition (Disk set). For each disk D_i of a candidate k-center S for $P(t)$ we define its *disk set* $P_i \subseteq P(t) \cap D_i$ as the subset of points assigned to D_i. A candidate k-center S with disk sets $P_1, \dots, P_k$ is *valid* if the disk sets partition $P(t)$. We say S is *valid* if there exist disk sets $P_1, \dots, P_k$ such that S with disk sets $P_1, \dots, P_k$ is valid.

During a continuous tranformation, the disk sets of a valid candidate k-center will change in the time interval $[0, T]$ while the points $P(t)$ do not move. In essence the time t is equivalent to the whole interval $[0, T]$. For ease of understanding, we use t' to denote any time in the interval $[0, T]$. Observe that our definition of topological stability leads to an intuitive way of recognizing a stable transition:

3.2.8 Lemma. *A transition from candidate k-center $\mathcal{B}_1(t)$ to candidate k-center $\mathcal{B}_2(t)$ in the time interval* $[0, T]$ *is* topologically stable *if and only if the change of the disks' centers and radii is continuous over* $[0, T]$ *and at each time* $t' \in [0, T]$, $\mathcal{B}(t')$ *is* valid.

Proof. By definition the disks must be transformed continuously and all points in $P(t)$ are covered in $[0, T]$ precisely when a valid candidate k-center exists. □

Now that we can recognize a topologically stable transition, we can reason about what such a transition looks like:

3.2.9 Lemma. *Any topologically stable transition from k-center $S_1(t)$ to k-center $S_2(t)$ in the timespan $[0, T]$ that minimizes $C(t)$ (the largest occurring minsum/minmax over $[0, T]$) can be obtained by a sequence of events where in each event, occurring at a time $t' \in [0, T]$, a disk $D_i \in S(t')$ adds a point to P_i and a disk $D_j \in S(t')$ removes a point from P_j. We call this* transferring.

Proof. This proof is by construction. Assume we have a transition from $S_1(t)$ to $S_2(t)$, which minimizes either the sum or the maximum of all radii, and also assume that it contains simultaneous continuous movement. Let this transition take place in $[0, T]$.

To determine $C(t)$ we need to consider only times $t' \in [0, T]$, where a disk $D_i \in \mathcal{B}$ adds a point p to its disk set P_i and disk D_j removes p from P_j. Only at t' must both disks contain p; before t' disk D_j may be smaller and after t' disk D_i may be smaller.

We claim that for any optimal simultaneous continuous movement of cost $C(t)$, we can discretize the movement into a sequence of events with cost no larger than $C(t)$. We do so recursively: for the continuous movement there exists a $t_0 \in [0, T]$ which is the first time a disk $D_i \in S$ adds a point p to P_i. Then at t_0, D_i has to contain both P_i and p and must have a certain size d. All the other disks $D_j \in S$ with $j \neq i$ only have to contain the points in P_j so they have optimal size if they have not moved from time 0. In other words, it is optimal to first let D_i obtain p in an event and to then continue the transition in $[t_0, T]$. This allows us to recursively discretize the simultaneous movement into sequential events. □

3.2.10 Corollary. *Any topologically stable transition from k-center $S_1(t)$ to k-center $S_2(t)$ in the timespan $[0, T]$ that minimizes $C(t)$ (the largest occurring minsum/minmax over $[0, T]$) can be obtained by a sequence of events where in each event the following happens: a disk $D_i \in S_1(t)$ that was defined by one, two or three points in $P(t)$ is now defined by a new set of points in $P(t)$ where the two sets differ in only one element. With every event, P_i must be updated with a corresponding insert and/or delete. We call these events a* swap *because we intuitively swap one of the defining elements.*

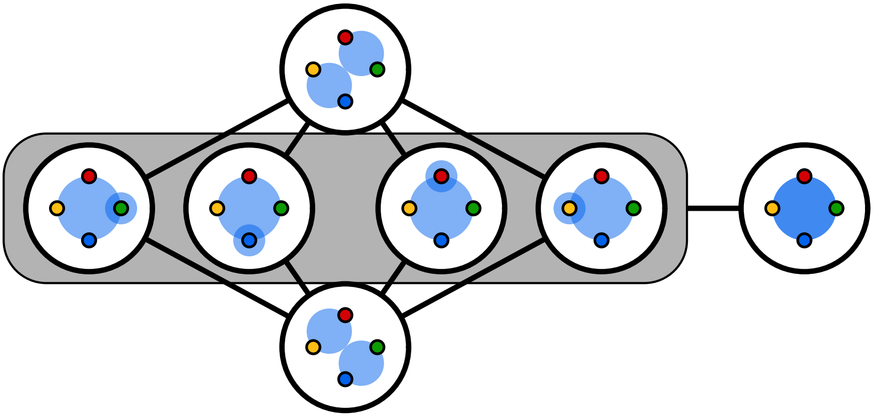


Figure 3.10 The G^2 graph on four points. Vertices of G^2 in the grey area all have an edge to the vertex on the right. This rightmost vertex has two disks defined by the pairs red and blue, and yellow and green.

The cost of a single stable transition Corollary 3.2.10 allows us to model a stable transition as a sequence of swaps, but how do we find the optimal sequence of swaps? A single minimal covering disk at time t is defined by at most three unique elements from $P(t)$, so there are at most $O(n^3)$ subsets of $P(t)$ that could define one disk of a k-center. Let these $O(n^3)$ sets be the vertices of a graph G. We create an edge between two vertices v_i and v_j, if we can transition from one disk to the other with a single swap and that transition is topologically stable. Each vertex is incident to only a constant number of edges (apart from degenerate cases) because during a swap the disk D_i corresponding to v_i can only add/remove one element to P_i. Moreover, the radius of the disk is maximal on vertices in G and not on edges. The graph G has $O(n^3)$ complexity and takes $O(n^4)$ time to construct, as we need to check for each vertex which edges should be added. This can be done by checking all $O(n)$ vertices for which the disk set differs by one element.

This graph provides a framework to trace the radius of the transition from a single disk to another disk. However, we want to transition from one k-center to another. We use the previous graph to construct a new graph G^k where each vertex w_i represents a set of k disks: a candidate k-center S_i. We again create an edge between vertices w_i and w_j if we can go from the candidate k-center S_i to S_j in a single swap. With a similar argument as above, each vertex is only connected to $O(k)$ edges. The graph thus has $O(n^{3k})$ complexity and can be constructed in $O(n^{3k+1})$ time. This time each of the $O(n^{3k})$ vertices has to check $O(kn) = O(n)$ other vertices to determine whether there should be an edge: for each of the k disks there are $O(n)$ vertices for which the disk set differs by one element. Each vertex w_i gets assigned the cost (minmax/minsum) of the k-center S_i where the cost is ∞ if S_i is invalid. Figure 3.10 shows an example of a G^2 graph.

Any connected path in this graph from w_i to w_j without vertices with cost ∞ represents a stable transition from w_i to w_j by Corollary 3.2.10, where the cost of the path (transition) is the maximum value of the vertices on the path. We can now find the optimal sequence of swaps to transition from any vertex w_i to w_j by finding the cheapest path in this graph in $O(n^{3k} \log n)$ time, for example using an adapted version of Dijkstra's algorithm to maintain the highest cost of the cheapest path. The running time is dominated by the $O(n^{3k+1})$ time it takes to construct the graph.

Maintaining the cost of a flip For a single point in time we can now determine the cost of a topologically stable transition from a k-center S_i to S_j in $O(n^{3k+1})$ time. If we want to maintain the cost $C(t)$ for all times t, the costs of the vertices in the graph must also change over time. If we plot the changes of these costs over time, the graph consists of monotonously increasing or decreasing segments, separated by moments in time (events) where two radii of disks are equal. A straightforward counting results in $O(n^{3k})$ of such events, dominated by splitting the cost function of each vertex in monotone segments. These events also contain all events where the structure of our graph G^k changes *and* all the moments where a vertex in our graph becomes *invalid* and thus gets cost ∞, which happen when four points become cocircular. The result of these observations is that we have a $O(n^{3k})$ size graph, with $O(n^{3k})$ relevant changes, where with each change we spend $O(kn) = O(n)$ time per vertex to restore the graph by recomputing the edge set. We can then compute an optimal solution as described in the previous section, at each event update G^k, and use the updated graph to find the cost of a continuous transition, if the event coincides with a discrete change in the optimal solution. This leads to an algorithm which can determine the cost of a topologically stable solution in $O(n^{6k+1})$ time.

3.2.11 Theorem. *Given a set of n points in the plane moving along constant-degree algebraic functions, the cost of a topologically stable solution to the k-EC-minmax/-minsum problem can be solved in* $O(n^{6k+1})$ *time.*

If we run the unstable and stable algorithms on the moving points, we obtain two functions that map time to a cost. The ratio of these two costs over time corresponds to the approximation ratio of the computed topologically stable solution at any point in time. The maximum of this ratio over time is the topological stability ratio of this solutoin, which is therefore obtained in $O(n^{6k+1})$ time as well. Since the stable algorithm is constructive, we can also find a topologically stable solution in the same time, by returning the set of disks corresponding to the minimum cost path through the time-varying graph G^k.

3.3 Conclusion

We considered the topological stability of common variants of the kinetic k-center problem. Topolocially stable solutions to these problems must change continuously but may do so arbitrarily fast. We have established tight bounds for the minsum case (Euclidean and rectilinear) as well as for the rectilinear minmax case for any $k \geq 2$. For the Euclidean minmax case, we proved nontrivial upper bounds for small values of k and presented a general lower bound tending towards 2 for large values of k. We also presented algorithms to compute topologically stable solutions together with the cost of stability for a set of moving points. This cost is measured by the approximation factor that our solution achieves for that particular set of moving points at any point in time. A practical application of these algorithms would be to identify points in time where we could slow down the solutions produced by the algorithms, to explicitly show stable transitions between optimal solutions.

Future work It remains open whether a general upper bound strictly below 2 is achievable for k-EC-minmax. We conjecture that this bound is indeed smaller than 2 for any constant k. For this, we need more insight in how to continuously transform solutions represented by an intersection graph with more general structures. Our algorithms to compute the cost of stability for an instance have high (albeit polynomial) run-time complexity. Can the results for KDS (e.g. [14]) help us speed up these algorithms? Alternatively, can we approximate the cost of stability more efficiently?

We considered topological stability in this chapter, but for typical applications, a bound on the speed at which a solution may change is required. We would hence like Lipschitz stable k-centers, but for $k > 2$ the Lipschitz stability ratio can be unbounded, if the centers have to move continuously [35]. A potentially interesting variant of the solution space topology is one where a disk may shrink to radius 0, at which point it disappears and may reappear at another location. Since this alleviates the problem in the example, it raises the question: would it allow us to bound the Lipschitz stability?

Chapter

4

Kinetic Orientation-Based Shape Descriptors

Shape descriptors are simplified representations of more complex shapes. They are used as summaries of a large collection of data, where we are not interested in all the details, but simply want to have an overview of the most important features. Shape descriptors play an important role in many fields that perform shape analysis, such as computer vision (shape recognition) [12, 20, 122], computer graphics (bounding boxes for broad-phase collision detection) [7, 54, 68, 15], medical imaging (diagnosis or surgical planning) [22, 55, 67, 123], and machine learning (shape classification) [104, 113, 120, 121]. We study *orientation-based* shape descriptors, that specifically capture the orientation of the input – in our case a point set. Such shape descriptors can be used in applications that ask for an orientation, even when it might not be very pronounced. For example, we can visualize how a large set of moving points evolves, by drawing a glyph that captures the direction of movement (for example an arrow or line segment) for a few subsets of the points. A different application that can benefit from utilizing orientation-based shape descriptors is the MotionRugs visualization technique [23], which produces visual summaries for movement data. These summaries consist of reasonable 1D representations of moving points, and one way of finding such representations is by projecting the input points to a line at each time step. High-quality summaries are obtained, if the orientation of this line coincides with a principal component of the point set, as we show in Chapter 5.

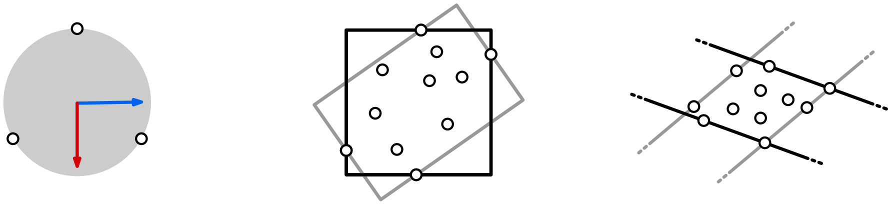


Figure 4.1 A flip in orientation for PC, OBB and STRIP. Small changes in the positions of the points make one orientation optimal over the other.

The three shape descriptors we consider in this chapter are the first *principal component* (PC), the smallest area *oriented bounding box* (OBB) and the thinnest *covering strip* (STRIP). We study these orientation-based shape descriptors in the kinetic setting: the n points that are used as input continuously change their position over time, while we compute their shape descriptors. However, even small changes in the point cloud can result in discrete "flips" in the optimal orientation of all three shape descriptors (see Figure 4.1). We propose stable algorithms for these orientation-based shape descriptors, and analyze their topological and Lipschitz stability.

Problem description Our input consists of a set of n moving points $P = P(t) = \{p_1(t), \dots, p_n(t)\}$ in two dimensions, where the trajectory $p_i(t)$ of each point is a continuous function $p_i : [0, T] \longrightarrow \mathbb{R}^2$. We assume that, at each time t, not all points are at the same position. The output consists of an orientation $\alpha = \alpha(t)$ of the shape descriptor for every point in time $t \in [0, T]$, which need not match the optimal orientation due to stability constraints. While an orientation $\alpha(t)$ is an element of the real projective line $\mathbb{RP}^1$, we typically represent $\alpha(t)$ by a unit vector in $\mathbb{R}^2$ and implicitly identify opposite vectors, which is equivalent. To quantify the quality of any output orientation, we define each of the three shape descriptors Π as the minimum of an optimization function $f_\Pi(\alpha, P)$. Let $\mathcal{L}(\alpha)$ be the set of lines with orientation α, and let $d(L, p)$ be the distance between a point p and a line L. Furthermore, let the *extent* of P along a unit vector α be $w_\alpha(P) = \max_{p,q \in P}(p - q) \cdot \alpha$, and let $\alpha^\perp$ be the vector (orientation) orthogonal to α. We can define the orientation of PC, OBB or STRIP as the orientation α that minimizes the following functions:

Principal component: $f_{\text{PC}}(\alpha, P) = \min_{L \in \mathcal{L}(\alpha)} \sum_{p \in P} d(L, p)^2$

Oriented bounding box: $f_{\text{OBB}}(\alpha, P) = w_\alpha(P) w_{\alpha^\perp}(P)$

Covering strip: $f_{\text{STRIP}}(\alpha, P) = w_{\alpha^\perp}(P)$

Though various other options are possible, we believe these functions naturally fit to the shape descriptors. The optimization functions quantify the quality of a shape descriptor for any orientation α. This allows us to also consider shape descriptors of suboptimal quality, which do not fully minimize the optimization function. This in turn enables us to make a trade-off between quality and stability. When computing a shape descriptor, we typically compute more than just an orientation, such as the exact position and dimensions of the descriptor. However, the stability is mostly affected by the optimal orientation: if the optimal orientation changes continuously and the points move continuously, then the shape descriptor changes continuously as well. We therefore ignore other aspects of the shape descriptors to analyze their stability, and assume that these aspects are chosen optimally for the given orientation without any cost with regard to the stability. Under this assumption, every orientation defines a unique shape descriptor.

Algorithmic models In this chapter we consider both stateless and state-aware algorithms, and we define a special kind of state-aware algorithm called a *chasing algorithm*. Remember that a stateless algorithm can be topologically stable only if it defines a mapping from input to output that is naturally continuous. On the other hand, a state-aware algorithm can enforce continuity by using its state to keep track of earlier outputs. Let $\mathcal{A}$ be an algorithm mapping input to output, and let $I(t)$ be the input depending on time t. We can then define a chasing algorithm as follows.

Chasing algorithm: The algorithm $\mathcal{A}$ maintains the most recent output (in our case the orientation α of a shape descriptor) as the state S and uses only the input at the current time $I(t)$ to find a new optimal solution OPT. The algorithm then moves the solution in S towards the new optimal solution OPT at the maximum allowed speed. This solution is the output for the current time t and is subsequently stored in S.

For Lipschitz stable chasing algorithms the output at time t might not coincide with OPT, as the solution in S was too different from OPT and the maximum allowed speed did not allow S to catch up to OPT: the output of our algorithm chases after the optimal solution. The challenge lies in bounding the ratio between the quality of the solutions used by the algorithm and the (unstable) optimal quality. We assume that a chasing algorithm maintains a solution over time, and it can choose the rate of change of the solution (e.g., rotation speed/direction) at every point in time.

Stability analysis In this chapter we analyze the topological and Lipschitz stability of the orientation-based shape descriptors. First, we need to choose appropriate topologies on the input space and the output space for the topological stability analysis. Given the described problem setting, choosing the topologies is straightforward: the input space is $\mathbb{R}^{2n}$, for which we choose the standard topology. The output space is topologically equivalent to the unit circle, which we denote here by O.

Second, for the Lipschitz stability analysis we need metrics to measure the distance between input instances and between solutions we output. We measure the output in radians, hence the difference between two outputs is a signed angle. For the input we use the maximum difference in position of the points under the ℓ_2 norm (equivalent to the measure used in Chapter 2).

Following the definition of K-Lipschitz stability described in Chapter 2, we assume the input points move with at most unit speed, and allow the orientation of the output to change with a speed of at most K radians per time unit. To accommodate the use of a chasing algorithm, we allow the orientation of the output to change, even when the input points stay stationary.

Remember that K-Lipschitz stability requires scale invariance: to derive meaningful bounds on the K-Lipschitz stability ratio, it is necessary that the relation between distances and speeds in input and output space should be scale invariant. This is currently not the case: if we scale the coordinates of the points, then the distances in the input space change accordingly, but the distances in the output space (between orientations) do not. To remedy this problem, we require that diameter D of $P(t)$ is at least 1 for every time t. Note that we can always appropriately scale to meet these assumptions, if not all points coincide in a single location. The unit diameter assumption is only mild, if points represent actual objects; the objects, for example humans or fish, have a certain size and cannot occupy the same physical space.

Related work Shape descriptors are a wide topic, studied in various subfields of computer science. We focus on the related work for PC, OBB, and STRIP, which shows that these three shape descriptors are related, yet slightly different. The ratio between the volume of the bounding box using the orientation of PC and the minimal-volume bounding box is unbounded [32]. Similar to PC, other fitness measures have been considered with respect to oriented lines, such as the sum of distances or vertical distances [27]. Computing OBB for static point sets is a classic problem in computational geometry. In two dimensions, one side of the optimal box aligns with a side of the convex hull and it can be computed in linear time after finding the convex hull [42, 108]; a similar property holds in three dimensions, allowing a cubic-time algorithm [85]. The relevance of bounding boxes in 3D as a component of other algorithms also led to efficient approximation algorithms, such as a $(1+\epsilon)$-approximation algorithm in $O(n + 1/\epsilon^{4.5})$ time or an easier algorithm with running time $O(n \log n + n/\epsilon^3)$ [8]. Bounding boxes find applications in tree structures for spatial indexing [10, 58, 95, 100] and in speeding up collision detection and ray-tracing techniques [7, 54, 15]. The optimal STRIP can be computed using the same techniques as OBB in two dimensions [42, 108]. Agarwal et al. [2] provide an $O(n + 1/\epsilon^{O(1)})$-time approximation algorithm via ϵ-kernels for various measures of a point set, including STRIP and OBB. Finally, the medial axis of a point set is a shape descriptor that is often used in computational topology. While this shape descriptor is not orientation-based, we mention it because it is the subject of one of the few results on stability in computational geometry. Letscher and Sykes [72] show that the medial axis for a union of disks changes continuously under certain conditions or if it is pruned appropriately.

Results and organization In Section 4.1 we prove that there exists no stateless algorithm for any of the three shape descriptors that is both topologically stable and achieves a bounded approximation ratio for the quality of an optimal shape descriptor. We then consider state-aware algorithms and analyze the topological stability for each of the shape descriptors in Section 4.2. In Section 4.3 we analyze the Lipschitz stability of chasing algorithms for all three shape descriptors. We show that chasing algorithms with sufficient speed can achieve a constant approximation ratio for OBB and STRIP, if we indeed assume that the points move with at most unit speed and that the diameter of the point set is always at least 1. We briefly discuss why this algorithm is not viable for PC. To the best of our knowledge, this is the first time a chasing algorithm has been analyzed, which deals with discrete changes. We conclude this chapter in Section 4.4.

4.1 Stateless algorithms

We prove that stateless algorithms cannot achieve bounded topological stability ratio for any of the three shape descriptors. This readily implies an unbounded K-Lipschitz stability ratio for any K. Important for the theorem below is that, if all points of set P lie on a single line with orientation α, then $f_\Pi(\gamma, P) = 0$ iff $\gamma = \alpha$, for a shape descriptor Π. The argument in the proof is entirely topological, and intuitively works as follows: we construct a set of inputs that is topologically equivalent to a disk in the input space. All inputs P at the boundary of this disk consist of n points that lie on a single line with orientation α, hence the ratio between $f_\Pi(\gamma, P)$ and $f_\Pi(\alpha, P)$, with $\gamma \neq \alpha$, will be unbounded. We show that no continuous function exists that maps the inputs at the boundary of the topological disk to correct orientations.

4.1.1 Theorem. *For stateless algorithms $\rho_{TS}(\textsc{pc}) = \rho_{TS}(\textsc{obb}) = \rho_{TS}(\textsc{strip}) = \infty$ if the point set contains at least three points.*

Proof. The idea is to construct a continuous map $j : D^2 \longrightarrow \mathbb{R}^{2n}$ on a two-dimensional closed disk D^2, such that the image of j consists of valid point sets in $\mathbb{R}^2$ (which do not have all points at the same position), and that the image of ∂D^2 under j forces the orientation of the shape descriptor. We parameterize D^2 using polar coordinates (r, ϕ) for $0 \leq \phi < 2\pi$ and $0 \leq r \leq 1$. We first construct a map j' as follows:

$$j'(r,\phi) = \left\{ \left(\frac{ri}{n} \sin\phi, \frac{ri}{n} \cos\phi \right) \mid 1 \leq i \leq n \right\}$$

Since j' always places all points on a line for a given ϕ, the orientation of the shape descriptor is always forced (otherwise the approximation ratio is ∞). However, $j'(0, \phi)$ is not a valid point set, since it places all points at the origin. Now let P^* be a point set of n points in $\mathbb{R}^2$, consisting of least three points that are not all collinear. Interpreting P^* as a vector in $\mathbb{R}^{2n}$, we define $j(r, \phi) = j'(r, \phi) + (1 - r)P^*$. By the choice of P^*, $j(r, \phi)$ is always a valid point set. Furthermore, the orientation of the shape descriptors is still fixed for point sets $j(1, \phi) = j'(1, \phi)$, namely $\alpha = \phi \pmod{\pi}$. As a result, any stateless algorithm with an approximation ratio $\rho < \infty$ defines, along with j, a continuous mapping h from D^2 to unit circle O where ∂D^2 is mapped twice around O, i.e., a double cover of O. The continuous mapping h is simply the output of the stateless algorithm composed with j. We claim that mapping h cannot exist.

For the sake of contradiction, assume that such a map h exists. Let $f, g : O \longrightarrow D^2$ be continuous functions. Function f maps every point $x \in O$ to the boundary $\partial D^2 \subseteq D^2$ such that the mapping covers the whole boundary once, while g maps all $x \in O$ to a single point $y \in D^2$. We can continuously shrink the image of f to a single point in D^2, in particular the image of g; hence f and g are homotopic. We now consider $h \circ f$ and $h \circ g$ and use the degree of these mappings (as first defined in [21]) to show that h cannot exist. Since f maps O to the boundary $\partial D^2 \subseteq D^2$, and h maps ∂D^2 to a double cover of O, we know that the degree of $h \circ f$ is two. On the other hand, g maps all of O to a single point in D^2, therefore $h \circ g$ has degree 0. By the Hopf theorem [80] $h \circ f$ and $h \circ g$ cannot be homotopic, as they can only be homotopic if and only if their degrees are equal. However, $h \circ f$ and $h \circ g$ must be homotopic, since homotopy equivalence is compatible with function composition and f is homotopic to g. This contradiction implies that h cannot exist. Thus the topological stability ratio is ∞ for stateless algorithms. □

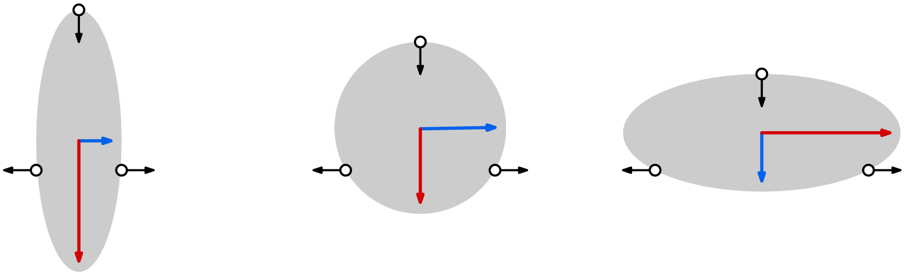


Figure 4.2 A flip between the first (red) and second principal component (blue).

4.2 Topological stability

In this section we turn to state-aware algorithms, and we analyze the topological stability of the shape descriptors. Specifically, we prove the following tight bounds.

4.2.1 Theorem. *The topological stability ratios of the shape descriptors are:*

- $\rho_{\text{TS}}(\textit{PC}) = 1$,
- $\rho_{\text{TS}}(\textit{OBB}) = \frac{5}{4}$,
- $\rho_{\text{TS}}(\textit{STRIP}) = \sqrt{2}$.

4.2.2 Lemma. $\rho_{\text{TS}}(\textit{PC}) = 1$

Proof. Consider a time t where the first principal component flips between distinct orientations, represented by unit vectors $\vec{v}_1$ and $\vec{v}_2$. The first principal component is the orientation of the line that minimizes the sum of squared distances between the points and the line, corresponding to the optimal solution defined by f_{PC}. It can be computed by centering the point set at the mean of the coordinates, computing the covariance matrix of the resulting point coordinates, and extracting the eigenvector of this matrix with the largest eigenvalue. Since eigenvalues change continuously if the data changes continuously [109, Theorem 3.9.1], both $\vec{v}_1$ and $\vec{v}_2$ must have some eigenvalue λ^* at time t. But that means that every interpolated vector $\vec{v} = (1-u)\vec{v}_1 + u\vec{v}_2$ also has eigenvalue λ^*, since $C\vec{v} = (1-u)C\vec{v}_1 + uC\vec{v}_2 = (1-u)\lambda^*\vec{v}_1 + u\lambda^*\vec{v}_2 = \lambda^*\vec{v}$. As a result, $f_{\text{PC}}(\vec{v}) = f_{\text{PC}}(\vec{v}_1)$, and we can continuously change orientation from $\vec{v}_1$ to $\vec{v}_2$ without decreasing the quality of the shape descriptor. □

The bounds on $\rho_{\text{TS}}(\text{OBB})$ and $\rho_{\text{TS}}(\text{STRIP})$ are split into upper and lower bounds. and hence in the upcoming proofs we follow the general structure outlined in Chapter 2.

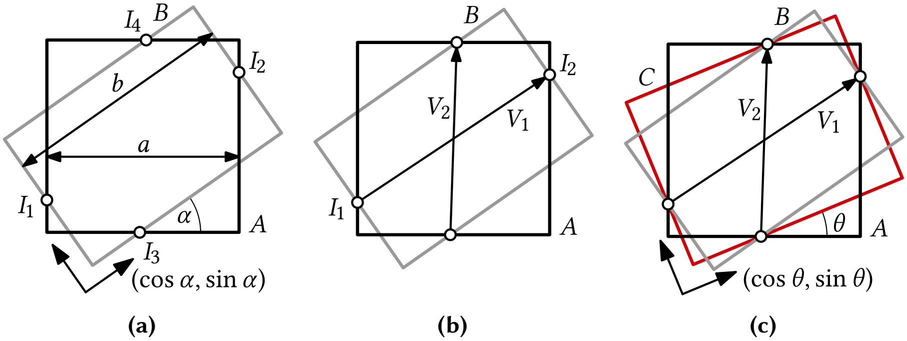


Figure 4.3 Construction of closed formula for area of intermediate solution C.

4.2.3 Lemma. $\rho_{\mathrm{TS}}(\textsc{obb}) \leq \frac{5}{4}$

Proof. Consider a time t at which two distinct oriented bounding boxes A and B have minimum area; both are assumed to have the smallest area 1 without loss of generality, as the problem at hand is invariant under scaling. At this time t we continuously change the orientation of the box between that of A and B while making sure that the box still contains all points (see Figure 4.3a). The goal is to compute the maximal size of the intermediate box in the worst case. Note that we may rotate either clockwise or counterclockwise; we always choose the direction that minimizes this maximal intermediate size.

Let a and b denote the length of the major axes of A and B respectively. Let angle α denote the smallest angle between the orientations of the major axes. Note that $\alpha \in \{0, \pi/2\}$ leads to A and B being identical, and that our problem is invariant under rotation, reflection and translation. We thus assume without loss of generality:

- $b \geq a \geq 1$;
- $0 < \alpha < \pi/2$;
- B is centered at the origin, and A at (dx, dy);
- the major axis of A is horizontal;
- α describes a counterclockwise angle from the major axis of A to the major axis of B.

The points P must all be contained in the intersection of A and B, for otherwise A and B would not be bounding boxes. Furthermore, no side of A may be completely outside B or vice versa, for otherwise one of the boxes could be made smaller. Thus, all sides intersect, and we are interested in four of these intersections $I_1, \dots, I_4$ (see Figure 4.3b). Specifically, we want to use the intersections that allow us to derive a valid upper bound on the size of the bounding box during rotation. Since the intersections depend on the direction of rotation, we choose the intersections that allow us to rotate from A to B in counterclockwise direction. The coordinates of the intersections can easily be computed (see Figure 4.3b). For example, for $I_1 = (x_1, y_1)$ we solve for the following two equations: $x_1 = dx - a/2$ and $x_1 \cos\alpha + y_1 \sin\alpha = -b/2$. The resulting coordinates of all intersections are:

- $I_1 = (-\frac{a}{2} + dx, -\frac{b-(a-2dx)\cos\alpha}{2\sin\alpha})$
- $I_2 = (\frac{a}{2} + dx, \frac{b-(a+2dx)\cos\alpha}{2\sin\alpha})$
- $I_3 = ((-\frac{1}{2a} + dy)\frac{\cos\alpha}{\sin\alpha} + \frac{1}{2b\sin\alpha}, -\frac{1}{2a} + dy)$
- $I_4 = ((\frac{1}{2a} + dy)\frac{\cos\alpha}{\sin\alpha} - \frac{1}{2b\sin\alpha}, \frac{1}{2a} + dy)$

Now consider an intermediate box C with angle $\theta \leq \alpha$ with respect to box A (see Figure 4.3c). Note that C contains the intersection of A and B as long as it contains $I_1, \dots, I_4$. We can define two vectors $V_1 = I_2 - I_1$ and $V_2 = I_4 - I_3$, which we project to lines at angle θ and $\theta + \frac{\pi}{2}$ to obtain the lengths of the sides of C. Note that V_1 and V_2 depend only on a, b and α, but not on dx and dy. Thus, using V_1 and V_2 we can obtain a formula for the area of C, which we call $\mathcal{C}$.

$$\begin{aligned}\mathcal{C}(a, b, \alpha, \theta) &= V_1 \cdot (\cos\theta, \sin\theta) \times V_2 \cdot (-\sin\theta, \cos\theta) \\ &= \frac{(b\sin(\alpha-\theta) + a\sin\theta) * (a\sin(\alpha-\theta) + b\sin\theta)}{ab\sin^2\alpha}\end{aligned}$$

We are now interested in the maximum of $\mathcal{C}$. We start by finding the partial derivative of $\mathcal{C}$ with respect to θ:

$$\frac{\partial\mathcal{C}}{\partial\theta} = \frac{(a^2 + b^2 - 2ab\cos\alpha)\sin(\alpha - 2\theta)}{ab\sin^2\alpha}$$

First observe that in the chosen domain $\frac{\partial\mathcal{C}}{\partial\theta} = 0$ if and only if $\theta = \alpha/2$, which implies that we can set $\theta = \alpha/2$. Using double-angle formulas, we may simplify $\mathcal{C}$.

$$\begin{aligned}
C(a, b, \alpha, \alpha/2) &= \frac{(a+b)^2 \sin^2(\alpha/2)}{ab \sin^2 \alpha} \\
&= \frac{(a+b)^2 \sin^2(\alpha/2)}{ab(2\sin(\alpha/2)\cos(\alpha/2))^2} \\
&= \frac{(a+b)^2}{2ab(2\cos^2(\alpha/2))} \\
&= \frac{(a+b)^2}{2ab(1+\cos(\alpha))}
\end{aligned}$$

We now split the domain of α into $(0, \frac{\pi}{4}]$ and $(\frac{\pi}{4}, \frac{\pi}{2})$, and prove both cases separately.

In the first case, when $\alpha \in (0, \frac{\pi}{4}]$, let $c \geq 1$ be such that $b = ca$ (since $b \geq a$). As b is at most the length of the diagonal of A, we get that $c \leq \sqrt{a^2 + (1/a)^2}/a = \sqrt{1 + 1/a^4} \leq \sqrt{2}$ (since $a \geq 1$). The resulting formula is $C(a, ca, \alpha, \alpha/2) = \frac{(1+c)^2}{2c(1+\cos\alpha)}$. This function is maximized when c and α are maximized. We thus set $\alpha = \pi/4$ and $c = \sqrt{2}$ to obtain that $C(1, \sqrt{2}, \pi/4, \pi/8) \leq \frac{1}{2} + \frac{1}{2}\sqrt{2} < \frac{5}{4}$ and thus this case meets the bound claimed.

What remains is to prove the case where $\alpha \in (\frac{\pi}{4}, \frac{\pi}{2})$. It might now be beneficial to rotate A clockwise instead of counterclockwise, to align the minor axis of A with the major axis of B: this clockwise rotation may result in smaller intermediate solutions C. Since C can only deal with counterclockwise rotation, we have to use different parameters to deal with the described situation. To simulate the clockwise rotation, we use $C(1/a, b, \pi/2 - \alpha, \theta)$; this reflects the whole setup over direction $\pi/4$ effectively considering the minor axis as the major axis instead. Note that we did not use the assumption that $a \geq 1$ anywhere above, until within the other case where $\alpha \leq \pi/4$. Note that $\frac{\partial C}{\partial \theta} = 0$ depends on the parameters we fill in, hence we can set $\theta = (\pi/2 - \alpha)/2 = \pi/4 - \alpha/2$ to find a maximum in this case.

For all possible values of a, b and α, we need to find the area of the largest intermediate box C. Since we can choose whether we rotate clockwise or counterclockwise, we find the area by taking the minimum of $C(a, b, \alpha, \alpha/2)$ and $C(1/a, b, \pi/2 - \alpha, \pi/4 - \alpha/2)$. We first simplify the latter.

$$\begin{aligned} C(1/a, b, \pi/2 - \alpha, \pi/4 - \alpha/2) &= \frac{(\frac{1}{a} + b)^2}{2\frac{1}{a}b(1 + \cos(\pi/2 - \alpha))} \\ &= \frac{a^2(\frac{1}{a} + b)^2}{2ab(1 + \sin(\alpha))} \\ &= \frac{(1 + ab)^2}{2ab(1 + \sin(\alpha))} \end{aligned}$$

To find the maximum of the function, we can use a mathematical program that looks for the values of a, b and α that comply to a set of constraints and maximize a target function. All constraints come from the assumptions, but we add a final constraint similar to what we did in the $\alpha \in (0, \frac{\pi}{4}]$ case. To ensure that all four intersection points exist, the projection of the diagonal of A to the major axis of B should be larger than b. Hence we add the constraint $b \le a \cos\alpha + \frac{1}{a} \sin\alpha$. The mathematical program now looks as follows:

$$\begin{aligned} &\textbf{maximize} && \min\left(\frac{(a + b)^2}{2ab(1 + \cos(\alpha))}, \frac{(1 + ab)^2}{2ab(1 + \sin(\alpha))}\right) \\ &\textbf{subject to} && b \le a\cos\alpha + \frac{1}{a}\sin\alpha \\ &&& \pi/4 < \alpha < \pi/2 \\ &&& 1 \le a \le b \end{aligned}$$

Using the mathematical program we can verify, for example via Mathematica 11.2 [118], that the area is at most $\frac{5}{4}$. □

4.2.4 Lemma. $\rho_{\text{TS}}(\textit{OBB}) \ge \frac{5}{4}$

Proof. Consider a point set P with four static points $p_1 = (0, 0)$, $p_2 = (2, 1)$, $p_3 = (0.75, 1)$ and $p_4 = (1.25, 0)$, and point p_5 moving linearly from $(2, 0)$ to $(1.2, 1.6)$; see Figure 4.4. The static points allow two minimal bounding boxes of area 2 and aspect ratio 2: one with orientation 0 (box A) and one with orientation $2\arctan(\frac{1}{2}) \approx 53.13$ degrees (box B). As p_5 is always in A or B, one of these boxes is always optimal. Initially, only A contains p_5 and in the end only box B does. The angles $\arctan(\frac{1}{2})$ (box C) and $\pi/4 - \arctan(\frac{1}{2})$ (box C') give an intermediate box of size $2.5 = \frac{5}{4} \cdot 2$

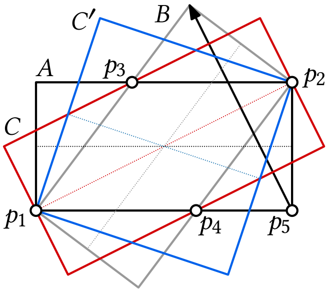


Figure 4.4 Moving points forcing a flip from bounding box A to B. Rotating continuously, a bounding box at least as big as C or C' is required.

on the static points. C is encountered on a counterclockwise rotation from A to B, and C' on a clockwise rotation. Neither C, C', nor any box rotated more towards B contains the initial location of p_5. Similarly, neither C, C', nor any box rotated more towards A contains the final location.

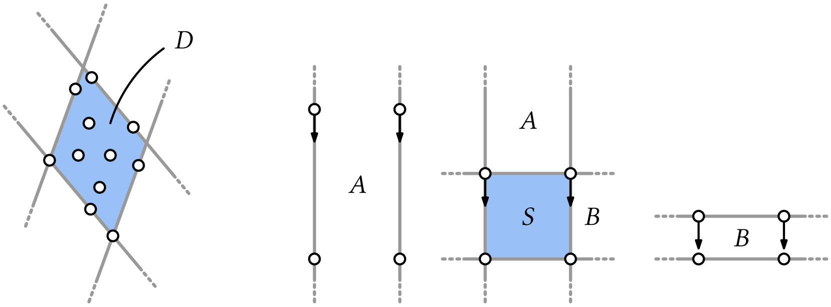
To derive a contradiction, assume a continuously moving OBB exists that achieves a ratio strictly less that $\frac{5}{4}$. This ratio implies that initially the OBB orientation is clockwise between C and C', and at the end of the motion it is not. However, as the assumed OBB moves continuously through O, it must at some point have been in the orientation of C or C'. But this implies a ratio of $\frac{5}{4}$, contradicting our assumption and proving the lower bound. □

4.2.5 Lemma. $\rho_{\mathrm{TS}}(\textit{STRIP}) \leq \sqrt{2}$

Proof. Consider a time t at which there are two thinnest strips A and B of width 1 with different orientations. All points must be contained in the diamond-shaped intersection D of A and B (see Figure 4.5a). If we continuously rotate a strip C from A to B, then at some point the width of C must be at least the length of one of the diagonals of D. To maximize the length of the shortest diagonal, D must be a square with side length 1. Therefore, the width of C is at most $\sqrt{2}$ during the rotation from A to B. □

4.2.6 Lemma. $\rho_{\mathrm{TS}}(\textit{STRIP}) \geq \sqrt{2}$

Proof. Let P consist of four points positioned in a unit square S. There are two thinnest strips A and B for P, each of which is oriented along a different pair of parallel edges of S (see Figure 4.5b). If the orientation of a strip is $\pi/4$ away from A and B, then its width is $\sqrt{2}$.



(a) A configuration having two minimal width strips and overlapping area *D*.

(b) An instance of moving points where the thinnest strip changes orientations. The configuration that leads to the best intermediate solutions is shown in the middle.

Figure 4.5 Upper bound **(a)** and lower bound **(b)** construction for $\rho_{\mathrm{TS}}(\textsc{strip})$.

Now assume that the top points of S are moving along the vertical sides of S, starting from high above. Clearly, at the start of this motion, any strip C approximating the thinnest strip must align with A. At the end of this motion, when the top points align with the bottom points of S, the strip C must align with B. Therefore, the strip C must at some point make an angle of $\pi/4$ with A and B. If x is the distance between the top and bottom points of S, then the width of C at this orientation is $(1 + x)\sqrt{2}/2$. The minimal width is $\min(x, 1)$. It is easy to verify that this ratio is at least $\sqrt{2}$, which concludes the proof. □

4.3 Lipschitz stability

To derive meaningful bounds on the Lipschitz stability ratio, we assume the points move with at most unit speed, and we require that diameter D of $P(t)$ is at least 1 for every time t. This assumption is sufficient to prove a bounded Lipschitz stability ratio for OBB and STRIP. For PC this is not sufficient, as we argue in Section 4.3.3.

To produce a K-Lipschitz stable solution we use a chasing algorithm similar to the generic algorithm introduced in at the start of this chapter. The algorithm maintains a solution over time, and it can rotate towards the optimal solution at every point in time. However, there are two differences from the generic algorithm. First, instead of chasing the orientation of an optimal solution OBB, we chase the orientation of the diametrical pair. Although chasing an optimal shape descriptor would be a more intuitive approach, chasing the diametrical pair is easier to analyze and sufficient to obtain an upper bound on the K-Lipschitz stability ratio for OBB and STRIP. Second, K-Lipschitz stability enforces a maximum speed at which the algorithm can move towards the solution we are chasing. This speed bound depends on parameter K and on how quickly the input changes – faster moving points require faster rotation to achieve the same ratio. Since we assume unit speed for the input, the K-Lipschitz stable solution changes with at most K radians during the movement of the points.

4.3.1 Chasing the diametrical pair

We denote the orientation of the diametrical pair as $\alpha = \alpha(t)$ and then define the *diametric box* of $P(t)$ to be the smallest oriented bounding box with orientation $\alpha(t)$. The dimensions of such a diametric box are defined by the diameter $D = D(t) \geq 1$ of $P(t)$ and the width $W = W(t)$ of the thinnest strip with orientation $\alpha(t)$ covering all points in $P(t)$. Furthermore, let $z = z(t) = W(t)/D(t)$ be the *aspect ratio* of the diametric box with orientation $\alpha(t)$. Finally, our chasing algorithm has orientation $\beta = \beta(t)$ and can change orientation with at most constant speed K. We generally omit the dependence on t if t is clear from the context.

Approach The main goal is to keep orientation β as close as possible to orientation α of an optimal diametric box, specifically within a sufficiently small interval around α. The challenge lies with the discrete flips of α. We must argue that, although flips happen instantaneously, a short time span does not admit many flips over a large angle in the same direction; otherwise we can never keep β close to α with a bounded speed. Furthermore, the size of the interval must depend on the aspect ratio z, since if $z = 0$, the interval around α must have zero size as well to guarantee a bounded approximation ratio.

For the analysis we introduce three functions depending on z: $T(z)$, $H(z)$, and $J(z)$. Function $H(z)$ defines an interval $[\alpha - H(z), \alpha + H(z)]$ called the *safe zone*. We aim to show that, if β leaves the safe zone at some time t, it must return to the safe zone within the time interval $(t, t + T(z)]$. We also define a larger interval $I = [\alpha - H(z) - J(z), \alpha + H(z) + J(z)]$. We refer to the parts of I outside of the safe zone as the *danger zone*. Figure 4.6 shows I at time t and time $t + \epsilon$. Although β may momentarily end up in the danger zone due to discontinuous changes, it must quickly find its way back to the safe zone. We aim to guarantee that β stays within I at all times. Let $E = E(t)$ refer to an endpoint of I. We call $J(z)$ the *jumping distance* and require that $J(z)$ upper bounds how far E can “jump” instantaneously. Intuitively, this requirement should ensure that β does fall outside I after a flip of an optimal diametric box. Let $\Delta E(z, \Delta t)$ denote how far E moves over a time period of length Δt,

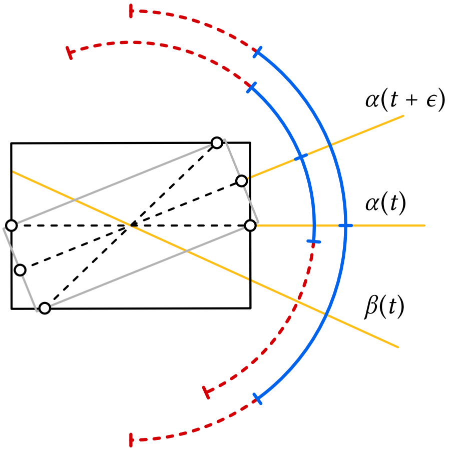


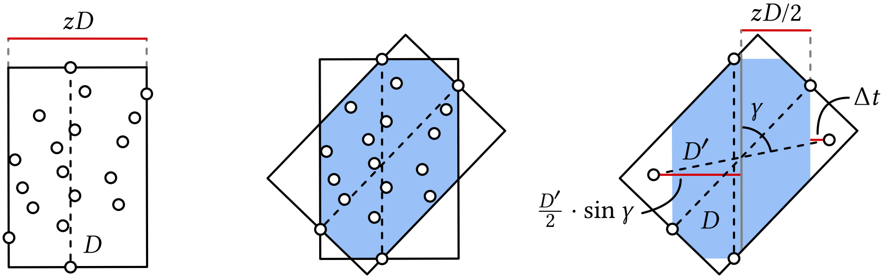
Figure 4.6 Intervals at time t (outer) and time $t+\epsilon$ (inner). The safe and danger zones are indicated in blue and dashed red respectively. Orientations are shown in yellow, for the diametric box at time t and $t+\epsilon$, and for the chasing algorithm at time t.

starting with a diametric box of aspect ratio z. We then require that $\Delta E(z,0) \leq J(z)$. Note that by defining this upper bound, $J(z)$ is defined recursively through E, since by definition $\Delta E(z,\Delta t)$ is upper bounded by how much α, $H(z)$ and $J(z)$ change over time Δt. Hence we need to carefully choose the right function for $J(z)$. For the other functions we choose $T(z) = z/4$ and $H(z) = c \arcsin(z)$ for a constant c (chosen later).

Changes in orientation and aspect ratio To verify that the chosen functions $T(z)$ and $H(z)$ satisfy the intended requirements, and to define the function $J(z)$, we need to bound how much α and z can change over a time period of length Δt. We refer to these bounds as $\Delta\alpha(z,\Delta t)$ and $\Delta z(z,\Delta t)$, respectively. Note that, since the diametric box can change discontinuously, we generally have that $\Delta\alpha(z,0) > 0$ and $\Delta z(z,0) > 0$.

4.3.1 Lemma. *$\Delta\alpha(z,\Delta t) \leq \arcsin(z+\Delta t(2+2z))$ for $\Delta t \leq (1-z)/(2+2z)$.*

Proof. Let D be the diameter at time t; the width of the strip containing all points is zD (see Figure 4.7a). Also, let D' be the diameter at time $t+\Delta t$, and let (p'_1, p'_2) be the diametrical pair at that time, such that the diametrical orientation differs by an angle γ from the orientation at time t (see Figure 4.7c). Note that $\Delta t \geq |D-D'|/2$, otherwise p'_1 and p'_2 must have moved faster than unit speed.



(a) **(b)** **(c)**

Figure 4.7 Illustrations supporting the proof of Lemma 4.3.1: **(a)** A diametric box with dimensions D, zD containing all points. **(b)** If the orientation of the diametrical pair changes, all points must lie in the blue area. **(c)** The orientation can change further in the same direction, after some points move outside the blue area to establish a new diameter.

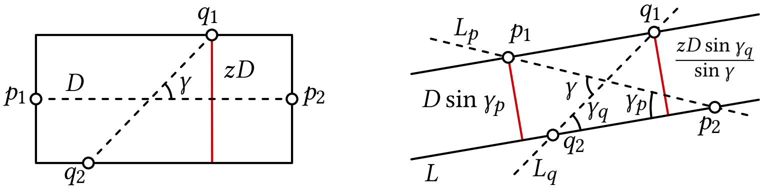
Both p_1' and p_2' must have been in a diametric box at time t. During the time period of length Δt, their movement can cause a flip in the orientation of the diametric box (see Figure 4.7b). As p_1' and p_2' move, possible outside the diametric box at time t, they establish the new diameter D' (see Figure 4.7c). We observe that $\Delta t \geq \frac{D'}{2}\sin(\gamma) - \frac{zD}{2}$.

As Δt is minimized when $D \geq D'$, we can obtain a lower bound for Δt by equalizing $(D - D')/2 = \frac{D'}{2}\sin(\gamma) - \frac{zD}{2}$. We obtain that $\Delta t \geq \frac{D'(\sin(\gamma)-z)}{2+2z} \geq \frac{\sin(\gamma)-z}{2+2z}$. This is equivalent to $\gamma \leq \arcsin(z + \Delta t(2+2z))$, which is well-defined only for $\Delta t \leq \frac{1-z}{2+2z}$. □

4.3.2 Lemma. $\Delta z(z, \Delta t) \leq z - \frac{\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t}{1+2\Delta t}$ *for* $\Delta t \leq \sin(\frac{1}{2}\arcsin(z))/2$.

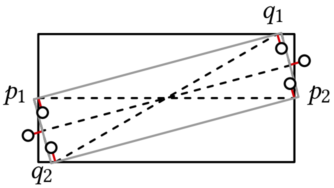
Proof. Let diameter D at time t be realized by the pair of points (p_1, p_2) with orientation α. The width of the diametric box is determined by two points q_1 and q_2; the distance between q_1 and q_2 is at most D (see Figure 4.8a).

To minimize the aspect ratio of the diametric box at time $t + \Delta t$, we need to find a thinnest strip that contains all of p_1, p_2, q_1, and q_2. For the thinnest strip, all four points are on the boundary of the strip in the worst case, and we assume without loss of generality that p_1 and q_1 are on the same side of the strip (same for p_2 and q_2). Consider the following lines: L oriented in the orientation of the thinnest strip (parallel to its boundary), L_p spanned by $p_1 p_2$ and L_q spanned by $q_1 q_2$. Let the angle between L_p and L_q be γ, $\gamma \geq \arcsin(z)$ (see Figure 4.8b). The distance between q_1 and q_2 is then $zD/\sin(\gamma)$. We denote the angle between L and L_p by γ_p, and between



(a) A diametric box with aspect ratio z and points p_1, p_2, q_1 and q_2 at the boundary.

(b) A strip with points located at the boundary, the width of the strip is the maximum of the red lines.



(c) The smallest diametric box for time $t + \epsilon$ and in red the distance the points can travel in Δt to further shrink the aspect ratio.

Figure 4.8 Illustrations supporting the proof of Lemma 4.3.2.

L and L_q by γ_q. We observe that $\gamma_p + \gamma_q = \gamma$, as the orientation of L must bisect the γ angle for the strip to be thinnest.

The width of the strip is $\max(D\sin(\gamma_p), zD\sin(\gamma_q)/\sin(\gamma))$. We show that this width is at least $D\sin(\frac{1}{2}\arcsin(z))$. This is clearly the case if $\gamma_p \geq \frac{1}{2}\arcsin(z)$, so assume the contrary. Since the function $\sin(\gamma - \gamma_p)/\sin(\gamma)$ is increasing, it is optimal to set $\gamma = \arcsin(z)$. But then $zD\sin(\gamma_q)/\sin(\gamma) = D\sin(\gamma_q) > D\sin(\frac{1}{2}\arcsin(z))$. Thus, the width of the thinnest strip is at least $D\sin(\frac{1}{2}\arcsin(z))$.

The diametric box at time t can flip to the orientation of this thinnest strip. However, the points can move to further shrink the aspect ratio of the diametric box. As a result, the width of the diametric box at time $t + \Delta t$ is at least $D\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t$ (see Figure 4.8c). For the same reason, diameter D' can shrink such that $D' \leq D + 2\Delta t$. The final aspect ratio is then $z' \geq (D\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t)/(D + 2\Delta t)$. Since $D \geq 1$, we obtain that $\Delta z(z, \Delta t) \leq z - (\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t)/(1 + 2\Delta t)$. Note that this bound is meaningful only for $\Delta t \leq \sin(\frac{1}{2}\arcsin(z))/2$. □

Jumping distance. We can now derive a valid function for $J(z)$. Recall that we require that $J(z)$ is at least the amount $\Delta E(z, \Delta t)$ that E can move in $\Delta t = 0$ time.

4.3.3 Lemma. *$J(z) = (c + 2)\arcsin(z)$ is a valid jumping distance function.*

Proof. Remember that the amount $\Delta E(z, \Delta t)$ that E can move is upper bounded by the amount of change of α, $H(z)$ and $J(z)$ over time Δt. The interval I grows as z grows, which makes it easier for β to stay inside I. Hence we analyze $\Delta E(z, \Delta t)$, as α changes and z shrinks over time $\Delta t = 0$. Lemma 4.3.1 and Lemma 4.3.2 provide the bounds $\Delta\alpha(z, 0) \leq \arcsin(z)$ and $\Delta z(z, 0) \leq z - \sin(\frac{1}{2}\arcsin(z))$. Combining the above, we get:

$$\begin{aligned}
\Delta E(z, \Delta t) &\leq \Delta\alpha(z, \Delta t) + H(z) - H(z - \Delta z(z, \Delta t)) + J(z) - J(z - \Delta z(z, \Delta t)) \\
\Delta E(z, 0) &\leq \Delta\alpha(z, 0) + H(z) - H(z - \Delta z(z, 0)) + J(z) - J(z - \Delta z(z, 0)) \\
&\leq (1 + c/2)\arcsin(z) + J(z) - J(\sin(\frac{1}{2}\arcsin(z)))
\end{aligned}$$

Since we require that $J(z) \geq \Delta E(z, 0)$, it suffices to show that the following holds: $J(\sin(\frac{1}{2}\arcsin(z))) \geq (1 + c/2)\arcsin(z)$. Using the provided function, we get that $J(\sin(\frac{1}{2}\arcsin(z))) = (c + 2)\arcsin(z)/2$ as required, so the provided function is a valid jumping distance function. □

4.3.4 Corollary. *If β is in I, then $|\alpha - \beta| \leq (2c + 2)\arcsin(z)$.*

Bounding the speed To show that the orientation β stays within the interval I, we argue that over a time period of $T(z)$ we can rotate β at least as far as E. As the endpoint of the safe zone moves at most as fast as E, this implies that if β leaves the safe zone at time t, it returns to it in the time period $(t, t + T(z)]$. Thus we require that $KT(z) \geq \Delta E(z, T(z))$, as β can rotate at most K units when the points move at unit speed. We need to keep up only when the safe zone does not span all orientations, that is, the above inequality must hold only when $H(z) \leq \pi/2$ or $z \leq \sin(\frac{\pi}{2c})$. To find a suitable value for K in Lemma 4.3.7, we choose a specific value $c = 3$.[1] Hence we need to chase α only when $z \leq \sin(\frac{\pi}{6}) = \frac{1}{2}$.

In our proofs we use the following trigonometric inequalities.

4.3.5 Lemma. *The following inequalities hold for* $0 \leq x \leq 1$*:*

1. $\sin(\lambda \arcsin(x)) \leq \lambda x$ *for* $\lambda \geq 1$
2. $\sin(\lambda \arcsin(x)) \geq \lambda x$ *for* $0 < \lambda \leq 1$.

Proof. We first show inequality (1). Let $x = \sin(y)$. We rewrite (1) into $\sin(\lambda y) \leq \lambda \sin(y)$. The derivative with respect of y is $\lambda \cos(\lambda y)$ for the left side and $\lambda \cos(y)$ for the right side. Since $\cos(y) \geq \cos(\lambda y)$ for $0 \leq y \leq \pi/\lambda$ and $\lambda \geq 1$, we get that $\sin(\lambda y) \leq \lambda \sin(y)$ for $0 \leq y \leq \pi/\lambda$. In particular, for $y = \pi/(2\lambda)$ we get that $1 = \sin(\lambda y) \leq \lambda \sin(y)$. Since $\sin(\lambda y) \leq 1$ and $\lambda \sin(y)$ attains its first maximum at $y = \pi/2$, we thus also get that $\sin(\lambda y) \leq \lambda \sin(y)$ for $0 \leq y \leq \pi/2$. Since $x = \sin(y)$ and $\sin(\pi/2) = 1$, the result follows.

For inequality (2), set $x = \sin(\frac{1}{\lambda} \arcsin(y))$. We then rewrite inequality (2) into $y \geq \lambda \sin(\frac{1}{\lambda} \arcsin(y))$. Note that $y = \sin(\lambda \arcsin(x)) \leq 1$, and hence $\frac{1}{\lambda} \geq \sin(\frac{1}{\lambda} \arcsin(y))$. This inequality holds for $0 \leq y \leq 1$, since we can apply (1) using $\frac{1}{\lambda} \geq 1$. As $y = \sin(\lambda\pi/2) < 1$ implies $x = \sin(\frac{1}{\lambda} \arcsin(y)) = 1$, (2) holds for $0 \leq x \leq 1$. □

4.3.6 Lemma. $x \leq \arcsin(x) \leq \frac{\arcsin(a)}{a} x$ *for* $0 \leq a \leq 1$ *and* $0 \leq x \leq a$.

Proof. First note that $\arcsin(x)$ is a convex function for $0 \leq x \leq 1$. Since the derivative of x and $\arcsin(x)$ is 1 at $x = 0$, this directly implies that $x \leq \arcsin(x)$. Furthermore, since $\arcsin(x) = \frac{\arcsin(a)}{a} x$ for $x = 0$ and $x = a$, the convexity of $\arcsin(x)$ also directly implies the second inequality. □

Using the above inequalities, we can show that β can stay in I , if the chasing algorithm is allowed to change with sufficient speed.

---

[1] We could have set $c = 3$ earlier and simplified some of the earlier analysis. We did not do so in order to demonstrate the general technique more clearly, rather than just specifically for this problem.

4.3.7 Lemma. *If* $K \geq 43$*, then* $|\beta(t) - \alpha(t)| \leq 8\arcsin(z)$ *(using* $c = 3$*) for all* t*.*

Proof. Consider a time t when $\beta(t)$ leaves the safe zone. We first argue that $\beta(t')$ will be in the safe zone at some time $t' \in (t, t + T(z)]$. To show this, we need to prove that $KT(z) \geq \Delta E(z, T(z))$ for $z \leq \frac{1}{2}$.

To apply the bounds of Lemmata 4.3.1 and 4.3.2, we must ensure that $T(z) = z/4$ satisfies the bounds for Δt. For Lemma 4.3.1, observe that $(1-z)/(2+2z)$ is decreasing and $z/4$ is increasing, and $(1-z)/(2+2z) = \frac{1}{6} \geq z/4$ for $z = \frac{1}{2}$. For Lemma 4.3.2 we apply Lemma 4.3.5 to show that $\sin(\frac{1}{2}\arcsin(z))/2 \geq z/4$. We thus get the provided bounds on $\Delta\alpha(z, T(z))$ and $\Delta z(z, T(z))$, and as a result a bound on $\Delta E(z, T(z))$. In particular, for $\Delta t \leq T(z)$, we get:

$$\begin{aligned}\Delta E(z, \Delta t) &\leq \Delta\alpha(z, \Delta t) + H(z) - H(z - \Delta z(z, \Delta t)) + J(z) - J(z - \Delta z(z, \Delta t)) \\ &= \arcsin(z + \Delta t(2+2z)) + 8\arcsin(z) - 8\arcsin\left(\frac{\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t}{1 + 2\Delta t}\right).\end{aligned}$$

We have that $z \leq \frac{1}{2}$, $\Delta t \leq z/4 \leq \frac{1}{8}$ and $z + \Delta t(2+2z) \leq \frac{7}{8}$. Then, using the inequalities of Lemma 4.3.6, where $2\arcsin(\frac{1}{2}) \leq 1.05$ and $\frac{8}{7}\arcsin(\frac{7}{8}) \leq 1.22$, and using Lemma 4.3.5, we get:

$$\begin{aligned}\Delta E(z, \Delta t) &\leq 1.22(z + \Delta t(2+2z)) + 8.4z - \frac{4z - 16\Delta t}{1 + 2\Delta t} \\ &\leq 9.62z + 1.22\Delta t(2+2z),\end{aligned}$$

where the last inequality uses the fact that $\Delta t \leq z/4$, and thus $16\Delta t \leq 4z$. Finally, filling in $\Delta t = T(z) = z/4$, we get:

$$\begin{aligned}\Delta E(z, T(z)) &\leq 10.23z + 0.61z^2 \\ &\leq 10.6z && \text{for } z \leq \frac{1}{2} \\ &\leq K\frac{z}{4} && \text{for } K \geq 43.\end{aligned}$$

Finally, we need to argue that $\beta(t)$ does not leave I in the interval $(t, t + T(z)]$. To show this, we need to prove that $K\Delta t \geq \Delta E(z, \Delta t) - J(z)$ for all $\Delta t \in [0, T(z)]$. Using the inequalities above, we have:

$$\begin{aligned}\Delta E(z, \Delta t) - J(z) \leq{}& \arcsin(z + \Delta t(2+2z)) + 3\arcsin(z) \\ &- 8\arcsin\left(\frac{\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t}{1 + 2\Delta t}\right).\end{aligned}$$

We first argue that this function is nondecreasing in z, such that $\Delta E(z, \Delta t) - J(z) \leq \Delta E(\frac{1}{2}, \Delta t) - J(\frac{1}{2})$. For that we consider its partial derivative in z:

$$\begin{aligned}
\frac{\partial(\Delta E(z, \Delta t) - J(z))}{\partial z} &= \frac{3}{\sqrt{1-z^2}} + \frac{1 + 2\Delta t}{\sqrt{1 - (2\Delta t + z + 2\Delta t z)^2}} - \\
&\quad \frac{4\cos(\frac{1}{2}\arcsin(z))}{\sqrt{1-z^2}\sqrt{(1+2\Delta t)^2 - (\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t)^2}} \\
&\geq \frac{3}{\sqrt{1-z^2}} + \frac{1+2\Delta t}{\sqrt{1-z^2}} - \frac{4}{\sqrt{1-z^2}} \frac{\cos(\frac{1}{2}\arcsin(z))}{\sqrt{1 - (\sin(\frac{1}{2}\arcsin(z)) - 2\Delta t)^2}} \\
&\geq \frac{4}{\sqrt{1-z^2}} \left(1 - \frac{\cos(\frac{1}{2}\arcsin(z))}{\sqrt{1 - \sin^2(\frac{1}{2}\arcsin(z))}} \right) \\
&\geq 0
\end{aligned}$$

As a result we can conclude the following:

$$\begin{aligned}
\Delta E(z, \Delta t) - J(z) &\leq \Delta E\left(\frac{1}{2}, \Delta t\right) - J\left(\frac{1}{2}\right) \\
&= \frac{\pi}{2} + \arcsin\left(\frac{1}{2} + 3\Delta t\right) - 8\arcsin\left(1 - \frac{4 + \sqrt{6} - \sqrt{2}}{4 + 8\Delta t}\right)
\end{aligned}$$

Note that this bound is 0 whenever $\Delta t = 0$. It is now sufficient to show that the derivative of this function with respect to Δt is at most K for $0 \leq \Delta t \leq \frac{1}{8}$. Let $a = 4 + \sqrt{6} - \sqrt{2} \approx 5.035$.

$$\begin{aligned}
\frac{\partial \Delta E(\frac{1}{2}, \Delta t)}{\partial \Delta t} &= \frac{6}{\sqrt{4 - (1 + 3\Delta t)^2}} + \frac{64a}{(4 + 8\Delta t)^2 \sqrt{\frac{2a}{4+8\Delta t} - (\frac{a}{4+8\Delta t})^2}} \\
&= \frac{6}{\sqrt{4 - (1 + 3\Delta t)^2}} + \frac{64a}{(4 + 8\Delta t)\sqrt{2a(4 + 8\Delta t) - a^2}} \\
&\leq \frac{6}{\sqrt{4 - (1 + 3\Delta t)^2}} + \frac{16a}{\sqrt{8a - a^2}} \\
&\leq 4.14 + 20.86 \leq 43 \leq K
\end{aligned}$$

Here we used that $3/\sqrt{4 - (1 + 3\Delta t)^2}$ is increasing in Δt and that $\Delta t \leq \frac{1}{8}$. We conclude that $K\Delta t \geq \Delta E(z, \Delta t) - J(z)$ for all $\Delta t \in [0, T(z)]$. Thus, $\beta(t)$ does not leave I in the time period $(t, t + T(z)]$. By repeating this argument whenever $\beta(t)$ leaves the safe zone, we can conclude that $|\beta(t) - \alpha(t)| \leq 8\arcsin(z)$ for all times t. □

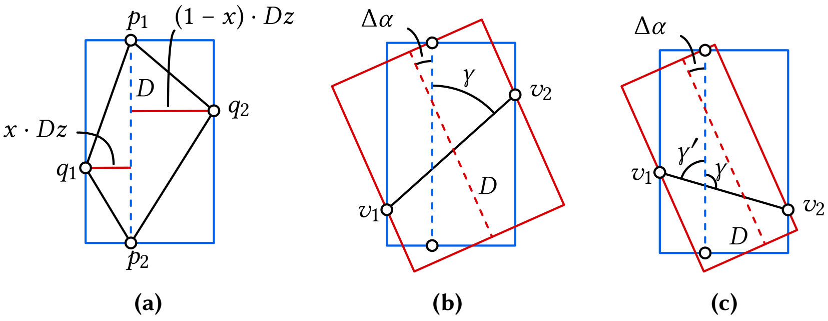


Figure 4.9 Illustrations supporting the proof of Lemma 4.3.8.

4.3.2 Lipschitz stability ratio

What remains is to analyze the approximation ratio of the chasing algorithm for OBB. Corollary 4.3.4 implies that the orientation β of the chasing algorithm is at most an angle $(2c+2)\arcsin(z)$ away from the orientation of the diameter.

4.3.8 Lemma. *If* $|\beta-\alpha| \leq (2c+2)\arcsin(z)$, *then* $f_{OBB}(\beta, P) \leq (4c+6)\min_\phi f_{OBB}(\phi, P)$.

Proof. Assume that at some time t we have a diametric box with diameter D and aspect ratio z, and let (p_1, p_2) be the diametrical pair. The smallest OBB must contain p_1 and p_2 and must hit the sides of the diametric box at, say, q_1 and q_2. The smallest OBB must contain the triangles formed by $\{p_1, p_2, q_1\}$ and $\{p_1, p_2, q_2\}$. Let the diametrical pair be located at a fraction x along the minor axis in the box, then the heights of these triangles with base p_1p_2 are $x \cdot Dz$ and $(1-x)\cdot Dz$ respectively. Their combined area is thus $D^2z/2$ and provides a lower bound for the area of OBB.

Now consider the box of the chasing algorithm, where $\Delta\alpha = |\beta-\alpha| \leq (2c+2)\arcsin(z)$. The major axis (in direction β) has length at most D. Let the minor axis be bounded by two points v_1 and v_2, and γ the angle between the lines spanned by v_1v_2 and by the diametrical pair p_1p_2, on the opposite side of $\Delta\alpha$ with respect to p_1p_2. Let the smallest angle between those two lines be γ'. Note that in the worst case v_1 and v_2 are located on the boundary of the diametric box. Would those points not lie on the boundary, then we can move them there and increase the area of the chasing box. The distance between v_1 and v_2 is therefore bounded by $zD/\sin(\gamma')$. Whenever $\gamma' = \gamma$, the angle between the minor axis of the chasing box and the line through v_1 and v_2 is $\pi/2-\gamma-\Delta\alpha$. Thus, the length of the minor axis is $zD\cos(\pi/2-\gamma-\Delta\alpha)/\sin(\gamma) = zD\sin(\gamma+\Delta\alpha)/\sin(\gamma)$.

However, it can also be the case that $\gamma' = \pi - \gamma$. The angle between the minor axis of the box and the line through v_1 and v_2 is now $\pi/2 - \gamma' + \Delta\alpha$. Analogously, we hence find that the length of the minor axis is $zD\cos(\pi/2 - \gamma' + \Delta\alpha)/\sin(\gamma') = zD\sin(\gamma' - \Delta\alpha)/\sin(\gamma')$. Using $\gamma' = \pi - \gamma$, the length of the minor axis can be simplified to $zD\sin(\gamma + \Delta\alpha)/\sin(\gamma)$, which is the same expression as for $\gamma = \gamma'$.

Since the function $\sin(\gamma + \Delta\alpha)/\sin(\gamma)$ is decreasing in γ, we attain the maximum when $z/\sin(\gamma) = 1$ or $\gamma = \arcsin(z)$. Hence, using $\gamma = \arcsin(z)$, we get that the area of the box of the chasing algorithm is at most $D^2\sin((2c + 3)\arcsin(z))$, which is at most $D^2 z(2c + 3)$ by Lemma 4.3.5. Thus, $f_{\text{OBB}}(\beta, P) \leq (4c + 6) min_\phi f_{\text{OBB}}(\phi, P)$. □

4.3.9 Lemma. *If* $|\beta - \alpha| \leq (2c + 2)\arcsin(z)$*, then* $f_{STRIP}(\beta, P) \leq (4c + 6)\min_\phi f_{STRIP}(\phi, P)$.

Proof. Assume that at some time t we have a diametric box with diameter D and aspect ratio z, and let (p_1, p_2) be the diametrical pair. The width of the diametric box is determined by two points q_1 and q_2.

We first derive a lower bound for the width of the thinnest STRIP; note that this follows the same rationale as in the proof of Lemma 4.3.2. Such a strip must contain the points p_1, p_2, q_1 and q_2. As adding points to a point set can only widen the thinnest strip, we consider just these four points for a lower bound. For the thinnest STRIP, all four points are on the boundary of the strip in the worst case, and we assume w.l.o.g. that p_1 and q_1 are on the same side of the strip (same for p_2 and q_2). Consider the following lines: L oriented in the orientation of the strip (parallel to its boundary), L_p spanned by p_1p_2 and L_q spanned by q_1q_2. Let the angle between L_p and L_q be γ, $\gamma \geq \arcsin(z)$. The distance between q_1 and q_2 is then $zD/\sin(\gamma)$. We denote the angle between L and L_p by γ_p, and between L and L_q by γ_q. We observe that $\gamma_p + \gamma_q = \gamma$, as the orientation of L must bisect the γ angle for the strip to be thinnest. The width of the strip is $\max(D\sin(\gamma_p), zD\sin(\gamma_q)/\sin(\gamma))$. We show that this width is at least $D\sin(\frac{1}{2}\arcsin(z))$. This is clearly the case if $\gamma_p \geq \frac{1}{2}\arcsin(z)$, so assume the contrary. Since the function $\sin(\gamma - \gamma_p)/\sin(\gamma)$ is increasing, it is optimal to set $\gamma = \arcsin(z)$. But then $zD\sin(\gamma_q)/\sin(\gamma) = D\sin(\gamma_q) > D\sin(\frac{1}{2}\arcsin(z))$. Thus, the width of the thinnest strip is at least $D\sin(\frac{1}{2}\arcsin(z)) \geq Dz/2$ by Lemma 4.3.5.

Now consider the strip of the chasing algorithm, with an orientation β differing at most $\Delta\alpha = |\beta - \alpha| \leq (2c + 2)\arcsin(z)$ from the orientation of the diametrical pair. Let the width of the strip be bounded by two points v_1 and v_2, where the angle between the line through v_1 and v_2 and the line through the diametrical pair p_1p_2, opposite of $\Delta\alpha$ with respect to p_1p_2, is γ. Let the smallest angle between those two lines be γ'. Note that, the distance between v_1 and v_2 is $zD/\sin(\gamma')$, since they define the width of the diametric box. When $\gamma' = \gamma$, the angle between the vector perpendicular to the orientation of the strip and the line through v_1 and v_2 is $\pi/2 - \gamma - \Delta\alpha$. Thus, the width of the strip is $zD\cos(\pi/2 - \gamma - \Delta\alpha)/\sin(\gamma) = zD\sin(\gamma + \Delta\alpha)/\sin(\gamma)$.

On the other hand, whenever $\gamma' = \pi - \gamma$, the angle between the vector perpendicular to β and the line $v_1 v_2$ is $\pi/2 - \gamma' + \Delta\alpha$. Analogously, we find that the width of the strip is $zD\cos(\pi/2 - \gamma' + \Delta\alpha)/\sin(\gamma') = zD\sin(\gamma' - \Delta\alpha)/\sin(\gamma')$. Using $\gamma' = \pi - \gamma$, the width of the strip can be simplified to $zD\sin(\gamma + \Delta\alpha)/\sin(\gamma)$, which is the same expression as for $\gamma' = \gamma$.

Since the function $\sin(\gamma + \Delta\alpha)/\sin(\gamma)$ is decreasing in γ, we attain the maximum when $z/\sin(\gamma) = 1$ or $\gamma = \arcsin(z)$. Thus, using $\gamma = \arcsin(z)$, the width of the strip is at most $D\sin((2c+3)\arcsin(z))$, which is at most $Dz(2c+3)$ by Lemma 4.3.5. We finally obtain that $f_{\text{STRIP}}(\beta, P) \leq (4c+6)\min_{\phi} f_{\text{STRIP}}(\phi, P)$. □

By combining Lemmata 4.3.7, 4.3.8, and 4.3.9, we obtain the following bounds on the Lipschitz stability of OBB and STRIP.

4.3.10 Theorem. *The following Lipschitz stability ratios hold for* OBB *and* STRIP*, assuming diameter $D(t) \geq 1$ for all t and points move with at most unit speed:*

- $\rho_{\text{LS}}(\textit{OBB}, 43) \leq 18$,
- $\rho_{\text{LS}}(\textit{STRIP}, 43) \leq 18$.

4.3.3 Lipschitz stability of principal component

The chasing algorithm in the previous section does not work for the first principal component. Specifically, our scale normalization, which requires the diameter to be at least one at any point in time, does not help. This can intuitively be attributed to the optimization function of PC. Rather than being defined by some form of extremal points, f_{PC} is determined by variance: although the diameter may be large, many close points may still largely determine the first principal component. We formalize this via the lemma below. It implies that requiring a minimal diameter is not sufficient for a chasing algorithm with bounded speed to approximate PC. The proof is inspired by the construction in [32] that shows the ratio on the areas between a bounding box aligned with the principal components and the optimal oriented bounding box can become infinite.

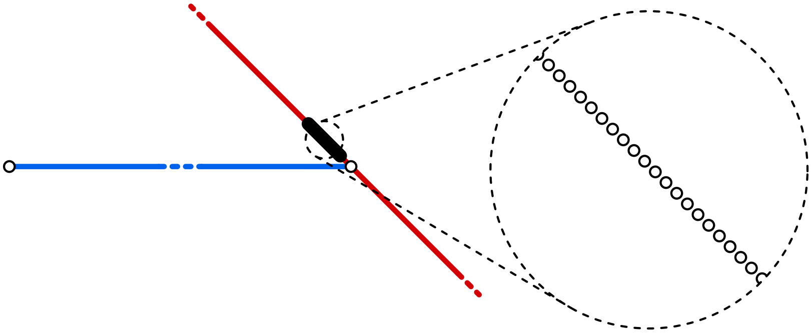


Figure 4.10 The two points connected by the blue line form a diametrical pair for the whole point set. The dense set of points located arbitrarily close to the right point can move around it in an infinitesimally short amount of time. Because the point set is so dense, the orientation of the first principal component (in red) follows it regardless of how far the leftmost point is placed.

4.3.11 Lemma. *For any constant K, there exists a point set $P(t)$ with minimum diameter 1 at all times, such that any shape descriptor that approximates the optimum of f_{PC} must move with speed strictly greater than K.*

Proof sketch. Consider the point set P containing two points that lie on a horizontal line and form a diametrical pair for P. All other points in P form a dense subset P' that is not collinear with the two points that form the diametrical pair, but are located very close to only one of the two points. See Figure 4.10 for the construction.

Remember that the optimization function for the first principal component minimizes the sum of squared distances from the points to the line. The dense subset P' contains so many points that any line that differs more than ϵ from the orientation of the line l through P', has a significantly larger sum of squared distances from the points in P to l. Hence PC follows P' regardless of the position of the two points forming the diametrical pair.

For any constant K, the points in P' can be placed and moved in such a way that in an infinitesimally small time frame, they can move around one of the points of the diametrical pair and change the orientation of PC by more than K. Thus any shape descriptor that approximates the optimum of f_{PC} must also change its orientation by more than K. □

▸ 4.4 Conclusion

We studied the topological and Lipschitz stability of three common orientation-based shape descriptors. Although stateless algorithms cannot achieve topological stability, we proved tight bounds on the topological stability ratio for state-aware algorithms. Our Lipschitz analysis focuses on upper bounds, showing that a chasing algorithm achieves a constant approximation ratio for a constant maximum speed, for OBB and STRIP. Since PC is not sufficiently scale invariant under the conditions in our analysis, it is left open whether the same algorithm can work for PC. Furthermore, the analysis techniques for the Lipschitz stability upper bounds have the potential to also work for other problems that could be approached via a chasing algorithm. Finally, it remains open to establish whether lower bounds exist that are stronger than those already given by our topological stability results.

Chapter 5

Spatially and Temporally Coherent Visual Summaries

When exploring large time-varying data sets, visual summaries are a useful tool to identify time intervals of interest for further consideration. A typical approach is to represent the data elements at each time step in a compact one-dimensional form or via a one-dimensional ordering. Such 1D representations can then be placed in temporal order along a time line. There are two main criteria to assess the quality of the resulting visual summary: *spatial quality* – how well do the 1D representations capture the structure of the data at each time step, and *stability* – how coherent are the 1D representations over consecutive time steps or temporal ranges? For example, the 1D representation for dynamic graphs should capture the network structure, under insertions and deletions. For hierarchical data the 1D representation should capture the implied tree structure, under value changes and insertions and deletions. Finally, for moving entities in 2D, the 1D representation should capture the spatial proximity under continuous movement of the entities. We focus on techniques that create such visual summaries for entities moving in 2D. Previous work has considered only the creation of 1D orderings, using spatial subdivisions and clustering techniques. In contrast, taking inspiration from the previous chapter, we propose to use methods based on principal component analysis and other dimensionality-reduction techniques to compute stable and spatially informative 1D representations. These more general 1D representations provide the user with additional visual cues

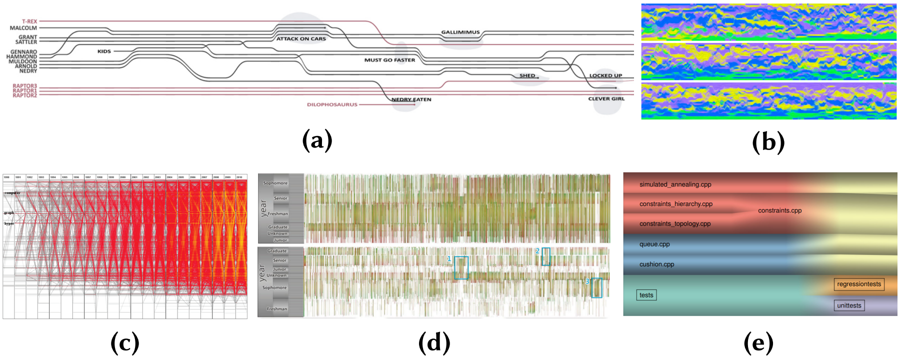


Figure 5.1 Examples of visual summaries in existing work. **(a)** Dynamic StoryLine graph [112], **(b)** Let It Flow for dynamic graphs [30], **(c)** Parallel Edge Splatting [25], **(d)** Extended Massive Sequence Views [110] and **(e)** Temporal Treemaps [69].

describing the spatial structure of the data, and naturally imply also a 1D ordering. To make dimensionality-reduction techniques suitable for visual summaries, we introduce stable variants of principle component analysis, Sammon mapping, and t-SNE in this chapter. Our Stable Principal Component method explicitly allows a user-configurable trade-off between the spatial quality and stability.

Visual summaries have been extensively used in various applications. For example, there are a variety of methods to handle time-varying graphs, such as Parallel Edge Splatting [25] (Figure 5.1c) and Extended Massive Sequence Views [110] (Figure 5.1d), that show the temporal evolution by drawing the graph at each time step in a narrow vertical strip. Similarly, Temporal Treemaps [69] (Figure 5.1e) encode hierarchies via (essentially) one-dimensional intervals and show the temporal evolution by placing these intervals consecutively along a line. Also Storyline Visualizations [73, 112] (Figure 5.1a) use a compact representation at each time step (essentially a pixel per protagonist); these representations must be coherent between consecutive time steps and as such trace a trajectory for each actor.

Arguably the most compact representation for one time step is a 1D ordering of the data objects. Such an ordering directly translates to a grid-based visualization of associated attributes, where a vertical strip of n grid cells encodes n objects. In principle, any aspect of the data can be used to create the ordering. For example, MotionRugs by Buchmüller *et al.* [23] (Figure 5.2d) computes orders from spatial

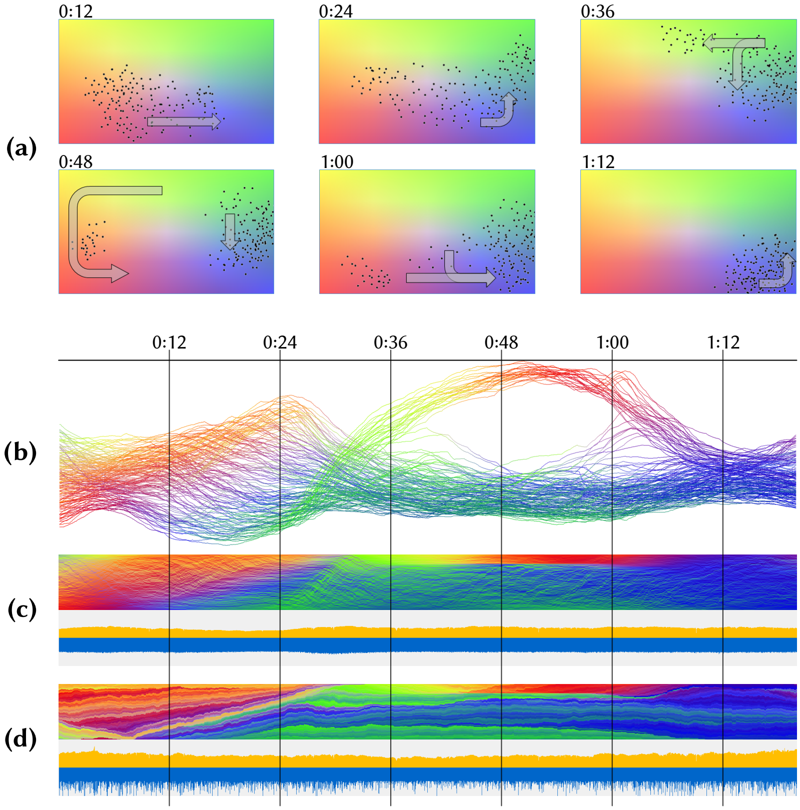


Figure 5.2 Visual summaries of fish movement data. **(a)** Sample frames of movement data; background indicates spatial coloring. **(b)** Our Stable Principal Component method translates moving entities into MotionLines; color indicates spatial location. **(c)** MotionLines compacted via ordering into a MotionRug [23]. Chart below indicates spatial quality (yellow) and stability (blue) per time step; high values indicate low quality. **(d)** Unstable ordering using Hilbert curve, note the artifical split between the green locations.

locations for entities moving in 2D, whereas Cui *et al.* [30] (Figure 5.1b) use node degree to order dynamic graph data.

Previous work [23, 57] which computes 1D orderings for moving entities in 2D focuses on spatial subdivisions or clustering techniques. All are designed to maintain spatial relations in the sense that entities which are close in the 1D ordering are also close in the 2D input. In contrast, we want to ensure high spatial quality by communicating all data well. Specifically, data points which are close in 2D need to also be close in the 1D representation. Furthermore, we would like to adequately represent temporal patterns in the data through visual summaries: small changes in the data should lead to small changes in the 1D representations. One way of achieving this, is by making sure that the ordering in 1D does not change much between consecutive representations. We propose to use actual dimensionality-reduction techniques to compute meaningful visual summaries for entities moving in 2D: the resulting visual summaries should both be stable and represent the data well.

Our stable dimensionality-reduction methods compute not only an ordering, but a more general 1D representation which provides the user with additional visual cues describing the spatial distribution of the data. We connect the position of entities in this 1D representation across time-steps to form so-called *MotionLines* (see Figure 5.2), which illustrate the potential of more general 1D representations. Furthermore, we have augmented the visual summaries in Figure 5.2 with spatial quality and stability information. These augmentations serve to show the quality of 1D representations; they can help a user in identifying interesting segments in the data, as well as gauging the reliability of the visual summary.

Formal problem statement Our input is a set $P = \{p_1, \dots, p_n\}$ of n point objects moving in the plane. We sample their positions at T consecutive time steps. That is, each object p_i is a sequence of T locations or points in the plane. We use $p_i(t)$ to denote the location of p_i at time t, $1 \leq t \leq T$, and, correspondingly, $P(t)$ to denote the complete point set at time t. A visual summary S of P is a sequence of 1D representations of the points in P, one per time step. We denote the representation at time t by S_t. A 1D representation naturally implies a 1D ordering. Thus, $S_t(p_i)$ denotes either a 1D position or the rank of point object p_i in the 1D representation at time t. The quality of a visual summary S is determined by two criteria:

Spatial quality How well does S_t capture the spatial structure of $P(t)$? We characterize the spatial structure via local neighborhoods: we say that a 1D representation has high spatial quality if points that are spatially close in the input are also close in the representation.

Stability How consistent are the 1D representations over time? Here we can consider absolute changes between representations or changes in local neighborhoods, as captured by nearest neighbors in the ordering. Both types of measures can be considered for consecutive time steps or over temporal ranges.

Clearly, a visual summary that uses the same representation for all time steps is maximally stable. However, the spatial quality of this representation will typically be low. Conversely, optimizing spatial quality for each time step in isolation tends to result in unstable summaries which make it more difficult for the user to track objects. Hence, we need algorithms that incorporate the temporal dimension into solving each time step.

Contributions and organization To make dimensionality-reduction techniques suitable for visual summaries, in Section 5.1 we introduce stable variants of Principle Component Analysis (PCA), Sammon mapping, and t-SNE. Our Stable Principal Component method [SPC] is explicitly parametrized for stability, allowing a tradeoff between spatial quality and stability. We also describe a stable Clustered Principal Component [CPC] method, particularly suited for data sets that exhibit clear clusters.

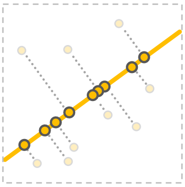
Since previous work has focused exclusively on 1D orderings, so does our quantitative evaluation. In Section 5.2 we survey a representative set of state-of-the-art ordering methods, which we compare against in our experiments. In Section 5.3 we discuss the quality metrics we use to capture spatial quality and stability. In particular, for spatial quality we use the so-called *Key Similarity* measures proposed by Guo and Gahegan [57] that characterize spatial proximity via k nearest neighbors. For stability we consider two different types of measures: absolute or neighborhood changes in the linear order. For absolute stability we use the number of *Jumps* and *Crossing* as proposed by Buchmüller *et al.* [23]. We model neighborhood changes between orders via changes in the k nearest neighbors, and again use the Key Similarity measures by Guo and Gahegan [57]. In Section 5.4 we report on the results of our experiments, which use both real-world and synthetic data. We can conclude that our stable dimensionality-reduction techniques outperform spatial subdivisions, space filling curves, and clustering techniques on stability, without sacrificing spatial quality or efficiency. We close in Section 5.5 with discussion of our results and directions for possible future work.

5.1 Stable dimensionality reduction

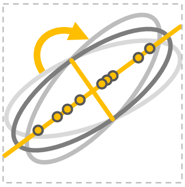
We describe four stable variants of well-established dimensionality-reduction techniques – PCA, Sammon mapping, and t-SNE – which we use to create 1D representations for moving entities. The dimensionality reduction performed by PCA consists of a projection onto a single vector, using the principal component for projections of high spatial quality. To develop a stable PCA-based method, we can interpolate the projection vector between time steps, instead of using the principal component at each time step. This enables an explicit user-configurable trade-off between spatial quality and stability (see Section 5.1.1). We extend this method to incorporate the presence of clusters in the data, preventing clusters from interleaving in 1D representations. Sammon mapping and t-SNE rely on local search heuristics (gradient descent) to compute the 1D representations. In Section 5.1.2 we show how we can improve stability by choosing initial solutions for the local search heuristic.

5.1.1 Stable principal component analysis

PCA was first introduced by Pearson [87] and can be used for dimensionality reduction to 1D by projecting points onto the first principal component: a vector in the direction along which the point set has most variance. Projecting onto this vector maximally preserves spatial relations in the original point set.



In the previous chapter, we analyzed the trade-off between spatial quality and stability of orientation-based shape descriptors, including PCA, from a theoretical point of view. The analysis shows that the principal components of a set of moving points in 2D exhibit unstable behavior when the point set is not stretched, that is, the variance along the first and second principal component is similar. Our approach leverages this result by explicitly enforcing stability when the point set is not stretched. The intuition behind this approach is as follows. If the variance along the first principal component is clearly higher than the variance along the second principal component, the direction is very discriminative: the point set is clearly stretched in this direction and sorting the points along this vector tends to lead to high spatial quality. If this is not the case, then the point set is “round” and the spatial quality is roughly equivalent for other directions as well. Our goal is to smoothly interpolate the projection vector in those cases.



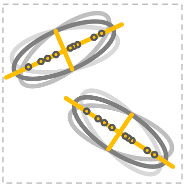
[SPC$_\sigma$] Stable Principal Component To create a stable version of PCA, we use the optimal direction (first principal component) as projection vector for any t where $P(t)$ is stretched, as well as for the first and last time step. For all time steps in between (when the point set is not stretched) we linearly interpolate the orientation of the projection vector. We use a parameter σ ($0 \leq \sigma \leq 1$) to control when we consider a point set as stretched or not.

Concretely, the Stable Principal Component algorithm is implemented as follows (see Algorithm 1 for an overview). To determine if a point set is stretched, we use the corresponding eigenvalues λ_1 and λ_2 of the first and second principal components, respectively. If $\lambda_2/\lambda_1 > \sigma$, then the point set is stretched, and otherwise it is not. For the time steps t where the point set is stretched (including $t = 1$ and $t = T$), we simply compute the first principal component as projection vector $\textsc{pv}[t]$. Note that $-\textsc{pv}[t]$ is equally good as projection vector, but results in a mirrored 1D representation. To avoid flipping, we therefore use the direction ($\textsc{pv}[t]$ or $-\textsc{pv}[t]$) that is most consistent with $\textsc{pv}[t-1]$ (computed using the dot product). For time steps t where the point set is not stretched we also first compute the (consistent) first principal component. We use these vectors to keep track of the signed angle α describing how the orientation of the first principal component has changed since the last time t' the point set was stretched (or $t' = 1$). Once we reach another time t'' where the point set is stretched (or $t'' = T$), we can linearly interpolate the orientation of the projection vector for all times t with $t' < t < t''$. Although linear interpolation of orientations is not unique in general, we can use the accumulated signed angle α to uniquely interpolate the projection vector. Finally, we project the point sets for all time steps onto the computed projection vectors $\textsc{pv}[t]$.

Since the eigenvalues and principal components of n points in 2D can be computed in $O(n)$ time, it is easy to see that the entire algorithm runs in $O(nT)$ time. The explicit trade-off between spatial quality and stability can be configured via parameter σ. If σ is set to a value close to 1, the focus of the algorithm is on spatial quality, and only when the point set is very “round”, stability will be enforced; $\sigma = 1$ eliminates interpolation and always uses the first principal component in every time step. However, if σ is set closer to 0, the focus will be on stability and even for moderately stretched point sets, linear interpolation can occur, thereby sacrificing spatial quality for stability; $\sigma = 0$ causes one interpolation, from the first principal component at $t = 0$ to the first principal component at $t = T$. Hence, by tuning σ, the preferred trade-off between spatial quality and stability can be obtained.

Algorithm 1 StablePrincipalComponent(P, σ)

```
Input: Point set $P$ over $T$ time steps, and $\sigma \in [0, 1]$
Output: Visual summary $S$ for $P$
Set PV[1] to the first principal component vector (PC) for $P(1)$
Set $t'$ to 1 and $\alpha$ to 0
for $t = 2$ to $T$ do
    Set PV[$t$] to PC of $P(t)$ and compute eigenvalues $\lambda_1, \lambda_2$
    Add the signed angle between PV[$t$] and PV[$t-1$] to $\alpha$
    if $\lambda_2/\lambda_1 \leq \sigma$ or $t = T$ then
        for $t_s = t' + 1$ to $t - 1$ do
            Set PV[$t_s$] to PV[$t'$] rotated over $\alpha \cdot \frac{t_s - t'}{t - t'}$
        Set $t'$ to $t$ and $\alpha$ to 0
for $t = 1$ to $T$ do
    Define $S[t]$ by projecting $P(t)$ onto PV[$t$]
return $S$
```



[CPC$_\sigma$] Clustered Principal Component If a point set is strongly clustered, then we would expect a 1D representation of this point set with high quality to separate the different clusters. However, in the Stable Principal Component algorithm described above, two clusters may be interleaved if their projections happen to overlap. Therefore, we also propose the Clustered Principal Component algorithm, which is essentially a hybrid between SPC$_\sigma$ and a clustering algorithm (such hybrids have also been explored in [114]).

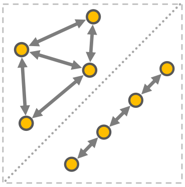
Intuitively, this algorithm performs SPC$_\sigma$ on the separate clusters. More specifically, for every frame we first perform Complete Linkage Clustering [53] [CLC] on the point set, resulting in a hierarchical clustering. CLC is agglomerative and repeatedly merges the two clusters that are closest, where the distance between two clusters is determined by the farthest two points in different clusters. To obtain a partitioning of the points, we stop the process when the closest distance between clusters doubles with respect to the previous iteration. While this heuristic suffices to find salient clusters in our data sets, many other techniques exist to find a good partitioning in a hierarchical clustering [86].

Next, we perform SPC$_\sigma$ on the individual clusters, with two small adaptations, resulting in projection vectors PV$_C[t]$ for a cluster C. First, we end the linear interpolation PV$_C[t]$ when the clustering changes and there is no longer a cluster with exactly the same points as C. We basically treat the time step as $t = T$, and interpolate to the last frame where cluster C still existed. Second, it is no longer straightforward to determine the most consistent direction ($\mathrm{PV}_C[t]$ or $-\mathrm{PV}_C[t]$) for a cluster when the clustering changes. Here we use the projection vector used by the majority of the points in the cluster at time $t-1$ to determine the most consistent direction.

To find the global 1D representation at time t, we use the first principal component of the whole set P to project the cluster centers. The 1D representations of points within a cluster are then placed around the projection of its cluster center. Although this approach may still result in overlap between two clusters when using 1D coordinates as representation, we can easily separate the clusters if we use a 1D ordering: first, we order the clusters according to their cluster centers, and then we order the points within a cluster according to their internal ordering.

5.1.2 Gradient-descent methods

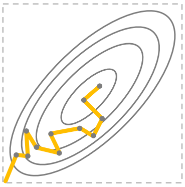
Many dimensionality-reduction techniques define a cost function to describe the spatial quality of the resulting representation, and aim to minimize that function. For example, Sammon mapping uses a function that measures how well distances are preserved, while t-SNE uses a function that measures how well local neighborhoods are preserved. Since finding the global minimum of such a cost function is typically hard, they often use local search heuristics (usually gradient descent) to find a good solution. In our experiments we consider two such dimensionality-reduction techniques: Sammon mapping and t-SNE. There are other dimensionality-reduction techniques, such as MDS [71] and Isomap [107], but based on their cost functions we believe that they give similar results (in fact, in the Euclidean plane, classical MDS is equivalent to PCA). We first recall Sammon mapping and t-SNE for a static point set, before explaining our adaptations for improved stability.



[SAM] Sammon Mapping Sammon mapping [98] aims to preserve distances. Let d_{ij} denote the Euclidean distance between points p_i and p_j, denote the resulting (1D) coordinates by x_i, and let $\delta_{ij} = |x_i - x_j|$. Sammon mapping computes coordinates x_i, attempting to minimize this cost function:

$$C = \frac{1}{\sum_{1\le i<j\le n} d_{ij}} \sum_{1\le i<j\le n} \frac{(d_{ij}-\delta_{ij})^2}{d_{ij}}$$

The cost C is then minimized using a gradient descent starting from a random initial solution.



[SNE] t-Distributed Stochastic Neighbor Embedding The goal of t-SNE [111] is to preserve local neighborhoods in the result of the dimensionality reduction. Again, let d_{ij} denote the Euclidean distance between points p_i and p_j. Similarities between points are captured by a probability distribution:

$$\mathcal{P}_{j|i} = \frac{\exp\left(-\frac{d_{ij}^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{d_{ik}^2}{2\sigma_i^2}\right)}$$

The values σ_i are chosen depending on the predefined *perplexity* κ (see [111] for details); in our experiments we use $\kappa = 40$. We further define $\mathcal{P}_{ij} = \frac{1}{2n}(\mathcal{P}_{j|i} + \mathcal{P}_{i|j})$ and we set $\mathcal{P}_{ij} = 0$ if $i = j$. Denote the resulting (1D) coordinates by x_i, and define δ_{ij} as

$$\delta_{ij} = \frac{(1 + |x_i - x_j|^2)^{-1}}{\sum_{k \neq l}(1 + |x_k - x_l|^2)^{-1}}$$

The cost function is defined by the Kullback-Leibler divergence as:

$$C = \sum_{i \neq j} \mathcal{P}_{ij} \log \frac{\mathcal{P}_{ij}}{\delta_{ij}}$$

Finally, the cost C is again minimized using a (momentum-based) gradient descent[1] starting from a random initial solution.

Stability improvements Both Sammon mapping (SAM) and t-SNE (SNE) start the gradient descent with a random initial solution, which may result in low stability over time. To improve the stability of both algorithms, we initialize them with the solution of the previous time step, resulting in two stable versions, **[SAMp]** and **[SNEp]**. The rationale is that, if the local minimum found in the previous time step still exists, but has slightly shifted, then this approach will likely find this local minimum again rather than a random other local minimum.

Recently, Rauber *et al.* [91] described Dynamic t-SNE: a more explicit way of making t-SNE stable over multiple time steps. Their approach performs a global optimization

---

[1] We tried using the existing implementation at `https://github.com/lejon/T-SNE-Java` to compute the t-SNE mapping. This implementation uses approximations to speed up the computation, which lead to artifacts in our results. We therefore implemented the default version of t-SNE ourselves. See supplementary material for more details.

over all time steps simultaneously, using a separate copy of each point for each time step. They enforce temporal coherence by adding a term to the optimization function depending on the distance between two copies of the same point at consecutive time steps. For two reasons we were not able to include this algorithm in our experiments. First, it is very slow. The paper reports a running time of about 6 minutes per time step. Although a single time step of our data consists of only hundreds of points, we consider thousands of time steps, making their algorithm prohibitively slow. Second, the implementation of Dynamic t-SNE rarely gives meaningful output when run on our data[2], which changes much faster than the data experimented on in [91]. We further believe that Dynamic t-SNE would converge slowly on a time-varying data set with many time steps: it would take at least T gradient descent iterations for frames that are T time steps apart to affect each other. Since t-SNE is already known to converge quite slowly, the combination may simply require too many iterations to obtain a reasonable solution. Thus, Dynamic t-SNE exacerbates the usual downsides of t-SNE, namely black-box parameter tuning and slow convergence.

▸ 5.2 Computing 1D orderings

Previous work by Guo and Gahegan [57] and Buchmüller *et al.* [23] which computes 1D orderings for moving entities in 2D uses spatial subdivisions or clustering techniques. For our experiments, we chose algorithms that performed best in their experiments. We also include a baseline algorithm [FXD] that is solely focused on stability. Figure 5.3 shows examples of orderings generated by some of the algorithms, including dimensionality-reduction methods, for one time step of our data.

[FXD] Fixed Order This algorithm outputs the same arbitrary linear order for every time step and hence serves as reference baseline for our experiments. With FXD, each horizontal line always represents the same moving entity.

▸ 5.2.1 Spatial subdivisions

Several well-known linearization approaches, which are primarily used for spatial-indexing applications, are based on the principle of iterating through some spatial subdivision. These approaches encompass tree data structures and space-filling curves. We focus on four established, representative techniques from this area, though many variations exist; see [74] for an overview.

---

[2] The implementation often gives NaN as output. The authors [106] have verified that this is a known problem with the implementation.

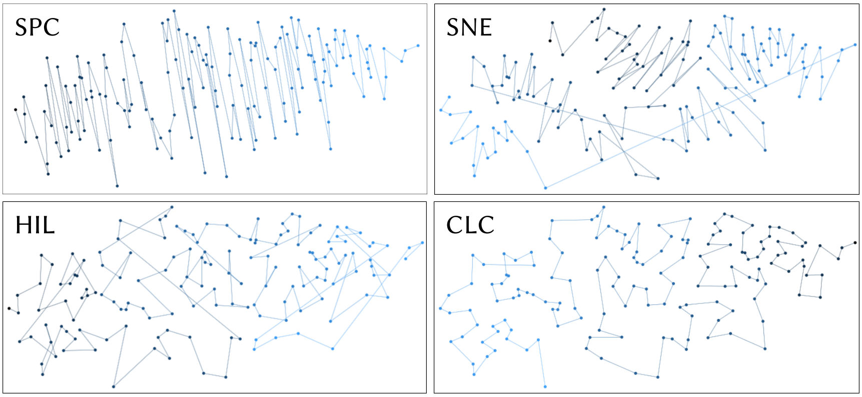
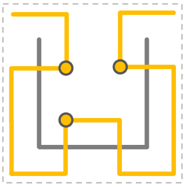
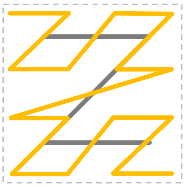
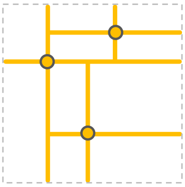


Figure 5.3 Orderings using dimensionality reduction (SPC and SNE), space-filling curves (HIL) and clustering (CLC).

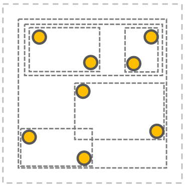




[HIL] Hilbert curve and [ZOR] Z-order curve The Hilbert curve [61] is a continuous space-filling curve. It can be applied to cover a spatial region in arbitrary precision by repeating the construction pattern recursively. A set of points in space can then be linearized by sampling the curve and noting the order in which the points are encoded on the curve. Another representative of space-filling curves is the Z-order curve, which differs from the Hilbert curve in its geometrical construction pattern resembling a Z shape, where the space is partitioned in four quadrants in the order NW, NE, SW, SE (see bottom figure on the right). Both approaches differ in neighborhood retention and construction complexity, as Lu and Ooi describe [74]. Guo and Gahegan [57] found that Hilbert curves avoid long jumps better than the Z-Order curve, which in turn outperforms Hilbert curves in the average of the compared metrics. Since both produce visually different outcomes, we include both strategies in the comparison.



[PQR] Point Quadtree Quadtrees [40] partition space recursively in four parts, until each part contains only a single point. Consequently, sparse areas cause fewer splits than dense areas. Standard quadtrees divide the space in equal parts, while Point Quadtrees split at an input point and thus potentially unevenly in terms of area. To derive the 1D ordering, a depth-first tree-iteration strategy is used; given the neighborhood structure in the tree, this is more suitable than a breadth-first strategy. See [39] for details on tree-iteration strategies. The standard quadtree essentially reflects a Z-Order curve linearization if the same quadrant iteration is applied. Hence, we use the point quadtree variant which produces different orderings due to the intermittent partition.

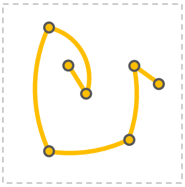


[RTR] R-tree In R-Trees [58] objects are stored recursively in minimum bounding rectangles (MBR). Each MBR can hold at most a predefined number of objects, thus ensuring a minimum fill. In comparison to quadtrees, more complex balancing is necessary, recomputing the MBRs, when the object limit is reached. Note that MBRs can overlap. Again, a depth-first iteration strategy is used to order points in an R-Tree.

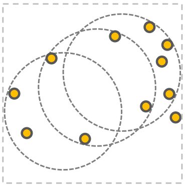
5.2.2 Clustering

Another method to compute a linear order from a point set is to first compute a hierarchical clustering on the point set, and then order the points in such a way that clusters stay together. Algorithms of this type are defined by two aspects: (1) how the points are clustered, and (2) how the linear order is computed from the clustering. We use the following method to compute the linear order from the clustering.

The hierarchical clustering is represented by a tree with the individual points stored in the leaves. We aim to order to leaves of such a tree without changing the cluster structure: we can change only the order of the children of any internal node. We follow the algorithm by Bar-Joseph *et al.* [6] to compute the order that minimizes the length of the path formed by visiting the input points in that order. The algorithm uses dynamic programming to efficiently find the optimal order for every subtree placing two specific points at the first and last position in the order.



[CLC] Complete Linkage Clustering Initially, every point is considered as a separate cluster to be hierarchically merged in a bottom-up fashion [53]. We do so by repeatedly merging the closest two clusters, until we obtain a single cluster. Distance between clusters is measured as the distance between their farthest points. While CPC also uses this clustering method, it differs as SPC is executed within each cluster. For CLC we follow the procedure described above, resulting in a linear order which relies only on the clustering.



[SNN] Shared Nearest Neighbors This clustering algorithm [65] works the same as CLC, but it uses a metric different from Euclidean distance to measure the dis(similarity) between two points. For two points p and q, we first count the number x of points that are in the set of k nearest neighbors for both p and q. We then define the *shared nearest neighbor* (SNN) distance between p and q as $1/(x + 1)$. In case of ties in SNN distance, we use Euclidean distance to break ties. In our experiments we use $k = 10$.

5.3 Metrics

In this section we discuss the quality metrics we use to capture spatial quality and stability. Since our quantitative evaluation focuses exclusively on 1D orderings (to be able to compare to previous work), we will from now on discuss only 1D orderings.

5.3.1 Spatial quality

Spatial quality measures the correspondence between $P(t)$ and the linear order S_t. We capture this by considering the local neighborhood of a point, as characterized by its nearest neighbors. One way to measure changes in local neighborhoods is using an evaluation of dimensionality reduction via persistent homology as introduced by Rieck and Leitte [93]. However, we choose not to use this type of measure. While this approach is more recent than the measure we are using, it does not compare to older results, it is more complex, and most importantly it is an indirect approach. Hence, we use the *Keys Similarity* measures as described by Guo and Gahegan [57] to directly measure the changes in nearest neighbors.

To simplify notation, we omit dependencies on time step t, as the metrics consider each time step in isolation. Thus, P denotes a point set in the plane, and S denotes a linear order. Let $n(i, j) \in P$ denote the j^{th} nearest neighbor of p_i in P, for each j with $1 \leq j \leq k$ for some constant k. We use $r(i, j)$ to denote the neighbor rank in S between p_i and $n(i, j)$. However, the difference in rank $|S(n(i, j)) - S(p_i)|$ is not unique. There are two neighbors at rank difference 1, two at rank difference 2, until we reach one end of a linear order. To avoid arbitrariness, we do not break ties but rather consider each pair with the same rank difference to have the same value for $r(i, j)$. Thus, there are two nodes with $r(i, j) = 1$ (rank difference 1), two nodes with $r(i, j) = 3$ (rank difference 2), etc.

Generally, Keys Similarity at time t is then defined as

$$KS(P,S) = \frac{\sum_{p_i \in P} \sum_{j=1}^{k} w(i,j) \cdot r(i,j)}{\sum_{p_i \in P} \sum_{j=1}^{k} w(i,j)},$$

where $w(i,j)$ denotes the weight or importance of maintaining the j^{th} nearest neighbor of p_i at time t – note that these weights need not be the same at every time step. We use two variants of Keys Similarity, as introduced by Guo and Gahegan [57].

[KSra] Rank-weighted Keys Similarity We define $w(i,j) = 1/j$ inversely proportional to the rank of the j^{th} nearest neighbor in P, such that maintaining the closest neighbors is considered more important than the more distant neighbors. This gives the following metric, where H_k is the k^{th} harmonic number:

$$KSra(P,S) = \frac{\sum_{p_i \in P} \sum_{j=1}^{k} r(i,j)/j}{\sum_{p_i \in P} \sum_{j=1}^{k} 1/j} = \frac{\sum_{p_i \in P} \sum_{j=1}^{k} r(i,j)/j}{n \cdot H_k}$$

[KSdi] Distance-weighted Keys Similarity We define $w(i,j) = 1/\|p_i - n(i,j)\|$ inversely proportional to the Euclidean distance, such that maintaining close neighbors is considered more important than distant neighbors. In contrast to KSra, this variant does not treat neighbors at (nearly) identical distances differently.

$$KSdi(P,S) = \frac{\sum_{p_i \in P} \sum_{j=1}^{k} r(i,j)/\|p_i - n(i,j)\|}{\sum_{p_i \in P} \sum_{j=1}^{k} 1/\|p_i - n(i,j)\|}$$

Other facets Our metrics focus on combinatorial aspects of the position of the point objects. Spatial structure in general knows many other facets, such as distances and directions between points, as well as density. For 1D representations, such as projections into a single dimension, distances and density and can factor into spatial quality. However, a linear order inherently does not lend itself to represent such concepts, as only neighbor rank can be read from an ordering.

5.3.2 Stability

Stability or temporal coherence measures the similarity between consecutive orders in S. In our evaluation, we use the following three measures for stability. The first two are based on absolute changes in the order and match the measures used by

Buchmüller *et al.* [23] to evaluate MotionRugs. The latter uses neighborhoods, based on the concepts by Guo and Gahegan [57].

We aim to compare the similarity between two linear orders, S_t and S_{t+1} for each t with $1 \le t < T$. We could easily use the same metrics to compare nonconsecutive orders, but this provides little insight for such inherently sequential data. To consider the stability over a temporal range $[t, t']$, we use standard summary statistics (e.g., average, minimum, or maximum) over all consecutive pairs.

[JMP] Jump distance We quantify the jump distance for a single point object p_i as the difference between its ranks in the two orders, that is, $|S_t(p_i) - S_{t+1}(p_i)|$. The jump distance between two orders is then the sum over all jump distances for each point object.

$$JMP_t(P, S) = \sum_{p_i \in P} |S_t(p_i) - S_{t+1}(p_i)|$$

The value for $JMP_t(P, S)$ lies between 0 (perfectly stable) and $n(n-1)/2$ (complete inversion of the order).

[CRS] Crossings Whereas JMP penalizes any change in the order, many points moving up together may not constitute much change. Instead we may count the number of inversions or crossings in the order, that is, the pairs p_i, p_j for which $S_t(p_i) < S_t(p_j)$ and $S_{t+1}(p_i) > S_{t+1}(p_j)$. The metric $CRS_t(P, S)$ lies between 0 (perfectly stable) and $n(n-1)/2$ (complete inversion of the order).

Buchmüller et al. [23] also use Kendall's τ coefficient to evaluate stability. We choose to omit this, as it is equivalent to $1 - 2 \cdot CRS_t(P, S)/(n(n-1)/2)$. That is, Kendall's τ is the same as CRS up to normalization to the range $[-1, 1]$.

[KSte] Temporal Keys Similarity We may also take the same approach as for spatial similarity and consider the similarity of local neighborhoods in both orders. As distances are not inherently meaningful in the combinatorial order and simply correspond to ranking differences, we use only the rank-weighted version of Keys Similarity. Also for this metric $KSte_t(P, S)$, we do not break ties in either order, but rather give them the same rank.

5.4 Experimental evaluation

In this section we report on quantitative experiments which compare our stable dimensionality-reduction techniques (Section 5.1) to existing 1D ordering techniques

(Section 5.2). For each algorithm, we assess the spatial quality and stability of the computed visual summaries according to the metrics discussed in Section 5.3. The parameter for SPC is explored after settling on the most effective measures for spatial quality and stability. Besides reporting on the introduced metrics, we also compare all algorithms on their run-time measurements.

The visual summaries in this section use a 2D RGB colormap introduced in [24]. As shown in Figure 5.2, the 1D representations in the visual summaries (b)-(d) visualize the data points as pixels colored according to their 2D position in (a). In visual summaries of high spatial quality, data points that are close in 2D should be close in 1D, hence similar colors should end up close to each other. Only when points separate into different clusters in 2D, should we see sharp contrasts in 1D, such as in Figure 5.2 between 0:48 and 1:00. Furthermore, in stable summaries the neighborhoods do not change much in the 1D representations, and thus the colors should smoothly change over time.

▸ 5.4.1 Data

For comparability, we use the same data as MotionRugs [23] along with two synthetic data sets, one generated using Netlogo [115] and another generated with the well known Reynolds model [92]. The first data set tracks 151 fish of the Notemigonus crysoleucas species (Golden shiner). Golden shiner fish live in large groups called “shoals”, moving in coordination at almost any given time. The 151 fish were tracked optically while moving through a 2.1m by 1.2m shallow water tank, thus avoiding movement in the third dimension. The tank did not feature any obstacles or hindrances besides the side walls. Different movement patterns can be observed in the data, which allows us to test quality in different situations. Among these patterns are uniform group movements, partial and complete changes of direction, circular movement patterns, splitting off in separate clusters, and changes in group density, speed, and acceleration. This data set is quite large, hence we use two excerpts of 2000 frames of movement, which were recorded at a rate of 25 frames per second. For each frame, the spatial coordinates of each fish are recorded in a Cartesian coordinate system. In the first excerpt, the fish first move around the boundary of the tank and finally enter a so-called milling formation, moving in a circular shape. During this movement the fish always form a single cluster. In contrast, the second excerpt shows the fish splitting in separate clusters, as can be seen in Figure 5.2a.

In addition to analyzing the first excerpt in full, we also zoom in on a part of the movement data, which triggers so-called “phantom splits” [23] for certain ordering methods, most notably HIL, PQR, or SNEp (see Figure 5.6). The shoal of fish appears to split, but this is purely an artifact of the method and not reflecting the data.

The second data set is generated with Netlogo using the Flocking model [116] from the openly available Models Library within the Netlogo application. Minimal adaptations were made to the model to ensure the boundaries of the canvas do not wrap around, and the trajectories of the moving entities could be extracted easily.

The third dataset, which we use to demonstrate clustered movement, is generated by an adaptation from the well known Reynolds model [92], where between the movers of the three visible clusters only repulsion forces apply, but no attraction, keeping the clusters separate. This data set was generated using code by Piljek [88].

The results of the experimental evaluation are split by data set and excerpt, to show how the algorithms are influenced by the properties of data. Sections 5.4.3 to 5.4.6 consider these results per data set.

5.4.2 Running Time

We implemented and executed all algorithms in Java 11 on a workstation with two Intel Xeon E5-2687W CPUs at 3.10GhZ, 16 Cores, 128GB Ram and an NVidia Quadro M600 GPU, running Windows 10. We measure running times only for computing the orderings excluding reading input, color mapping and rendering. The running times range from a few milliseconds for the Z-Order curve (ZOR) to just over 8 hours for t-SNE (SNE). General observations include comparably good performance for the subdivision methods (ZOR, HIL, PQR, RTR), with values under one second (see Tables 5.1–5.4). Only SPC variants are on par with this speed. While CPC is slower than SPC, it is still considerably faster than the remaining methods.

5.4.3 Experimental evaluation fish data set (excerpt 1)

Figures 5.6 and 5.8 show visual summaries for all algorithms for the first excerpt of the fish data set. The MotionRugs are accompanied by a visualization of the mean KSdi and KSte values for each frame, cut off slightly above the mean values of most algorithms. This ensures that the differences between the average behavior of the algorithms becomes visible at a glance. Table 5.1 provides summary statistics over all time steps and for each metric. Below, we first discuss spatial quality and stability statistics separately, along with a discussion on phantom splits. We follow up with an exploration into the effects of the parameter value on the outcome of SPC and finally consider the trade-off between spatial quality and stability for all methods.

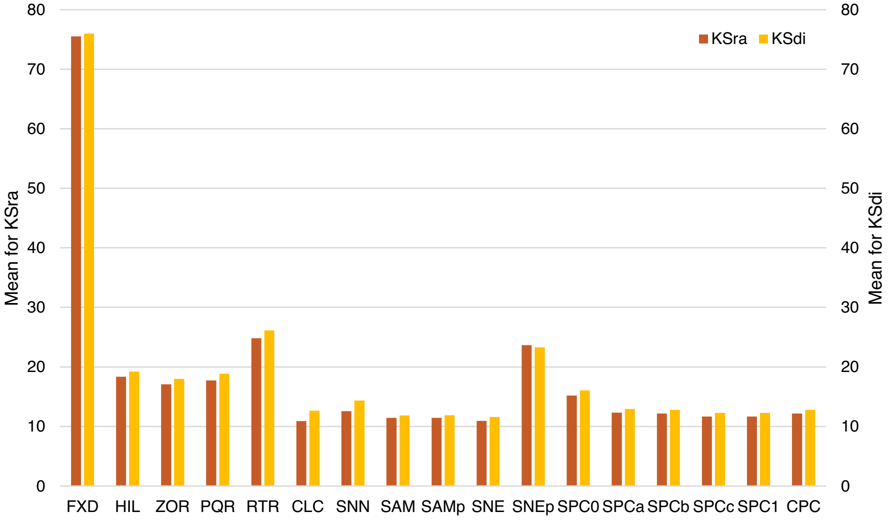


Figure 5.4 Spatial-quality metrics: mean KSra (left) and KSdi (right) for all algorithms over all frames of the first excerpt of the fish data set.

Spatial quality Figure 5.4 compares the spatial-quality measures KSra and KSdi, as measured on all algorithms for the first excerpt of the fish data set. For both measures lower values indicate higher spatial quality. Overall, we see that the KSra measurements are slightly lower for all algorithms, except SNEp where KSdi has a minimal edge over KSra. As expected, FXD achieves the worst spatial quality. Furthermore, SNEp and the algorithms using spatial subdivisions are outperformed by the clustering algorithms, and other dimensionality-reduction techniques. It is interesting that the spatial quality of SAMp is only marginally worse than SAM, while the spatial quality of SNEp is clearly inferior to SNE. This shows that even though the same adaptations were made to improve stability, these methods are affected in very different ways. Comparing the spatial quality of SPC and CPC to the algorithms that perform best on spatial quality, we see that SPC and CPC both achieve comparable spatial quality. The choices for parameter σ of SPC on the first excerpt of the fish data set are 0, 1, and variables $a = 0.35$, $b = 0.53$, $c = 0.78$. The choice for the intermediate values a, b and c is different for the various data sets and will be justified in the parameter exploration. For this data set we choose $\sigma = 0.53$ for CPC. This is a parameter value that according to the parameter experiment performs well on both spatial quality and stability. Due to the strong correlation of both spatial quality measures, we focus on only KSdi in the remainder.

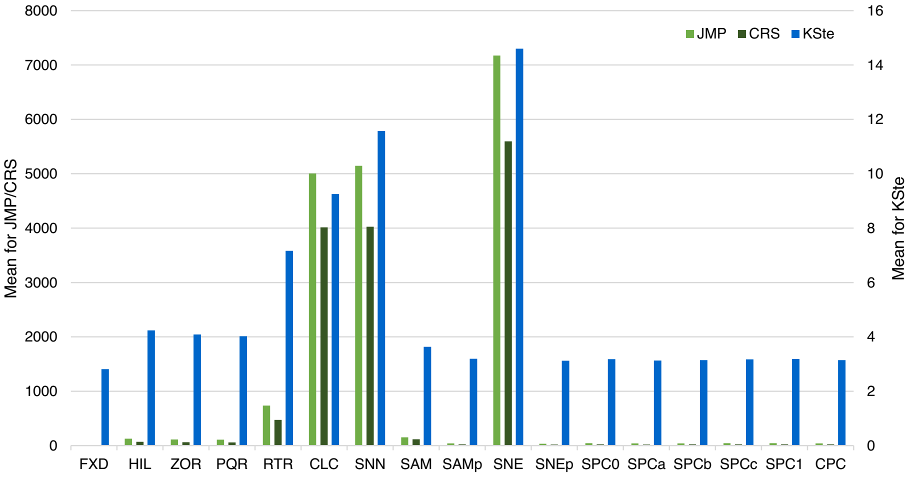


Figure 5.5 Stability metrics: mean JMP, CRS (left axis), and KSte (right axis) for all methods over all frames of the first excerpt of the fish data set.

Stability Figure 5.5 compares the stability measures: JMP, CRS and KSte. While JMP and CRS measure absolute changes between orders, KSte captures changes in local neighborhoods. For each measure lower values indicate higher stability. We see that CRS results in lower values than JMP, which is expected: two entities can jump to different positions in the next frame without crossing, but they cannot cross each other without jumping. Looking at the individual strategies, for FXD the result is again obvious: all measures are at their minimum. While JMP and CRS are generally low, CLC, SNN and SNE show very high numbers. Those three algorithms also perform worst according to the KSte metric. Another outlier that performs poorly on KSte is RTR, which also performs comparatively poorly on JMP and CRS. Of the remaining algorithms, the spatial subdivisions perform worst on KSte. The SAM, SAMp and SNEp algorithms, the SPC variants and CPC show similar and very low mean values of KSte. Again similar results across all data sets, with some notable differences for the clustered data set. Here lower values are measured for the absolute changes for clustering techniques, and CPC performs relatively better than on other data sets. Both of these results are to be expected. Finally, SNE is less of an outlier for this data set, as SNN performs worse on KSte. Again, we observe a strong correlation between the metrics, and thus consider only KSte in the remainder.

Phantom splits Let us briefly consider the visual summaries as highlighted in Figure 5.6. Each frame is represented by a horizontal column of pixels, where each fish corresponds to a pixel. The pixels are colored according to the position of the fish in 2D, as shown in Figure 5.2. The ordering method clearly defines the resulting visual patterns. An anomaly can be identified in the subdivision methods (HIL, ZOR, PQR, RTR) and SNEp, the so-called phantom split pattern [23]. These visual summaries suggest that the shoal of fish somehow splits, but this is not the case. Such patterns are hence undesirable, as they convey false information. Other algorithms do not seem to be prone to these kind of visual artifacts or generally produce too fuzzy visual results for such patterns to appear. Some algorithms, such as CLC and SNE, result in such cluttered visuals despite having good spatial quality. This clutter is a consequence of instability: the summaries fail to convey patterns over time despite individual frames being objectively good.

SPC parameter We now investigate the parameter σ of SPC and its effect on the results. We run SPC for 101 different values for σ from 0 and 1 with increments of 0.01. As discussed before, we use KSdi to measure the spatial quality of the visual summaries, and specifically we use the mean over all frames. For stability we use the mean as well as the max KSte to quantify stability. While mean KSte captures cohesion over time, max KSte should be low to prevent visual artifacts from disrupting temporal patterns. The results for the first excerpt of the fish data set are shown in Figures 5.7 and 5.8. Note that the highest plotted value of σ is 0.95, while the lowest is 0.29. Values above and below these extremes are identical to results with 0.95 and 0.29 respectively. The σ values indicated by labels in the figures are chosen as representatives, and used in our other experiments.

Overall, we see an inverse relation between stability and spatial quality. Values of σ closer to 1 result in better spatial quality, while values closer to 0 sacrifice some spatial quality for more stability. This is to be expected, as SPC_1 always projects the fish to the first principal component; this will likely lead to the best spatial quality that can be achieved for any parameter value.

As σ is decreased, SPC increasingly uses interpolated lines for projection instead. This interpolation smooths changes in angle of the line, but the projection reflects spatial relations less accurately as a result. When σ drops below 0.30, the interpolation happens purely between the first and last frame of the data set. Contrary to expectation, this negatively affects both spatial quality and stability: the first principal component rotates both clockwise and counterclockwise at varying speeds, not matching the uniform interpolation over such a long time period; as a result, the

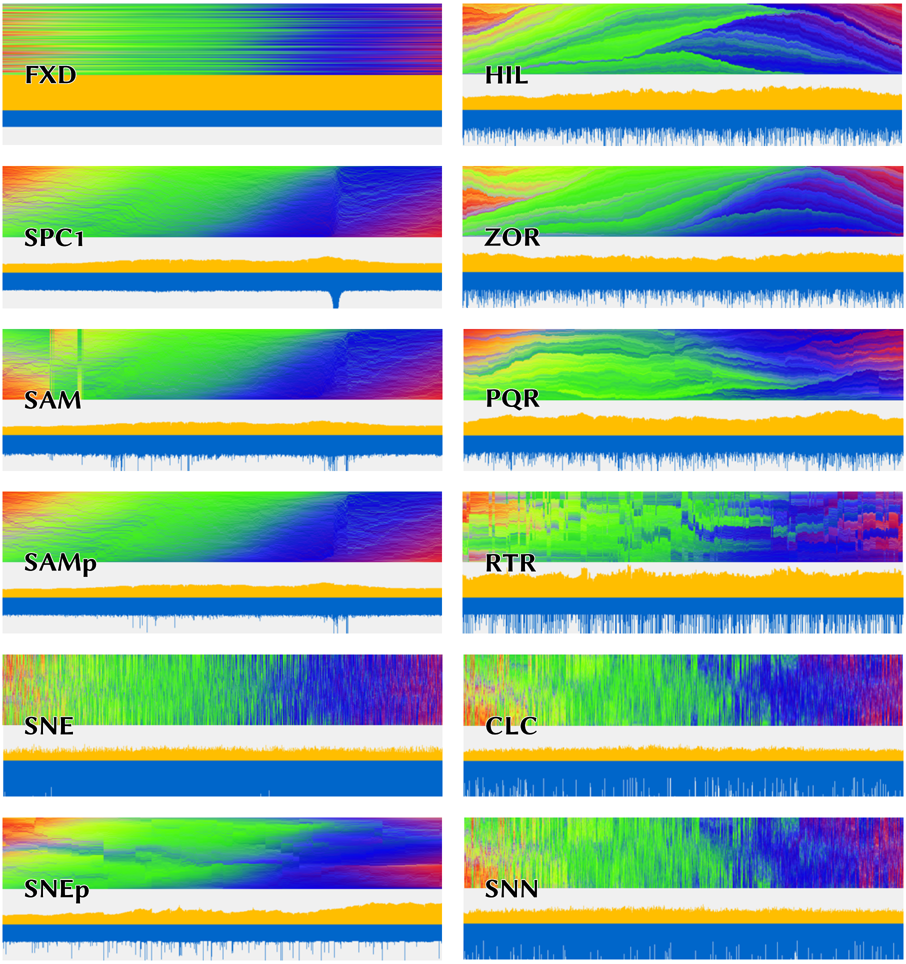


Figure 5.6 Visual summaries for the first excerpt of the fish data set, focussed on so-called phantom splits for SNEp, HIL, ZOR, PQR and RTR. Below each summary we show KSdi (yellow) and KSte (blue), capped at 37.5 and 6.25, respectively.

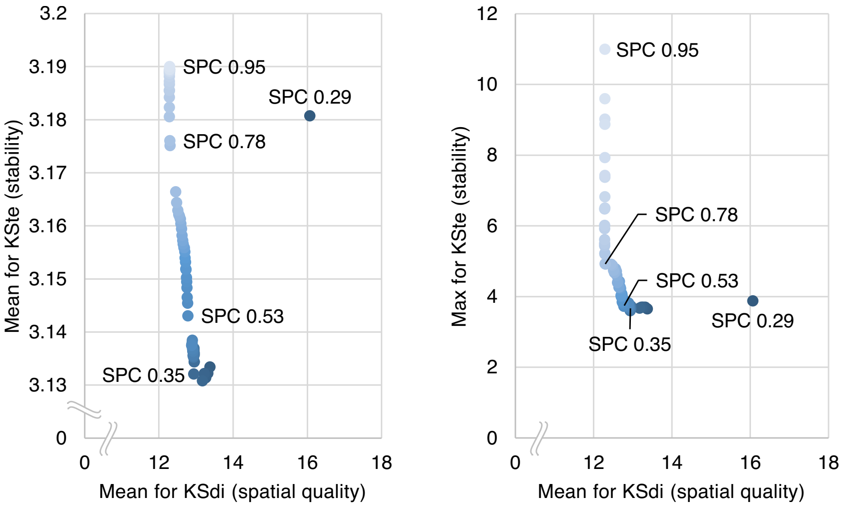


Figure 5.7 A comparison between the mean and mean (left) as well as max and mean (right) for KSte and for KSdi respectively, for uniformly distributed σ of SPC_σ on the first excerpt of the fish data set.

interpolated lines do not correspond at all to the first principal components, neither in angles nor in rotation direction. This mismatch in angles leads to poor spatial quality per frame, while the mismatch in rotation direction also decreases stability.

Finally, we explicitly show the effects of changing σ on the resulting visual summaries using Figure 5.8. In this figure, we visualize spatial quality in yellow and stability in blue. On the left we show how much every data point contributes to the measures, with brighter colors indicating worse spatial quality/stability, while darker colors show placements in the ordering of high spatial quality or stability. On the right the aggregated values over all points in a time step are visualized in a histogram. Figure 5.8 specifically shows an instability that occurs in the first excerpt of the fish data set, using the same intermediate values for σ as before. When the first principal component is used without introducing stability ($\sigma = 1$), we see a short burst of instability, along with slightly elevated measurements in spatial quality. As σ decreases and more stability is introduced by interpolating the direction of the first principal component over larger time frames, we see that the instability is distributed over more frames and decreases. The spatial quality is not negatively impacted by this, until σ becomes too low: when we interpolate over too many frames at once, spatial quality will drastically deteriorate, as seen for $\sigma = 0$.

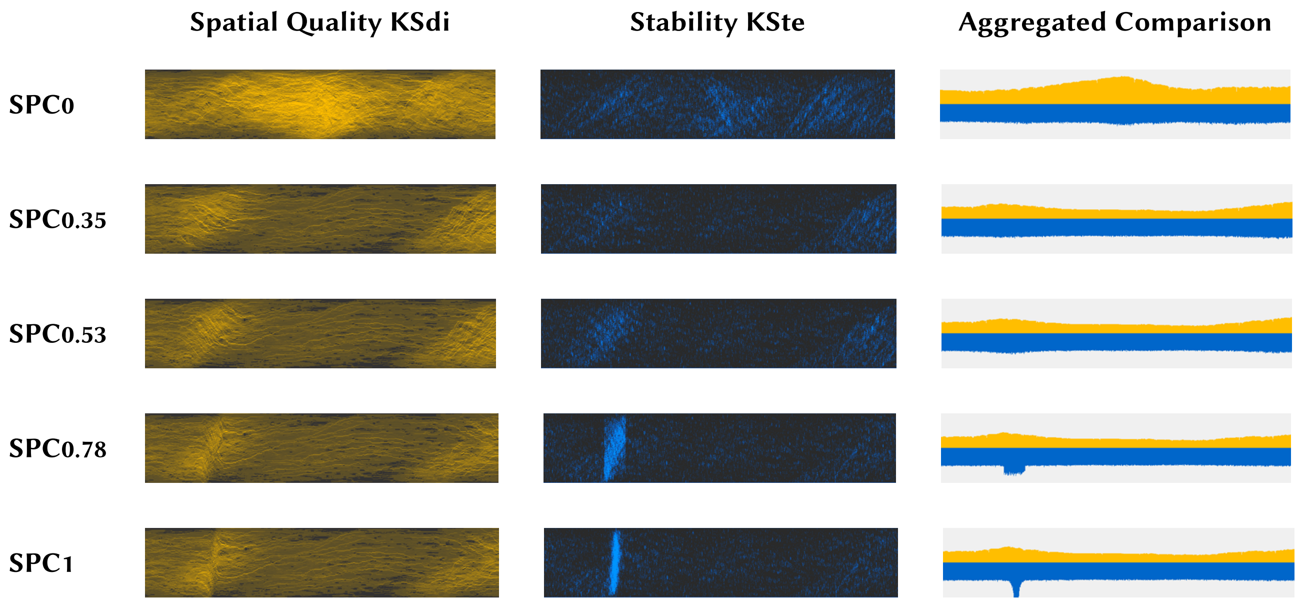


Figure 5.8 Visual summaries for the first excerpt of the fish data set, focussed on instabilities to show σ affects interpolation in SPC. The left and middle summaries show how each data point adds to the KDdi (yellow) and KSte (blue) measures. Brighter colors indicate worse spatial quality and stability. The measures are aggregated per frame in bar charts on the right.

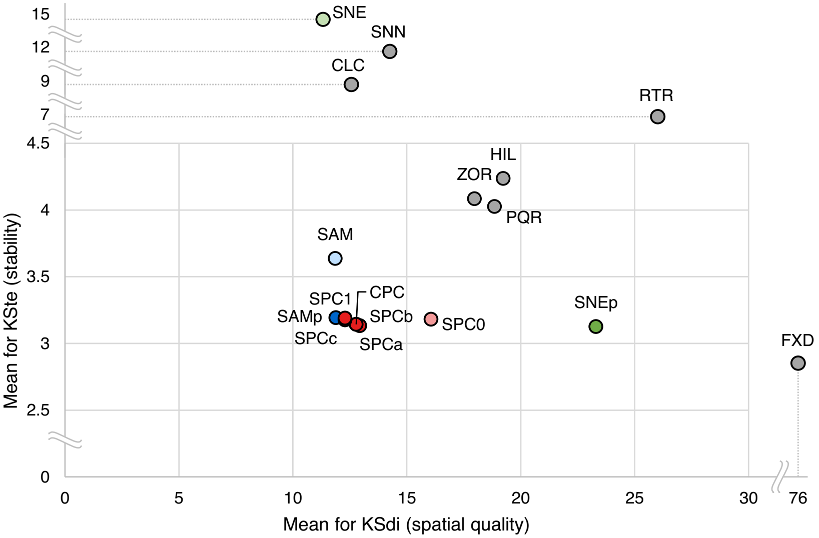


Figure 5.9 A comparison between the mean for KSte and for KSdi for all algorithms on the first excerpt of the fish data set.

Trade-offs Our main goal is to investigate the trade-off between spatial quality and stability. Figure 5.9 shows a scatterplot on the means of KSdi and KSte of all algorithms. Since lower values indicate better quality for both, methods in the bottom-left corner perform well on both aspects. In both figures SPC variants and CPC are colored in shades of red, SAM variants in blue and SNE variants in green. The "best" variants have fully opaque colors, while the variants that are unstable or have worse spatial quality have a lighter shade.

Methods based on spatial subdivisions (ZOR, HIL) and space-filling curves (PQR, RTR), albeit fast to compute, perform poorly on spatial quality and stability. The clustering methods (CLC, SNN) as well as SNE, on the other hand, perform well on spatial quality, but exhibit poor stability. These methods are also slow to compute.

The fixed order (FXD) and SNEp are on the other extreme, having good stability, but very poor spatial quality. Furthermore, the strong influence of initialization for t-SNE stands out. When initialized with random coordinates (SNE), the spatial quality is very good, but the stability is extremely poor. On the other hand, initializing t-SNE with the embedding of the previous time step (SNEp) greatly improves stability, but spatial quality suffers greatly.

That leaves SAM, SAMp, SPC variants, and CPC, which perform well on both aspects. We note that SAM and SAMp perform very similarly on spatial quality (with a difference of 0.03 between KDdi values), but SAMp performs significantly better in terms of stability. SPC variants also strike a good balance between spatial quality and stability. All SPC variants have slightly worse spatial quality than SAM variants, but also improve stability. However, recall that SPC is significantly faster to compute than the Sammon mapping algorithms SAM and SAMp. Finally, CPC performs equal to SPC on the same value for σ, which is expected since the fish stay grouped in a single cluster: on a single cluster CPC and SPC are equivalent.

It is also important for stability to be consistently low, to avoid visual artifacts and ensure that visual patterns in the summary point to actual movement patterns. As such, bursts of high instability are undesirable. We hence also consider the maximum value of KSte; see Figure 5.10 for a scatterplot. The overall composition remains similar, but differences in stability are highlighted. Note that SAM, SAMp and SNEp are deteriorating with respect to other methods; we can also see clear bursts of instability in Figure 5.6 for these methods.

Interestingly, SAMp performs worse here on stability than SAM, unlike for other data sets. This shows that, although initializing the gradient descent with the solution of the previous time step generally improves stability, there is no guarantee that it will always do so. There may be outliers, as is the case here.

Figure 5.10 also highlights stability differences between SPC variants. SPC_1 always uses the first principal component, which can behave erratically for round point sets, decreasing stability drastically in those situations. However, SPC overcomes this problem for other values of σ by interpolating over these bursts of instability. Indeed, SPC is largely unaffected for lower parameter values, having the smallest standard deviation overall (see Table 5.1).

Overall, stable methods such as SAMp, SNEp, and SPC for parameter values lower than 1 perform very well in terms of average and worst-case stability, while only marginally sacrificing spatial quality in the case of SAMp and SPC. SPC does so at a fraction of the computational cost necessary for more complex dimensionality-reduction techniques. Considering all the above, we conclude that stable dimensionality-reduction methods are the best approach for computing visual summaries of time-varying data.

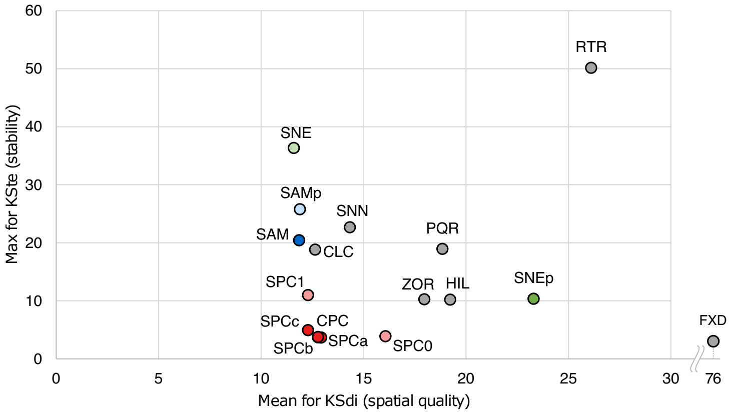


Figure 5.10 A comparison between the max for KSte and the mean for KSdi for all algorithms on the first excerpt of the fish data set.

Table 5.1 Statistics on the first excerpt of fish data set

Fish (excerpt 1)			FXD	HIL	ZOR	PQR	RTR	CLC	SNN
Spatial Quality	**KSra** ●	min	72.17	11.56	12.37	11.36	16.53	7.43	8.45
		max	79.37	25.17	26.33	25.69	32.76	15.60	19.92
		mean	75.53	18.34	17.06	17.72	24.82	10.90	12.56
		stdev	1.44	2.84	2.72	2.65	2.16	1.35	1.40
	KSdi ●	min	53.85	9.27	11.80	9.07	14.65	8.36	8.96
		max	88.45	30.10	31.30	28.26	35.30	18.95	22.89
		mean	75.99	19.23	17.97	18.85	26.11	12.64	14.34
		stdev	2.14	3.51	3.27	3.08	2.61	1.69	1.78
Stability	**JMP** ●	min	0.00	0.00	2.00	0.00	0.00	0.00	0.00
		max	0.00	616.00	576.00	3032.00	7624.00	11400.00	11400.00
		mean	0.00	126.06	111.71	108.31	736.84	5006.78	5147.02
		stdev	0.00	95.39	84.41	127.33	1258.74	4065.33	3970.93
	CRS ●	min	0.00	0.00	1.00	0.00	0.00	0.00	0.00
		max	0.00	369.00	401.00	1677.00	5647.00	11316.00	10993.00
		mean	0.00	70.22	61.05	59.13	472.04	4015.75	4026.70
		stdev	0.00	56.81	49.18	72.68	850.10	3531.41	3320.76
	KSte ●	min	2.81	2.81	2.83	2.81	2.81	2.81	2.81
		max	2.81	10.18	10.26	18.94	50.16	18.83	22.67
		mean	2.81	4.24	4.08	4.03	7.17	9.25	11.57
		stdev	0.00	1.07	0.96	1.17	6.85	2.91	3.47
Run time (s)			0.0	0.180	0.065	0.072	0.965	233.6	257.9

for all algorithms and all metrics, including run time in seconds.

SAM	**SAMp**	**SNE**	**SNEp**	**SPC**$_0$	**SPC**$_a$	**SPC**$_b$	**SPC**$_c$	**SPC**$_1$	**CPC**
8.20	8.27	7.21	7.30	8.58	8.35	8.35	8.35	8.35	8.35
15.07	15.14	17.16	37.14	27.59	18.12	17.97	15.69	15.97	17.98
11.44	11.45	10.92	23.66	15.19	12.33	12.17	11.68	11.66	12.17
1.97	1.98	1.69	9.59	4.27	2.74	2.67	2.01	1.98	2.67
6.29	6.37	6.54	6.48	6.80	6.67	6.71	6.71	6.71	6.71
15.69	16.34	19.04	45.97	30.60	19.33	19.10	17.26	17.64	19.12
11.86	11.89	11.60	23.31	16.07	12.94	12.78	12.30	12.29	12.78
2.05	2.09	1.85	9.21	4.74	2.91	2.83	2.20	2.19	2.83
24.00	0.00	12.00	2.00	6.00	6.00	4.00	4.00	4.00	4.00
11400.00	1782.00	11400.00	1070.00	110.00	94.00	102.00	202.00	542.00	94.00
153.18	38.98	7173.36	34.28	44.53	39.21	40.40	43.27	43.83	40.39
876.16	64.15	3526.70	60.82	15.64	14.43	15.99	26.05	37.70	15.98
12.00	0.00	6.00	1.00	3.00	3.00	2.00	2.00	2.00	2.00
11288.00	1222.00	11318.00	614.00	65.00	54.00	55.00	120.00	345.00	53.00
115.90	22.96	5597.44	18.98	23.56	20.61	21.26	23.03	23.46	21.26
865.29	43.46	3040.43	37.60	8.80	7.99	8.88	15.19	23.17	8.87
2.99	2.81	2.90	2.83	2.86	2.86	2.84	2.84	2.84	2.84
20.41	25.76	36.33	10.33	3.88	3.67	3.74	4.92	10.99	3.71
3.64	3.19	14.60	3.13	3.18	3.13	3.14	3.18	3.19	3.14
0.82	0.86	4.85	0.60	0.15	0.13	0.15	0.27	0.48	0.15
146.4	101.8	12661	9691	0.62	0.367	0.431	0.521	0.532	10.99

5.4.4 Experimental evaluation fish data set (excerpt 2)

This section gives an in-depth explanation of the results for the second excerpt of the fish data set. In this data set, the fish start moving as a single cluster, after which they split into two separate clusters, and eventually merge into a single cluster again. We investigate the spatial quality and stability in isolation, followed by an exploration of the parameter for SPC. We conclude this section with discussion of the trade-off between spatial quality and stability. Table 5.2 shows the summary statistics over all time steps and for each metric.

Spatial quality In Figure 5.11 we can see the spatial quality measurements for the second excerpt of the fish data set, which are nearly identical to the results for the first excerpt. Even though the movement patterns of the fish are quite different in the two excerpts, this does not seem to affect the overall spatial quality of the resulting orderings. One small but notable difference in the results is that SNEp no longer has a higher KSra than KSdi measurements for the second excerpt.

Stability A similar overview of the stability for this data set can be found in Figure 5.12. Again we see very similar results in comparison to the first excerpt of the fish data set. Differences can be seen for SAM, and the spatial subdivision and clustering techniques. These have both higher absolute values for all stability measures, as well as higher values relative to the other techniques.

Parameter experiment The results of the parameter experiment can be found in Figure 5.13. We choose the intermediate values of the parameter σ to be $a = 0.35$, $b = 0.49$ and $c = 0.65$, and $\sigma = 0.40$ for CPC. The cut-off values are 0.30 and 0.86, meaning all values below and above the respective cut-offs result in the same summaries. While the plotted measurements for all values of σ look very similar to those of the first excerpt, there are some small differences. We choose the intermediate values to be in similar positions in the plot, so that they are nicely spread over all 101 measurements. However, the values are different than for the first excerpt, which indicates that we need different parameter values to achieve similar spatial quality and stability. Overall we see that the maximum values for KSte are also a lot higher for the first excerpt of the fish data set. This is probably due to the fact that in the first excerpt the shoal of fish becomes very rounded, hence forcing the first principal component to change quickly. The difference in intermediate values is therefore not unexpected: even though the stability measurements over the whole second excerpt are very similar to the measurements for the first excerpt, individual instabilities require different parameter settings to result in stable 1D representations.

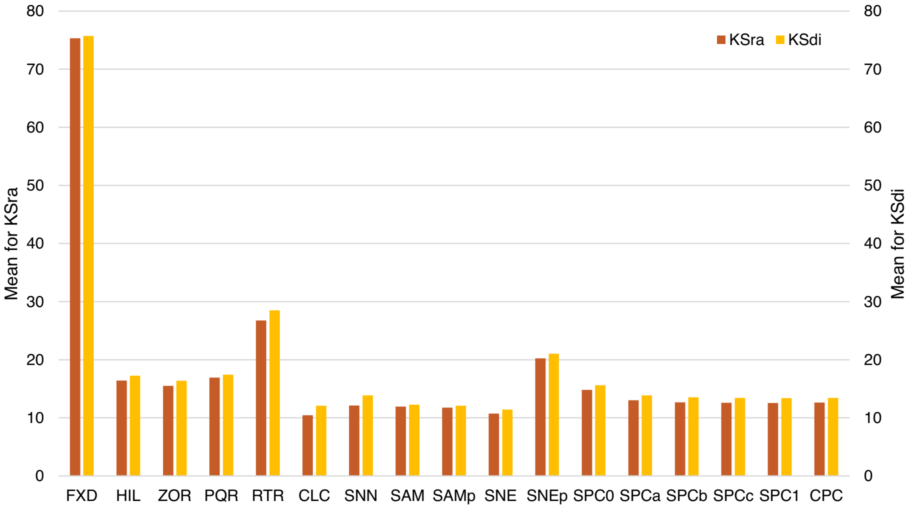


Figure 5.11 Spatial-quality metrics: mean KSra (left) and KSdi (right) for all algorithms over all frames of the second excerpt of the fish data set.

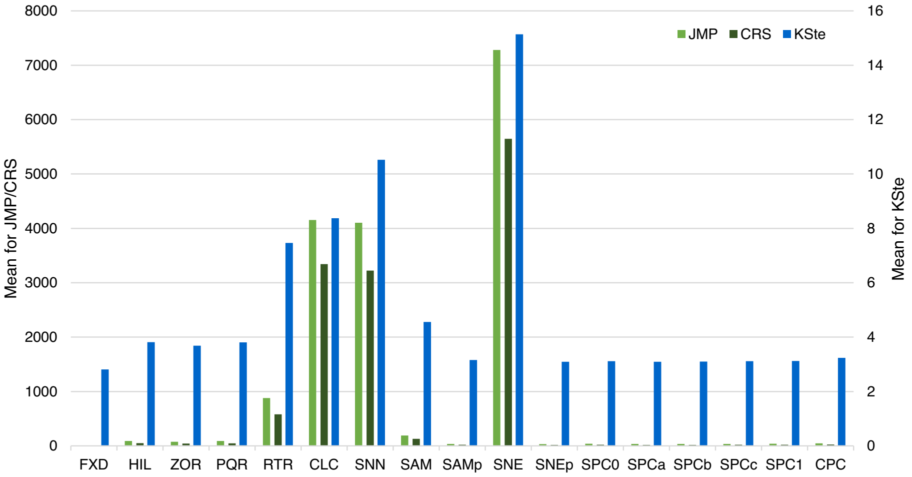


Figure 5.12 Stability metrics: mean JMP, CRS (left axis), and KSte (right axis) for all methods over all frames of the second excerpt of the fish data set.

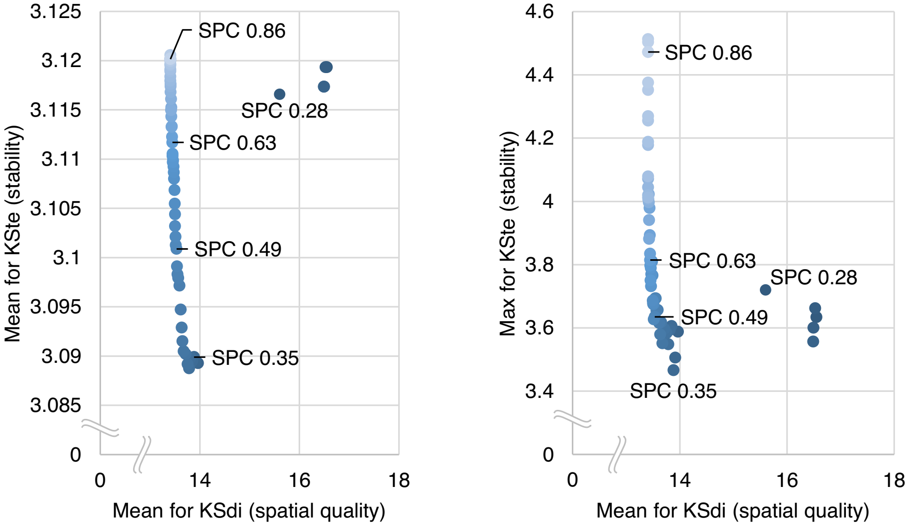


Figure 5.13 A comparison between the mean and mean (left) as well as max and mean (right) for KSte and for KSdi respectively, for uniformly distributed σ of SPC_σ on the second excerpt of the fish data set.

Trade-offs The trade-off between spatial quality and stability is shown in Figures 5.14 and 5.15. As we already saw in the overall stability, SAM performs a lot worse on the second excerpt of the fish data set, compared to the first excerpt. Furthermore, CPC performs a lot worse on maximum stability. This is due to the fact that the fish in the second excerpt split into different clusters and afterwards merge again. The changes in the clustering of CPC will cause big changes in the ordering, leading to instability at the few frames where the clustering changes. Finally, we see that SNE performs better, relative to its performance on the first excerpt, especially for maximum KSte values.

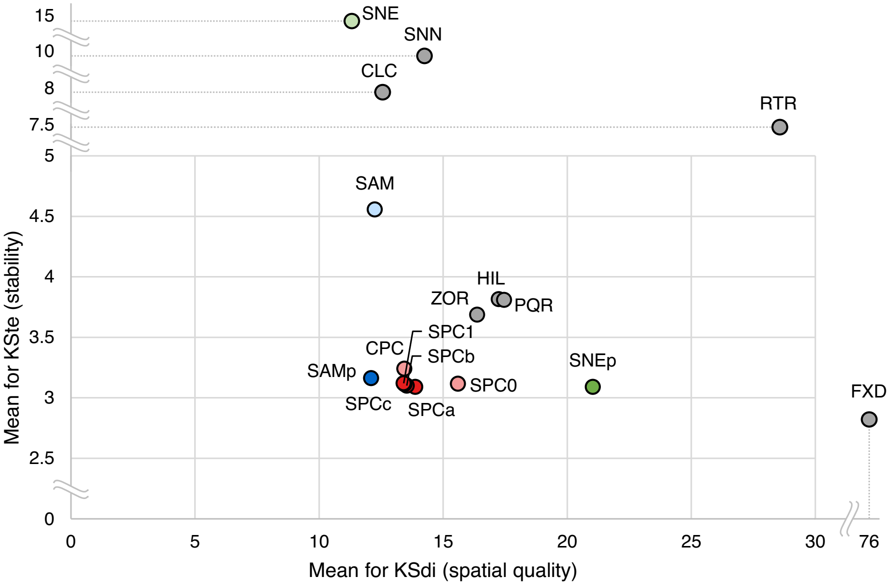


Figure 5.14 A comparison between the mean for KSte and for KSdi for all algorithms on the second excerpt of the fish data set.

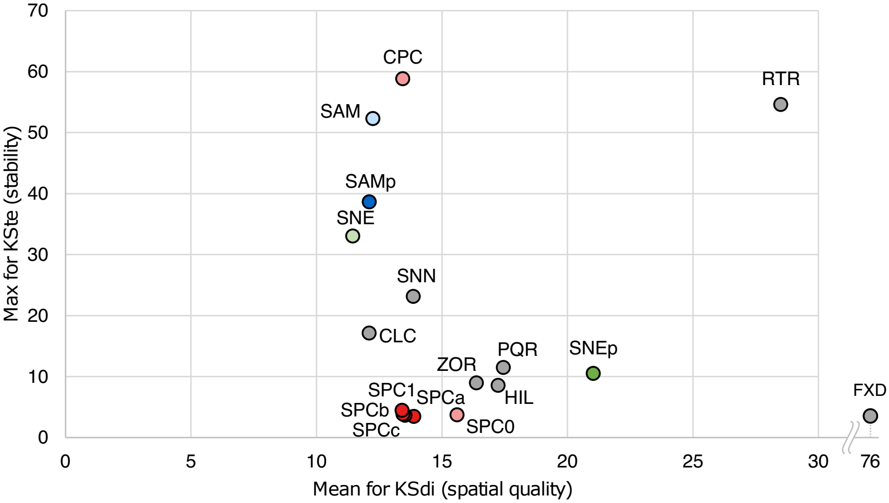


Figure 5.15 A comparison between the max for KSte and the mean for KSdi for all algorithms on the second excerpt of the fish data set.

Table 5.2 Statistics on the second excerpt of fish data set

Fish (excerpt 2)			**FXD**	**HIL**	**ZOR**	**PQR**	**RTR**	**CLC**	**SNN**
Spatial Quality	**KSra** ●	min	68.89	11.18	10.75	12.64	19.41	7.39	8.68
		max	82.70	23.20	22.30	22.94	35.36	15.13	17.68
		mean	75.32	16.43	15.49	16.91	26.77	10.45	12.12
		stdev	2.52	2.22	2.29	1.95	2.74	1.37	1.40
	KSdi ●	min	43.32	8.60	8.56	7.00	12.81	5.49	6.00
		max	91.97	28.83	24.55	24.60	40.92	17.65	21.54
		mean	75.71	17.24	16.37	17.45	28.50	12.10	13.86
		stdev	2.87	2.45	2.40	2.34	3.42	1.77	1.82
Stability	**JMP** ●	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		max	0.00	536.00	442.00	998.00	8466.00	11400.00	11400.00
		mean	0.00	89.84	76.19	88.62	879.72	4154.77	4103.39
		stdev	0.00	80.68	66.93	85.90	1408.15	3733.44	3645.59
	CRS ●	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		max	0.00	296.00	298.00	552.00	6321.00	11324.00	11063.00
		mean	0.00	48.74	40.93	47.48	581.10	3342.71	3223.36
		stdev	0.00	46.03	38.09	47.83	977.84	3278.34	3077.72
	KSte ●	min	2.81	2.81	2.81	2.81	2.81	2.81	2.81
		max	2.81	8.57	8.96	11.47	54.60	17.11	23.14
		mean	2.81	3.82	3.69	3.81	7.46	8.37	10.53
		stdev	0.00	0.90	0.81	0.88	7.31	3.04	3.65
Run time (s)			0.0	0.257	0.160	0.065	0.767	200.8	223.0

for all algorithms and all metrics, including run time in seconds.

	SAM	**SAMp**	**SNE**	**SNEp**	**SPC$_0$**	**SPC$_a$**	**SPC$_b$**	**SPC$_c$**	**SPC$_1$**	**CPC**
	7.98	8.05	7.13	10.00	11.36	8.36	8.36	8.36	8.36	8.36
	16.87	16.02	16.35	32.53	19.66	16.87	15.03	14.99	15.03	16.25
	11.93	11.76	10.75	20.26	14.81	13.03	12.68	12.60	12.57	12.62
	1.88	1.72	1.51	5.38	2.04	1.74	1.58	1.55	1.53	1.63
	6.88	5.79	5.74	10.10	7.45	6.83	6.56	6.35	6.12	6.80
	17.17	15.61	18.11	80.00	21.23	18.23	16.46	16.42	16.14	17.36
	12.25	12.10	11.44	21.03	15.60	13.88	13.53	13.45	13.41	13.44
	1.72	1.65	1.71	6.15	2.11	1.81	1.62	1.60	1.58	1.74
	12.00	0.00	12.00	2.00	10.00	8.00	8.00	8.00	8.00	8.00
	11360.00	2580.00	11400.00	1248.00	96.00	76.00	94.00	106.00	170.00	5078.00
	191.18	34.94	7280.40	31.20	37.28	34.41	35.58	36.77	37.48	45.64
	737.56	78.24	3242.40	71.86	14.15	10.98	12.17	14.83	19.55	193.91
	6.00	0.00	6.00	1.00	5.00	5.00	4.00	4.00	4.00	4.00
	10389.00	1821.00	11317.00	720.00	56.00	42.00	49.00	59.00	97.00	3799.00
	130.86	20.75	5646.96	17.18	19.60	17.96	18.63	19.26	19.74	25.92
	599.65	54.57	2850.57	42.87	7.86	5.99	6.66	8.12	11.07	138.97
	2.90	2.81	2.90	2.83	2.88	2.88	2.87	2.87	2.87	2.87
	52.29	38.64	33.02	10.52	3.72	3.47	3.63	3.81	4.47	58.84
	4.56	3.16	15.14	3.09	3.12	3.09	3.10	3.11	3.12	3.24
	5.15	1.07	4.40	0.65	0.13	0.10	0.11	0.14	0.19	2.54
	189.5	106.9	14342	9504	0.647	0.402	0.460	0.484	0.526	10.41

5.4.5 Experimental evaluation Netlogo data set

We examine the statistics for the Netlogo data set used in our experiments in more detail in this section. Again, we first consider the spatial quality and stability separately, followed by the parameter exploration for SPC. Finally we discuss the trade-off between spatial quality and stability as observed on the Netlogo data set. Table 5.3 gives summary statistics over all time steps and for each metric.

Spatial quality As can be seen in Figure 5.16, while the absolute values for the Netlogo data set are higher than for the fish data set, the relative values are very similar. The fixed order gives very bad results on spatial quality, followed by SNEp and RTR. The spatial subdivision techniques all perform similarly, and are slightly better than the previously mentioned techniques. Of the remaining algorithms, the clustering techniques (CLC and SNN) perform slightly worse than all remaining dimensionality-reduction techniques (SAM, SAMp, SNE, SPC and CPC). For the Netlogo data set we have parameter values $a = 0.40$, $b = 0.59$ and $c = 0.62$ for SPC, and $\sigma = 0.40$ for CPC.

Stability In Figure 5.17 we plot the stability statistics for the Netlogo data set. While the chart looks quite different from the stability chart for the fish data set, this is mostly due to the fact that JMP and CRS count the absolute number of changes in the orders, whereas KSte is normalized. Since the Netlogo data set behaves less stable than the fish data set, all metrics show higher values. However, the Netlogo data set also contains more moving entities, which increases the absolute number of changes even further.

Comparing the statistics of JMP and CRS, we see very similar performances of all algorithms, with SNE being the least stable, while SAMp is the most stable (excluding FXD). On the fish data set SNE was the least stable method overall, and while it still has the highest number of absolute changes on the Netlogo data set, it performs a lot better according to KSte, meaning neighborhoods are perserved relatively well over time. Now we see that RTR performs worst on KSte, followed by the space-filling curves (HIL and ZOR) and the clustering algorithms (CLC and SNN) together with PQR. The SPC variants perform relatively worse than on the fish data set, with parameter values close to but lower than 1 being optimal for stability. Finally, SAM, SAMp, SNEp, and CLC are the most stable according to KSte.

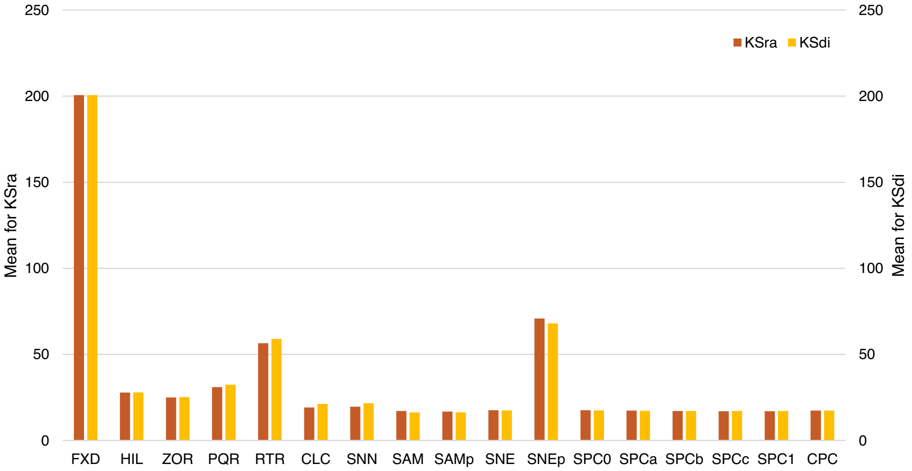


Figure 5.16 Spatial-quality metrics: mean KSra (left) and KSdi (right) for all algorithms over all frames of the Netlogo data set.

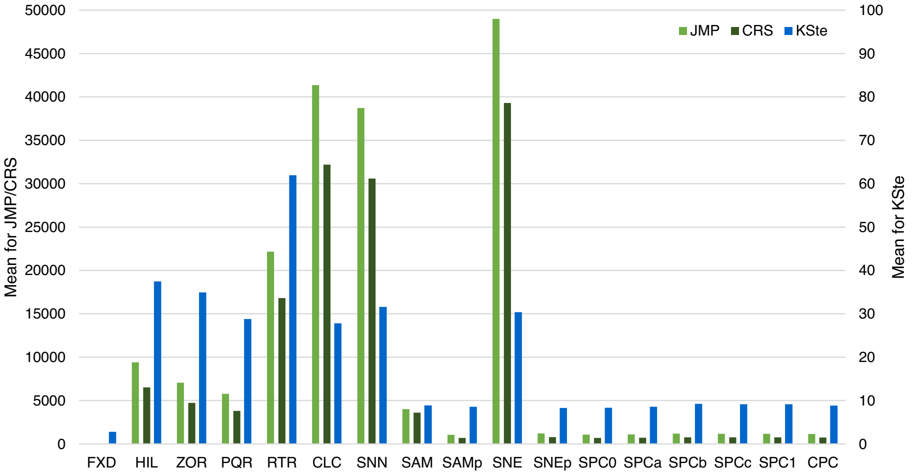


Figure 5.17 Stability metrics: mean JMP, CRS (left axis), and KSte (right axis) for all methods over all frames of the Netlogo data set.

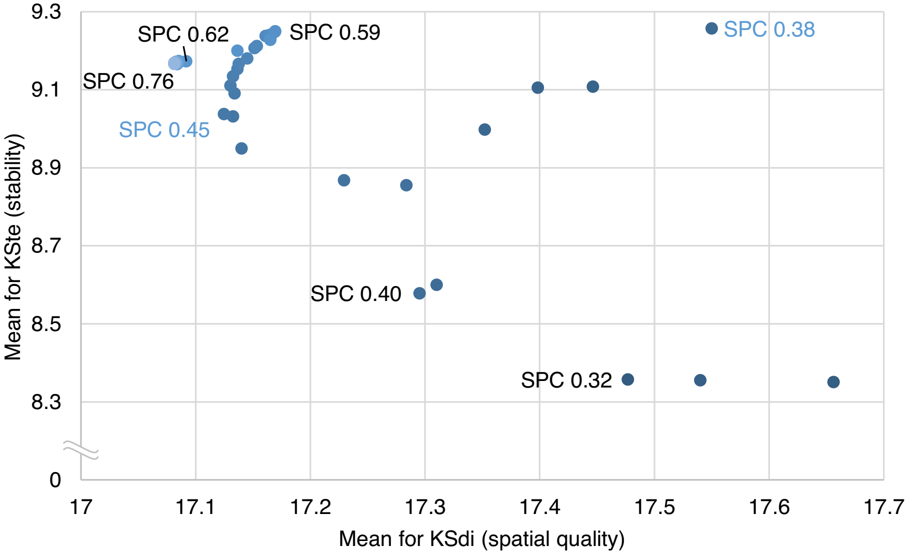


Figure 5.18 A comparison between the mean for KSte and for KSdi, for uniformly distributed σ of SPC_σ on the Netlogo data set.

Parameter experiment The results of the parameter experiment are slightly different for the Netlogo data set. In Figures 5.18 and 5.19 the results are plotted. The cut-off values for σ are 0.76 and 0.32, so everything above 0.76 uses exactly the first principal component per frame, and similarly for values below 0.32 we always interpolate between the first and last frame. The parameter values that are indicated by labels in the figures are the values we used in our other experiments ($a = 0.40$, $b = 0.59$, and $c = 0.62$). Note that there are two blue labels, which represent other values of interest that we will use in the analysis below.

We will consider the results between the values indicated by black labels in the figures. Starting from the lowest parameter value 0.32 we see that increasing σ has chaotic effects on both spatial quality and stability up to 0.40 where this fickle behavior ends. On closer inspection, the values between 0.32 and 0.40 constantly pick up more frames where the entities are stretched enough to use the first principal component. Since the intervals between which interpolation happens, constantly change, the results do not steadily change, but are quite erratic. Parameter value 0.38 shows the worst combination, having both poor spatial quality and stability.

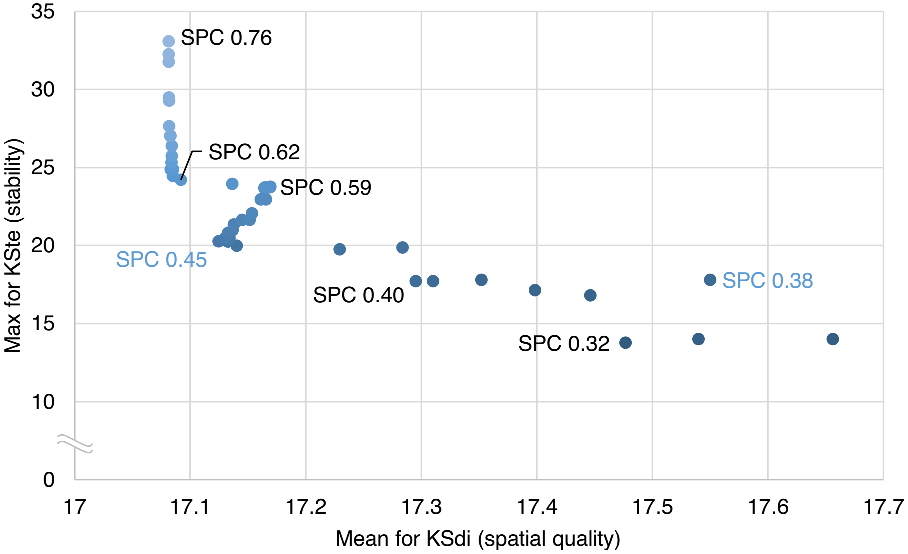


Figure 5.19 A comparison between the max for KSte and the mean for KSdi, for uniformly distributed σ of SPC_σ on the Netlogo data set.

From 0.40 to 0.45 we see a steady decrease in stability and increase in spatial quality, as expected when increasing σ. Further increasing the parameter to 0.59 has negative effects on both the spatial quality and stability. On closer inspection, this increase in spatial quality can be attributed to properties of the Netlogo data set. During instabilities we can observe that at certain frames where SPC projects to an interpolated line, the spatial quality is better than when we project to the first principal component. While we expect projections to the first principal component to have high spatial quality, it is not always the case, as we see here. In the Netlogo data set this occurs when the cluster of points changes direction and shortly does not form a convex shape. It is therefore not unreasonable that interpolating less (and using first principal component more) when increasing σ from 0.46 to 0.56 can negatively effect spatial quality.

At 0.61 SPC splits the interpolation over the two consecutive instabilities that were seen as one big instability. This split improves both spatial quality and stability, up to 0.62 where there are a couple of non-interpolation frames between the instabilities. Increasing the parameter further leads to certain instabilities not being interpolated over any longer, which negatively affects the maximum KSte values.

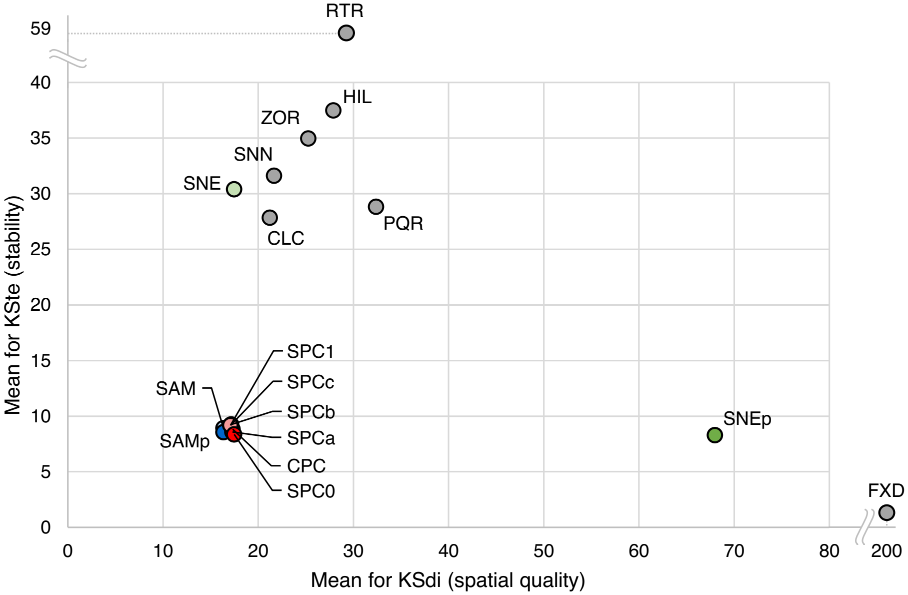


Figure 5.20 A comparison between the mean for KSte and for KSdi for all algorithms on the Netlogo data set.

Trade-offs For the Netlogo data set the trade-offs between spatial quality and stability can be found in Figures 5.20 and 5.21. When considering the mean values for KSte and KSdi, we see a similar spread as before: SAM and SPC variants along with CPC are in the bottom left corner, SNEp and FXD have worse spatial quality but good stability, while the remaining techniques (spatial subdivision, clustering and SNE) have relatively good spatial quality but bad stability. However, SNE is more stable than we have seen for the fish data set, outperforming all spatial subdivision and clustering techniques, except for PQR and CLC. For the maximum values of KSte, we again see CPC perform worse than on the first excerpt of the fish data set. The Netlogo data set mostly consists of a single cluster, but the occasional outlier can trigger the clustering in CPC to find different clusters, resulting spikes of instability.

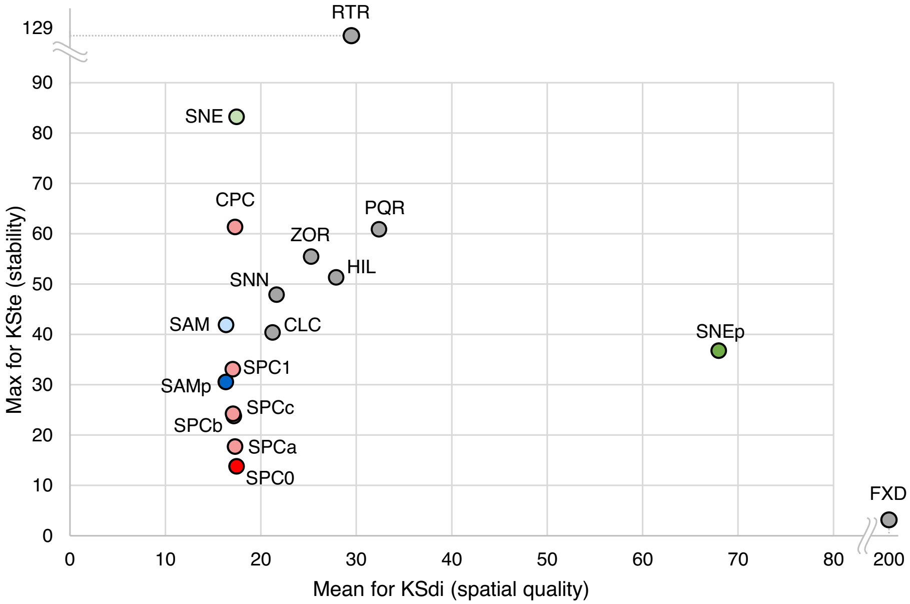


Figure 5.21 A comparison between the max for KSte and the mean for KSdi for all algorithms on the Netlogo data set.

Table 5.3 Statistics on the Netlogo data set

Netlogo data set			FXD	HIL	ZOR	PQR	RTR	CLC	SNN
Spatial Quality	**KSra** ●	min	189.76	19.67	17.13	23.51	38.22	13.43	14.82
		max	208.59	39.67	35.79	44.13	70.81	27.51	28.80
		mean	200.56	27.82	24.98	30.99	56.53	19.18	19.60
		stdev	3.15	3.72	4.14	4.30	5.26	2.63	2.28
	KSdi ●	min	183.56	18.85	17.37	21.39	39.49	13.80	15.33
		max	214.30	41.80	36.31	45.60	75.58	32.98	33.76
		mean	200.59	27.89	25.26	32.38	59.03	21.22	21.66
		stdev	4.12	4.18	4.51	5.18	6.06	3.38	3.01
Stability	**JMP** ●	min	0.00	5528.00	4598.00	1858.00	4970.00	7880.00	6198.00
		max	0.00	13758.00	12238.00	22992.00	46496.00	79982.00	79992.00
		mean	0.00	9414.17	7065.75	5768.82	22164.50	41351.26	38720.13
		stdev	0.00	1586.18	1286.40	2717.96	8014.49	23832.28	25768.68
	CRS ●	min	0.00	3753.00	2633.00	1204.00	3435.00	5493.00	4723.00
		max	0.00	11287.00	8578.00	14010.00	35039.00	73754.00	72110.00
		mean	0.00	6522.55	4737.94	3816.76	16814.07	32187.21	30593.19
		stdev	0.00	1347.81	1245.45	1685.39	6389.95	20475.70	22200.35
	KSte ●	min	2.80	26.32	21.72	12.34	20.62	18.45	22.11
		max	2.80	51.32	55.45	60.91	128.73	40.41	47.91
		mean	2.80	37.48	34.96	28.80	61.95	27.83	31.61
		stdev	0.00	5.39	7.35	8.48	20.52	4.59	4.79
Run time (s)			0.0	0.169	0.112	0.061	0.967	4481	3046

for all algorithms and all metrics, including run time in seconds.

	SAM	**SAMp**	**SNE**	**SNEp**	**SPC$_0$**	**SPC$_a$**	**SPC$_b$**	**SPC$_c$**	**SPC$_1$**	**CPC**
	13.72	13.72	11.92	16.40	13.92	13.81	13.81	13.81	13.81	13.81
	22.66	22.21	26.73	92.00	21.83	21.25	21.82	21.09	21.52	21.53
	17.14	16.75	17.57	70.86	17.53	17.31	17.11	17.04	17.03	17.30
	1.78	1.61	3.03	14.38	1.83	1.85	1.77	1.64	1.63	1.77
	12.57	12.60	12.36	17.69	12.84	12.86	12.86	12.86	12.86	12.85
	22.09	22.04	26.64	106.54	23.21	22.70	22.77	21.94	22.08	22.79
	16.35	16.34	17.47	67.99	17.48	17.30	17.17	17.09	17.08	17.32
	1.64	1.64	3.17	16.37	1.90	1.89	1.83	1.71	1.70	1.85
	230.00	154.00	502.00	184.00	566.00	176.00	176.00	176.00	176.00	176.00
	80000.00	4628.00	80000.00	16564.00	1884.00	2528.00	3406.00	3458.00	4986.00	9282.00
	4020.88	1059.15	48987.86	1223.89	1078.29	1098.25	1187.71	1172.05	1171.03	1142.85
	14884.00	900.08	26536.86	1488.21	333.65	558.37	957.94	1000.90	1053.44	735.88
	126.00	81.00	288.00	99.00	340.00	93.00	93.00	93.00	93.00	93.00
	79316.00	3070.00	79429.00	10479.00	1273.00	1667.00	2308.00	2332.00	3309.00	6218.00
	3612.16	698.99	39300.60	792.62	693.40	707.86	763.85	753.85	752.84	735.16
	14690.86	622.54	23131.64	964.32	232.17	381.28	647.36	677.40	711.94	497.88
	3.61	3.35	4.93	3.45	5.21	3.42	3.42	3.42	3.42	3.42
	41.88	30.52	83.19	36.73	13.76	17.72	23.76	24.21	33.09	61.30
	8.90	8.56	30.39	8.29	8.36	8.58	9.25	9.17	9.17	8.82
	5.51	5.47	12.08	4.62	2.15	3.45	5.94	6.23	6.59	4.58
	294.4	221.6	29348	24414	0.201	0.225	0.282	0.292	0.304	23.05

5.4.6 Experimental evaluation clustered data set

The results of our experimental evaluation for the clustered data set are presented in this section. The structure of this section is similar to the previous two sections, first considering spatial quality and stability in isolation, followed by the parameter experiment. We end the section by considering the trade-off between spatial quality and stability. Table 5.4 provides summary statistics over all time steps.

Spatial quality The chart in Figure 5.22 shows the spatial quality for all techniques. Familiar patterns can be found: FXD, RTR, and SNEp are the worst performers, but are this time joined by PQR as another technique that gives relatively worse spatial quality. The other algorithms all score relatively close on spatial quality; in order of increasing spatial quality (and decreasing measure values), the other spatial subdivision techniques are followed by SPC variants, the clustering techniques and finally SNE, SAM, SAMp and CPC. Since there are multiple cluster in this data, which do not interact with each other, it is not surprising that clustering techniques and CPC perform so well.

Stability The results on the stability of the clustered data set are shown in Figure 5.23. These results are again very similar to the results on the excerpts of the fish data set, with the exception that clustering techniques and SNE perform better in comparison to the other techniques. Since the clusters do not interact and contain the same points in every frame, the clustering techniques also perform very well on stability. Especially when considering the KSte we see the clustering techniques and SNE perform better than on the other data sets. The clustering technique CLC even beats some spatial subdivision techniques (HIL and ZOR) on this measure.

Parameter experiment The parameter experiment also gave some surprising results for the clustered data set, as already explained in the main text. Figure 5.24 shows charts containing the results. The cut-off values are 0.98 and 0.42 for this data set, meaning that every value above 0.98 and below 0.42 uses exactly the projection vectors as the visual summaries using the cut-off values. The parameter values that are indicated by labels in the figures are the values we used in our other experiments. As intermediate values we choose $a = 0.50$, $b = 0.59$, and $c = 0.86$, while for CPC we choose $\sigma = 0.50$.

Starting from 0.86, we see that lowering the parameter value shows an inverse relation between spatial quality and stability: as the parameter value decreases, the

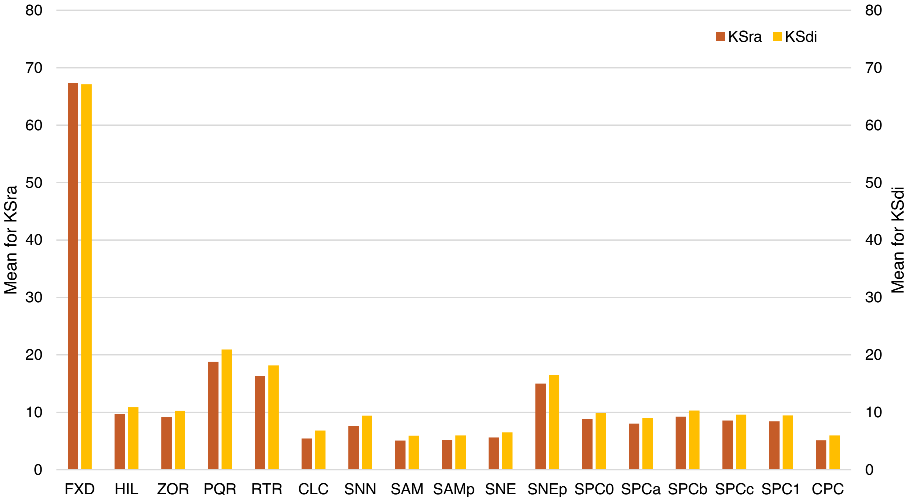


Figure 5.22 Spatial-quality metrics: mean KSra (left) and KSdi (right) for all algorithms over all frames of the clustered data set.

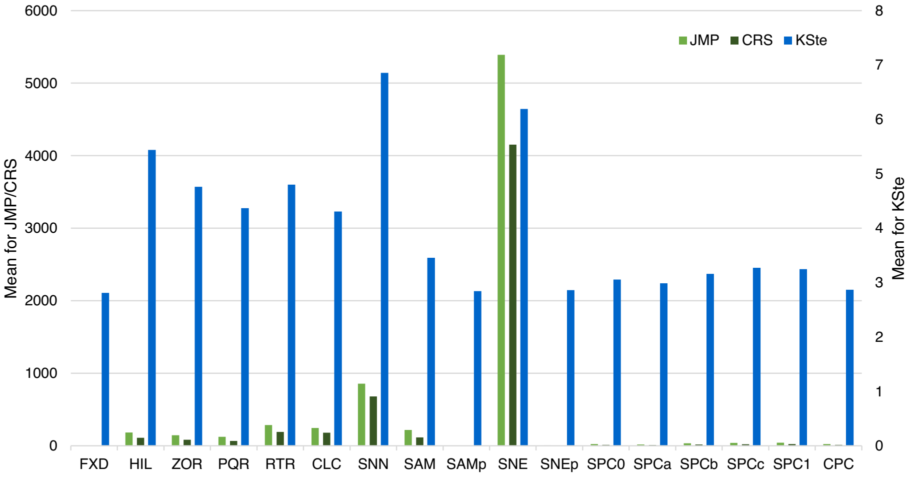


Figure 5.23 Stability metrics: mean JMP, CRS (left axis), and KSte (right axis) for all methods over all frames of the clustered data set.

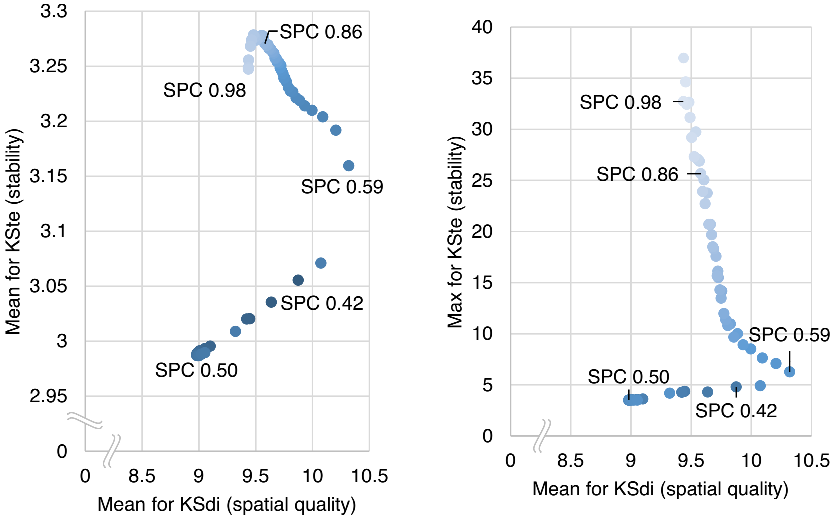


Figure 5.24 A comparison between the mean and mean (left) as well as max and mean (right) for KSte and for KSdi respectively, for uniformly distributed σ of SPC_σ on the clustered data set.

stability increases while the spatial quality deteriorate. This is the expected behavior, which we already saw for the fish data set.

From 0.59 on, we see that lowering σ improves both the spatial quality as well as the stability, just as we saw for some parameters in the Netlogo data set. The first principal component does not seem to be the vector that results in the best spatial quality here, hence interpolating more can give better spatial quality, while improving stability. Lowering σ further after 0.50 results in worse spatial quality and stability, as we saw in all other data sets as well.

Finally, when increasing σ above 0.86 we see the stability improve. While this is counterintuitive in general, it can be explained for this data set. As σ increases, we interpolate less and over configurations where the point set is not stretched. This has a positive effect on the stability in this data set, since it prevents 2 clusters from overlapping a lot: when interpolating, we get a lot of frames where the projected points of two clusters interleave, while the points move in opposite directions. This causes many changes in the neighborhood of all the points in those two clusters. If we interpolate less, this behavior is prominent and contained in a few frames, leading to higher stability according to KSte.

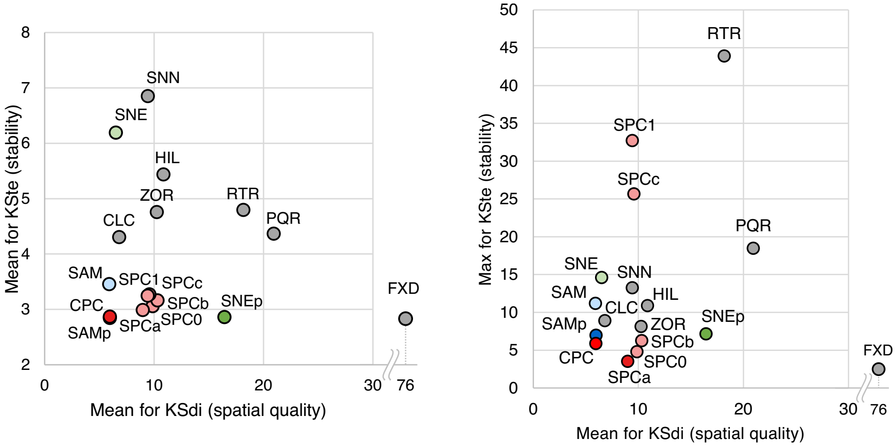


Figure 5.25 A comparison between the mean and mean (left) as well as max and mean (right) for KSte and for KSdi respectively, for for all algorithms on the clustered data set.

Trade-offs The trade-offs between spatial quality and stability for the clustered data set can be observed in Figure 5.25. As we already observed when considering stability in isolation, clustering techniques and SNE perform really well on this data set, especially when considering maximum values for KSte. These techniques end up in the bottom left corner, making them viable techniques for data sets that are clustered. Still, they are still outperformed by SPC variants for low σ values, SAMp, SNEp and CPC, when it comes to stability. For $\sigma = 0.50$ SPC performs particularly well, even better than SAMp and CPC on maximum KSte. However, SAMp and CPC also have very good spatial quality, making them the best techniques for this data.

Table 5.4 Statistics on the clustered data set

Cluster data set			**FXD**	**HIL**	**ZOR**	**PQR**	**RTR**	**CLC**	**SNN**
Spatial Quality	**KSra** ●	min	64.91	6.10	6.13	11.94	9.35	4.45	5.66
		max	70.03	15.99	16.14	27.75	30.41	7.90	9.67
		mean	67.36	9.69	9.15	18.79	16.31	5.45	7.60
		stdev	0.95	1.85	1.63	2.94	3.18	0.53	0.63
	KSdi ●	min	65.11	6.93	7.03	13.86	11.12	5.61	7.06
		max	68.33	17.45	18.22	30.05	33.49	9.67	11.89
		mean	67.09	10.87	10.25	20.94	18.17	6.81	9.43
		stdev	0.72	2.04	1.90	3.17	3.25	0.68	0.81
Stability	**JMP** ●	min	0.00	34.00	28.00	0.00	0.00	0.00	0.00
		max	0.00	574.00	480.00	2096.00	4592.00	8450.00	6832.00
		mean	0.00	184.08	145.20	123.17	284.76	244.52	856.02
		stdev	0.00	72.45	55.66	135.47	780.47	725.05	903.04
	CRS ●	min	0.00	18.00	14.00	0.00	0.00	0.00	0.00
		max	0.00	400.00	281.00	1146.00	3410.00	8374.00	6433.00
		mean	0.00	109.32	81.99	67.59	190.30	180.25	681.28
		stdev	0.00	48.47	35.78	76.38	532.56	666.13	792.64
	KSte ●	min	2.81	3.21	3.09	2.81	2.81	2.81	2.81
		max	2.81	10.90	8.15	18.49	43.91	8.90	13.27
		mean	2.81	5.44	4.76	4.37	4.80	4.31	6.85
		stdev	0.00	1.14	0.84	1.42	5.30	0.86	2.13
Run time (s)			0.0	0.189	0.143	0.055	0.563	75.67	87.29

for all algorithms and all metrics, including run time in seconds.

SAM	**SAMp**	**SNE**	**SNEp**	**SPC$_0$**	**SPC$_a$**	**SPC$_b$**	**SPC$_c$**	**SPC$_1$**	**CPC**
4.40	4.43	4.41	7.70	5.86	5.82	5.60	5.60	5.60	4.39
7.50	7.36	11.78	21.09	16.43	12.18	17.00	25.54	26.39	7.06
5.09	5.14	5.61	15.00	8.84	8.03	9.23	8.55	8.42	5.11
0.56	0.57	1.13	3.97	2.04	1.40	1.96	2.30	2.01	0.59
5.21	5.25	5.21	8.82	6.76	6.68	6.47	6.46	6.46	5.23
8.39	8.27	13.12	22.85	18.00	13.31	18.65	28.18	29.22	7.95
5.92	5.98	6.51	16.44	9.88	8.98	10.32	9.58	9.44	5.96
0.56	0.61	1.25	4.20	2.23	1.47	2.15	2.52	2.20	0.64
18.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00	0.00	0.00
4364.00	276.00	8450.00	546.00	146.00	66.00	232.00	1842.00	4032.00	4294.00
217.57	3.07	5389.14	5.35	24.65	18.78	33.20	40.85	42.16	23.81
772.54	8.97	2212.80	17.60	19.22	11.33	31.75	106.54	173.01	283.74
9.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
2626.00	206.00	8385.00	518.00	85.00	34.00	133.00	1027.00	2416.00	2149.00
116.02	1.81	4151.47	2.95	12.95	9.74	17.55	22.00	23.21	11.97
402.28	6.32	1919.62	15.37	10.80	6.07	17.51	59.84	103.38	142.00
2.97	2.81	2.81	2.81	2.81	2.81	2.83	2.81	2.81	2.81
11.18	6.96	14.60	7.14	4.80	3.51	6.25	25.66	32.72	5.88
3.46	2.84	6.19	2.86	3.06	2.99	3.16	3.27	3.25	2.87
0.84	0.12	1.55	0.14	0.23	0.12	0.41	1.44	1.49	0.19
78.89	51.99	24420	3994	0.383	0.238	0.273	0.311	0.323	6.05

5.5 Conclusion

We propose several stable methods for visual summaries using 1D representations, based on existing dimensionality-reduction techniques. The quantitative analysis indicates that our stable methods perform best, in particular SPC: it performs better in stability, and performs as well as or better than its competitors in terms of spatial quality and computational efficiency. We leverage interpolation in our adaptation of PCA to SPC, allowing explicit parametrization. Furthermore, CPC forms a useful extension to SPC when the input data is heavily clustered, since it allows clusters to be separated in 1D representations. SAM and SNE were modified by changing the initial state of the gradient-descent computation; while this generally improves stability, the effect on spatial quality depends on the sensitivity of the underlying measure to local minima. Below we discuss several avenues for future work, based on the strengths and weaknesses of the available methods.

Movement characteristics In the first excerpt of the fish data set, SPC performs particularly well. This excerpt is of a single, mostly convex cluster of moving entities. Thus, proximity is the primary concern for determining neighborhoods and hence indicates which entities should be close to each other in the 1D representation. By definition, the first principal component captures the most discriminating axis – for our single cluster data this is most indicative of neighborhoods, explaining the performance of our method in terms of spatial quality. Clustering methods suffer in quality (either spatially or temporally) as there are no clear clusters to exploit.

With only a few clusters SPC still performs well, although the method emphasizes the cluster order and the ordering within clusters may use a suboptimal axis, as seen for the second excerpt of the fish data set. The case of many clusters with only a handful of entities we can consider to be effectively the same as a single cluster, as each cluster defines a center and the order within a cluster has little to no influence on the spatial quality.

With multiple, reasonably sized clusters, such as the clustered data set, separating the different clusters in a linear order can be desirable. By their nature, clustering-based methods will perform better in separating these clusters. But our experiments show that such methods will nonetheless struggle to find a good, stable order within the identified clusters.

We thus introduced our hybrid CPC method, combining the capabilities of SPC on a single cluster with CLC to allow better ordering within a cluster. However, at points in time where the cluster composition changes, stability is now harder to achieve.

This contrasts our SPC method which uses both frames before and after the moment of instability to achieve a stable, high-quality result. We leave to future work how such a hybrid method can be turned into a *clairvoyant* algorithm that already aligns the SPC axes of clusters before a change in clusters is actually occurring.

In the case of a complexly shaped cluster, we face yet another issue. Cluster detection might not be adequate to find the necessary structure. Neither does a single, straight projection axis necessarily capture proximity or neighborhood structure well and is hence likely to give unsatisfactory results as well. Perhaps methods from topological persistence can play an important role in identifying the structures of these clusters. We leave the development and evaluation of algorithms for more complex data as future work. Our results show the potential here, for adapting existing methods to explicitly consider stability.

Beyond spatial data Our stable methods can be used in any situation with time-varying data in at least two (numeric) dimensions, to determine the ordering. In a MotionRug, another dimension is then used to color the elements in each order. Such an approach may thus be useful for providing an overview also for abstract data. However, we expect it to be primarily useful when proximity (or more generally, neighborhoods) of items are meaningful in the dimensions used to derive principal components. Investigating precise conditions under which this approach is effective is left to future work.

Overview-first Visual summaries are primarily an overview-first tool. They are intended to give an analyst a rough idea of what happens during the motion of the entities, as a first entry point to find time spans or sets of entities to further investigate. It is therefore important to understand how movement patterns relate to patterns visible in the summary and vice versa.

To ensure that *collective* movement of subgroups leads to observable patterns in a visual summary, we need the attribute used for coloring to be similar for spatially close entities. The spatial coloring we use in this chapter inherently has this property, but also double encodes space, as the 1D representation is also reflecting the spatial dimensions. In our data sets this is the case for speed and inherent in other properties derived from the spatial arrangement, such as distance to centroid; see [23] for MotionRugs colored by for example speed. Without a relation between spatial proximity and attribute value, the colors in the MotionRug may jump and it becomes difficult to follow entities or subgroups. One notable exception may be to simply assign a distinct fixed color to each element. Though it is impossible to find sufficiently many visually distinct colors when the number of entities is large, it would eliminate ambiguity by entities changing their speed, or any other time-varying property used for coloring.

Furthermore, we may want to augment a visual summary with information about its spatial and temporal quality. To an analyst, this may also convey useful information. Beyond communicating a certain level of reliability for observed patterns in the overview, areas with large spatial distortion or high instability may indicate times where unusual behavior is occurring, and may thus warrant further investigation. We have augmented our summaries in Figures 5.2 and 5.6, using a simple bar chart to show spatial quality and stability per time step. Figure 5.8 also shows a fine-grained representation of spatial and temporal quality, simply using this as an attribute to color the visual summary. Various other encodings could be considered, e.g. reducing the saturation of the colors or underlining the summary with two lines where the pixel colors indicate the spatial and temporal quality. How to best visually convey the spatial and temporal quality, and how this affects user understanding are left to future work.

Finally, MotionLines can also be seen as an augmentation of very compressed visual summaries, such as MotionRugs. By using additional space, we can encode more spatial properties into the summary: As can be seen in Figure 5.26, a MotionLines visual summary not only communicates proximity via the ordering, but also shows relative distances. The 1D representations produced by the dimensionality-reduction techniques all provide this information, and can hence be used to create more expressive visual summaries such as MotionLines. This hints at a possible trade-off between the compactness and expressiveness of visual summaries, which can be further investigated in future work.

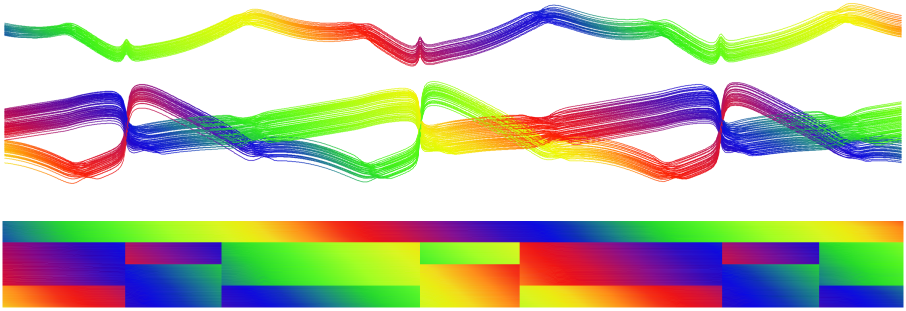
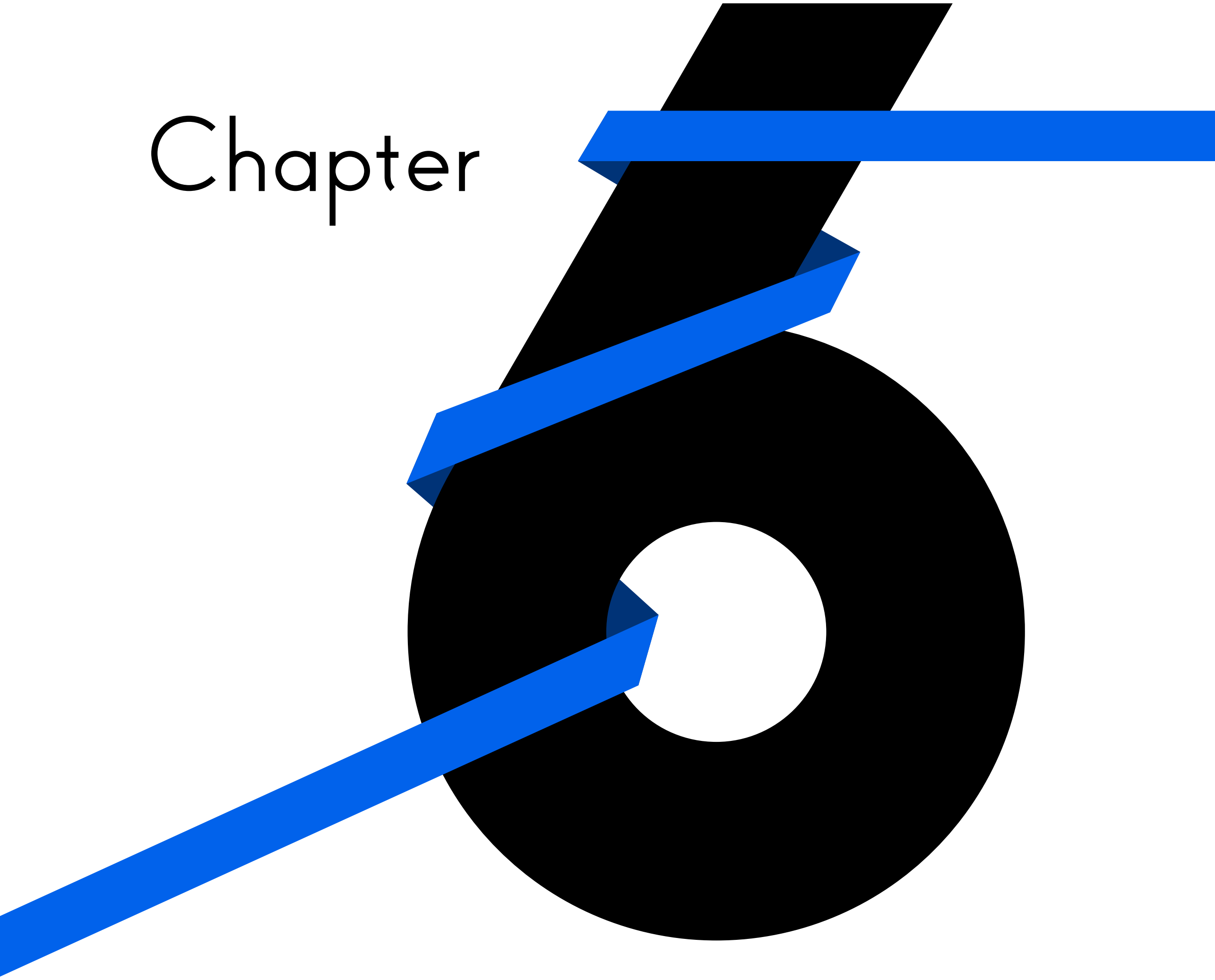


Figure 5.26 A MotionLines and a MotionRug of the clustered data set, using 1D representations produced by our CPC algorithm ($\sigma = 0.5$).

Chapter

6



Conclusion and Future Work

In this thesis, we studied the stability of geometric algorithms. Intuitively, stability means that small changes in the input of an algorithm should lead to small changes in the output. We formalize this intuition by developing a framework for algorithm stability. In the framework, we propose various ways of analyzing the trade-off between the quality and stability of algorithms for time-varying data. We then applied this framework to three problems in computational geometry: the kinetic Euclidean minimum spanning tree problem, the kinetic k-centers problem and kinetic orientation-based shape descriptors. Furthermore, we proposed stable dimensionality-reduction methods for visual summaries to compute meaningful overviews.

We give a more detailed summary of these results in Section 6.1 and discuss open problems and perspectives for future work in Section 6.2.

▸ 6.1 Main results

We proposed a framework for algorithm stability, to study the trade-off between the quality and stability of solutions produces by an algorithm. The framework introduces three types of stability analysis for algorithms on time-varying data: event stability, topological stability and Lipschitz stability. The different types of stability gradually impose stronger stability requirements on the solutions of an algorithm. Event stability measures only the number of discrete changes in the solution of an algorithm; topological and Lipschitz stability measure how close to optimal a solution is, while requiring that the solution changes continuously. Additionally, Lipschitz stability enforces a bound on the speed at which the solution can change with respect to the changes in the input. Besides these types of analysis, we also distinguish between different models for algorithms on time-varying data: stateless, state-aware and clairvoyant algorithms. These models depend on the availability of the data and influence the stability that can be achieved. Stateless algorithms are simply functions from input to output, while clairvoyant algorithms have access to the full time-varying data. State-aware algorithm form a middle ground: they have knowledge of the data only up to a certain point in time.

We first applied this framework to state-aware algorithms for the kinetic Euclidean minimum spanning tree (EMST) problem: we obtained approximations to optimal EMSTs that improve the event stability with respect to existing work, and approximations that allow topologically stable solutions. We additionally showed that k-Lipschitz stable spanning trees cannot approximate optimal EMSTs, for low values of K. We then proved bounds on the topological stability ratio, a ratio on the quality between stable and optimal solutions, for the kinetic k-center problem, and developed clairvoyant algorithms to compute this ratio. Furthermore, we analyzed the stability of three orientation-based shape descriptors: the first principal component, the minimal area oriented bounding box, and the thinnest strip. We showed that no topologically stable stateless algorithm can exist for these shape descriptors, and proved tight bounds on the topological stability ratio for state-aware algorithms. Finally, we showed that chasing algorithms, algorithms which move the current solution towards an optimal solution, with sufficient speed can produce Lipschitz stable approximations to optimal oriented bounding boxes and thinnest strips.

Visual summaries give an overview of time-varying data, typically by computing a 1D representation of the data at each point in time, and placing these representations along a time line. Such summaries have high quality if they are stable and represent the spatial structure of the data well. We proposed to compute 1D representations for visual summaries using stable dimensionality-reduction techniques. Specifically, we developed the Stable Principal Component algorithm, which allows a user configurable trade-off between spatial quality and stability. Via quantitative analysis, we showed that the dimensionality-reduction methods produce 1D representations that not only preserve spatial structure in the data, but are also coherent over time.

6.2 Future work

While our framework for algorithm stability allowed us to gain many insights into the trade-offs between solution quality and stability of algorithms for time-varying data, it opened even more opportunities for future work. In the remainder of this section, we outline a few promising directions to explore further.

Applying and extending the framework In this thesis we apply our framework to three theoretical problems in computational geometry, but there are many other problems that can be considered for stability analysis. In graph drawing and geographic information systems, algorithms are widely used to solve problems on time-varying data, including network analysis, map labeling, trajectory analysis, and so on. In many cases, optimal solutions and approximations for static data can already be computed by algorithms. Hence it should be possible to formulate chasing algorithms for time-varying variants of those problems, and analyze such chasing algorithms using our framework.

Although our framework currently considers only the trade-off between solution quality and stability, it could potentially also incorporate the algorithmic aspect of stability. This would allow us to get insights into the trade-offs with the running time of algorithms for time-varying data. For example, we would like to know how efficiently we can compute stable solutions. An example can be found in Chapter 3, where we give a clairvoyant algorithm that produces topologically stable solutions for the k-centers problem and analyze its running time. However, our framework currently does not offer any tools to specifically find stable solutions that can be computed efficiently. These solutions would be very valuable in practice: if we can develop a state-aware algorithm that efficiently computes a stable solution, then this allows us to find stable solutions on the fly. Consider a situation where GPS data is being processed in an online fashion. Such an algorithm would produce stable solutions, by processing the GPS data as it is collected. However, it is important that solutions are computed efficiently, otherwise a state-aware algorithm has no benefits over a clairvoyant one.

The visual summaries studied in Chapter 5 are only one of the many visualization techniques that benefit from stable algorithms. In the introduction we already considered a time-varying set visualization, where the elements of the set where moving planes. While visualization techniques such as KelpFusion [77] are already able to produce high quality set visualizations on static input data, no stable algorithms currently exist. Sets are often visualized using spanning trees, and hence the topo-

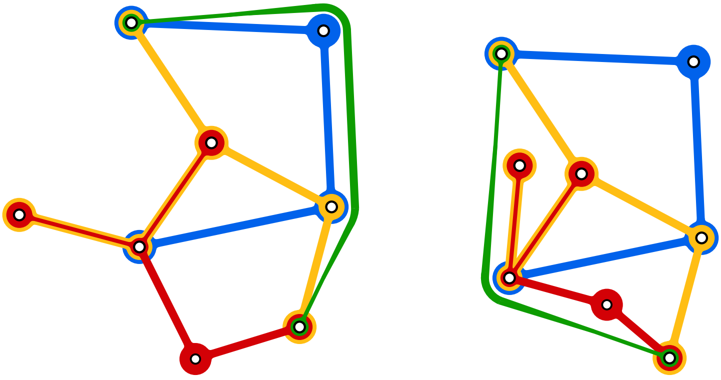


Figure 6.1 A set visualization based on spanning trees: The green set either undergoes a discrete change or overlaps other sets between **(a)** and **(b)**.

logically stable spanning trees in Chapter 2 would form the perfect starting point for a stable visualization technique. However, we are faced with many challenges when moving from static to time-varying data. Static set visualizations often optimize for minimal ink usage and few overlapping sets, and this can lead to instabilities or visualizations of low quality in the time-varying setting. For example, to prevent overlapping sets, one set can route around other sets as if they are obstacles. As the points move, this detour can become needlessly long, requiring a different routing to minimize ink usage. Such an update results in either a discrete change, or overlapping sets during a continuous transformation (See Figure 6.1).

Human-in-the-loop algorithms also benefit from being stable. Such algorithms present the user with a solution, which they can then adapt to their liking, before returning the solution back to the algorithm. Ideally the algorithm takes the newly made changes into account and produces a new solution that still reflects the changes made by the user intermittently. However, if the algorithm simply looks for an optimal solution, then it will always return to the initial solution that was presented to the user. It is therefore important that such an algorithm is stable: the algorithm may only make small changes to the solution, in the places where the user formulated additional constraints.

Bibliography

[1] Pankaj Agarwal, Jie Gao, Leonidas Guibas, Haim Kaplan, Natan Rubin, and Micha Sharir. Stable Delaunay graphs. *Discrete & Computational Geometry*, 54(4):905–929, 2015.

[2] Pankaj Agarwal, Sariel Har-Peled, and Kasturi Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[3] Oswin Aichholzer, Franz Aurenhammer, and Ferran Hurtado. Sequences of spanning trees and a fixed tree theorem. *Computational Geometry: Theory and Applications*, 21(1-2):3–20, 2002.

[4] Sunil Arya and David Mount. A fast and simple algorithm for computing approximate Euclidean minimum spanning trees. In *Proc. 27th Symposium on Discrete Algorithms (SODA)*, pages 1220–1233, 2016.

[5] Andrea Bacciotti and Lionel Rosier. *Liapunov Functions and Stability in Control Theory*. Springer, 2nd edition, 2006.

[6] Ziv Bar-Joseph, Erik Demaine, David Gifford, Angele Hamel, Tommi Jaakkola, and Nathan Srebro. K-ary Clustering with Optimal Leaf Ordering for Gene Expression Data. *Bioinformatics*, 19(9):1070–8, 2003.

[7] Gill Barequet, Bernard Chazelle, Leonidas Guibas, Joseph Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3d. *Computer Graphics Forum*, 15(3):387–396, 1996.

[8] Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91–109, 2001.

[9] Julien Basch, Leonidas Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[10] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. 1990 Special Interest Group on Management of Data Conference (ACM SIGMOD)*, pages 322–331, 1990.

[11] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry: Theory and Applications*, 43(3):312–328, 2010.

[12] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24(4):509–522, 2002.

[13] Sergey Bereg, Binay Bhattacharya, David Kirkpatrick, and Michael Segal. Competitive algorithms for maintaining a mobile center. *Mobile Networks and Applications (MONET)*, 11(2):177–186, 2006.

[14] Mark de Berg, Marcel Roeloffzen, and Bettina Speckmann. Kinetic 2-centers in the black-box model. In *Proc. 29th Symposium on Computational Geometry (SoCG)*, pages 145–154, 2013.

[15] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[16] Sergei Bespamyatnikh, Binay Bhattacharya, David Kirkpatrick, and Michael Segal. Lower and upper bounds for tracking mobile users. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (IFIP TCS)*, volume 223, pages 47–58.

[17] Sergei Bespamyatnikh, Binay Bhattacharya, David Kirkpatrick, and Michael Segal. Mobile facility location. In *Proc. 4th Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, pages 46–53, 2000.

[18] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[19] Peter Brass, Eowyn Cenek, Cristian Duncan, Alon Efrat, Cesim Erten, Dan Ismailescu, Stephen Kobourov, Anna Lubiw, and Joseph Mitchell. On simultaneous planar graph embeddings. *Computational Geometry: Theory and Applications*, 36(2):117–130, 2007.

[20] Michael Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proc. 23rd Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1704–1711, 2010.

[21] Luitzen Brouwer. Über Abbildung von Mannigfaltigkeiten. *Mathematische Annalen*, 71(1):97–115, 1911.

[22] Dragana Brzakovic, Xiao Mei Luo, and P. Brzakovic. An approach to automated detection of tumors in mammograms. *IEEE Transactions on Medical Imaging*, 9(3):233–241, 1990.

[23] Juri Buchmüller, Dominik Jäckle, Eren Cakmak, Ulrik Brandes, and Daniel Keim. MotionRugs: visualizing collective trends in space and time. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):76–86, 2019.

[24] Juri Buchmüller, Udo Schlegel, Eren Cakmak, Daniel Keim, and Evanthia Dimara. SpatialRugs: enhancing spatial awareness of movement in dense pixel visualizations. In *Proc. 11th EuroVis Workshop on Visual Analytics (EuroVA)*, pages 1–5, 2020.

[25] Michael Burch, Corinna Vehlow, Stephan Diehl, and Daniel Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011.

[26] Timothy Chan. More planar two-center algorithms. *Computational Geometry*, 13(3):189–198, 1999.

[27] Francis Chin, Cao An Wang, and Fu Lee Wang. Maximum stabbing line in 2d plane. In *Proc. 5th Computing and Combinatorics Conference (COCOON)*, pages 379–388, 1999.

[28] Fan Chung and Ronald Graham. A new bound for Euclidean Steiner minimal trees. *Annals of the New York Academy of Sciences*, 440(1):328–346, 1985.

[29] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. In *Proc. 21st Symposium on Computational Geometry (SoCG)*, pages 263–271, 2005.

[30] Weiwei Cui, Xiting Wang, Shixia Liu, Nathalie Riche, Tara Madhyastha, Kwan-Liu Ma, and Baining Guo. Let It Flow: a static method for exploring dynamic graphs. In *Proc. 7th Pacific Visualization Symposium (PacificVis)*, pages 121–128, 2014.

[31] Bastian Degener, Joachim Gehweiler, and Christiane Lammersen. Kinetic facility location. *Algorithmica*, 57(3):562–584, 2010.

[32] Darko Dimitrov, Christian Knauer, Klaus Kriegel, and Günter Rote. Bounds on the quality of the PCA bounding boxes. *Computational Geometry*, 42(8):772–789, 2009.

[33] Zvi Drezner. On a modified 1-center problem. *Management Science*, 27:838–851, 1981.

[34] Zvi Drezner. On the rectangular p-center problem. *Naval Research Logistics*, 34(2):229–234, 1987.

[35] Stephane Durocher. *Geometric Facility Location under Continuous Motion.* PhD thesis, University of British Columbia, 2006.

[36] Stephane Durocher and David Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *International Journal of Computational Geometry & Applications*, 18(03):161–183, 2008.

[37] Stephane Durocher and David Kirkpatrick. The Steiner centre of a set of points: stability, eccentricity, and applications to mobile facility location. *International Journal of Computational Geometry & Applications*, 16(04):345–371, 2006.

[38] Jeff Erickson. Dense point sets have sparse Delaunay triangulations or "... but not too nasty". *Discrete & Computational Geometry*, 33(1):83–115, 2005.

[39] Shimon Even and Guy Even. *Graph Algorithms, Second Edition.* Cambridge University Press, 2012.

[40] Raphael Finkel and Jon Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4:1–9, 1974.

[41] Fabrizio Frati, Michael Kaufmann, and Stephen G. Kobourov. Constrained simultaneous and near-simultaneous embeddings. *Journal of Graph Algorithms and Applications*, 13(3):447–465, 2009.

[42] Herbert Freeman and Ruth Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413, 1975.

[43] Sorelle Friedler and David Mount. Approximation algorithm for the kinetic robust k-center problem. *Computational Geometry*, 43(6-7):572–586, 2010.

[44] Jyh-Jong Fu and Richard C.T. Lee. Voronoi diagrams of moving points in the plane. *International Journal of Computational Geometry & Applications*, 1(01):23–32, 1991.

[45] Jie Gao, Leonidas Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. *Discrete & Computational Geometry*, 30(1):45–63, 2003.

[46] Jie Gao, Leonidas Guibas, and An Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35(1-2):2–19, 2006.

[47] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. *Journal of Computational Geometry*, 7(1):308–331, 2016.

[48] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Sliding labels for dynamic point labeling. In *Proc. 23rd Canadian Conference on Computational Geometry (CCCG)*, pages 205–210, 2011.

[49] Emilio Di Giacomo, Walter Didimo, Marc van Kreveld, Giuseppe Liotta, and Bettina Speckmann. Matched drawings of planar graphs. *Journal of Graph Algorithms and Applications*, 13(3):423–445, 2009.

[50] Wayne Goddard and Henda Swart. Distances between graphs under edge operations. *Discrete Mathematics*, 161(1-3):121–132, 1996.

[51] Arthur van Goethem, Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans, Max Sondag, and Jules Wulms. The painter's problem: covering a grid with colored connected polygons. In *Proc. 26th Graph Drawing and Network Visualization (GD)*, pages 492–505, 2017.

[52] Arthur van Goethem, Irina Kostitsyna, Kevin Verbeek, and Jules Wulms. Repulsion region in a simple polygon. In *Abstr. 36th European Workshop on Computational Geometry (EuroCG)*, 73:1–73:7, 2018.

[53] Allan Gordon. A review of hierarchical classification. *Journal of the Royal Statistical Society: Series A (General)*, 150(2):119–137, 1987.

[54] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. 23rd Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH)*, pages 171–180, 1996.

[55] Xianfeng Gu, Yalin Wang, Tony Chan, Paul Thompson, and Shing-Tung Yau. Genus zero surface conformal mapping and its application to brain surface mapping. *IEEE Transactions on Medical Imaging*, 23(8):949–958, 2004.

[56] Leonidas Guibas, Joseph Mitchell, and Thomas Roos. Voronoi diagrams of moving points in the plane. In *Proc. 17th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 113–125, 1992.

[57] Diansheng Guo and Mark Gahegan. Spatial ordering and encoding for geographic data mining and visualization. *Journal of Intelligent Information Systems*, 27(3):243–266, 2006.

[58] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. 1984 Special Interest Group on Management of Data Conference (ACM SIGMOD)*, pages 47–57, 1984.

[59] John Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, 1989.

[60] Nicholas Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 2002.

[61] David Hilbert. Über die Stetige Abbildung Einer Line auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[62] Michael Hoffmann. A simple linear algorithm for computing rectangular 3-centers. In *Proc. 11th Canadian Conference on Computational Geometry (CCCG)*, pages 72–75, 1999.

[63] Ivor van der Hoog, Marc van Kreveld, Wouter Meulemans, Kevin Verbeek, and Jules Wulms. Topological stability of kinetic k-centers. In *Proc. 13th Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 43–55, 2019.

[64] R.Z. Hwang, Richard C.T. Lee, and R.C. Chang. The slab dividing approach to solve the Euclidean p-center problem. *Algorithmica*, 9:1–22, 1993.

[65] Ray Jarvis and Edward Patrick. Clustering using a similarity measure based on shared near neighbours. *IEEE Transactions on Computers*, 22(11):1025–1034, 1973.

[66] Naoki Katoh, Takeshi Tokuyama, and Kazuo Iwano. On minimum and maximum spanning trees of linearly moving points. *Discrete & Computational Geometry*, 13(2):161–176, 1995.

[67] András Kelemen, Gábor Székely, and Guido Gerig. Elastic model-based segmentation of 3-D neuroradiological data sets. *IEEE Transactions on Medical Imaging*, 18(10):828–839, 1999.

[68] James Klosowski, Martin Held, Joseph Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[69] Wiebke Köpp and Tino Weinkauf. Temporal Treemaps: Static Visualization of Evolving Trees. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):534–543, 2019.

[70] Irina Kostitsyna, Maarten Löffler, Max Sondag, Willem Sonke, and Jules Wulms. The hardness of Witness puzzles. In *Abstr. 34th European Workshop on Computational Geometry (EuroCG)*, 67:1–67:6, 2018.

[71] Joseph Kruskal. Multidimensional Scaling by Optimizing Goodness of fit to a Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[72] David Letscher and Kyle Sykes. On the stability of medial axis of a union of disks in the plane. In *Proc. 28th Canadian Conference on Computational Geometry (CCCG)*, pages 29–33, 2016.

[73] Shixia Liu, Yingcai Wu, Enxun Wei, Mengchen Liu, and Wang Liu. StoryFlow: Tracking the Evolution of Stories. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2436–2445, 2013.

[74] Hongjun Lu and Beng Chin Ooi. Spatial Indexing: Past and Future. *IEEE Data Engineering Bulletin*, 16(3):16–21, 1993.

[75] Wladimir Markoff. Über Polynome, die in einem gegebenen Intervalle möglichst wenig von Null abweichen. *Mathematische Annalen*, 77(2):213–258, 1916.

[76] Nimrod Megiddo and Kenneth Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[77] Wouter Meulemans, Nathalie Henry Riche, Bettina Speckmann, Basak Alper, and Tim Dwyer. Kelpfusion: a hybrid set visualization technique. *IEEE transactions on visualization and computer graphics*, 19(11):1846–1858, 2013.

[78] Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms. A framework for algorithm stability. In *Proc. 13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 805–819, 2018.

[79] Wouter Meulemans, Kevin Verbeek, and Jules Wulms. Stability analysis of kinetic orientation-based shape descriptors. *CoRR*, abs/1903.11445, 2019.

[80] John Milnor and David Weaver. *Topology from the differentiable viewpoint*. Princeton U. Press, 1997.

[81] Clyde Monma and Subhash Suri. Transitions in geometric minimum spanning trees. *Discrete & Computational Geometry*, 8(3):265–293, 1992.

[82] Torrie Nichols, Alexander Pilz, Csaba Tóth, and Ahad Zehmakan. Transition operations over plane trees. In *Proc. 13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 835–848, 2018.

[83] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 310–319, 2010.

[84] Doron Nussbaum. Rectilinear p-piercing problems. In *Proc. 1997 Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 316–323, 1997.

[85] Joseph O'Rourke. Finding minimal enclosing boxes. *International Journal of Parallel Programming*, 14(3):183–199, 1985.

[86] Julian Orford. Implementation of criteria for partitioning a dendrogram. *Journal of the International Association for Mathematical Geology*, 8(1):75–84, 1976.

[87] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[88] Isabel Piljek. VisSwarmR: Visual Analytics tool for the generation of Collective Movement Datasets, version 1.0, 2020. `https://github.com/piljek/VisSwarmR`.

[89] Zahed Rahmati, Mohammad Ali Abam, Valerie King, Sue Whitesides, and Alireza Zarei. A simple, faster method for kinetic proximity problems. *Computational Geometry: Theory and Applications*, 48(4):342–359, 2015.

[90] Zahed Rahmati and Alireza Zarei. Kinetic Euclidean minimum spanning tree in the plane. *Journal of Discrete Algorithms*, 16:2–11, 2012.

[91] Paulo Rauber, Alexandre Falcão, and Alexandru Telea. Visualizing Time-Dependent Data Using Dynamic t-SNE. In *Proc. 18th Eurographics Conference on Visualization (EuroVis)*, pages 73–77, 2016.

[92] Craig Reynolds. Steering Behaviors For Autonomous Characters. In *Proc. 1999 Game Developers Conference (GDC)*, pages 763–782, 1999.

[93] Bastian Rieck and Heike Leitte. Persistent homology for the evaluation of dimensionality reduction schemes. *Computer Graphics Forum*, 34(3):431–440, 2015.

[94] Theodore Rivlin. *The Chebyshev Polynomials*, volume 40 of *Pure and Applied Mathematics*. John Wiley & Sons, 1974.

[95] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed R-trees. In *Proc. 1985 Special Interest Group on Management of Data Conference (ACM SIGMOD)*, pages 17–31, 1985.

[96] Natan Rubin. On kinetic Delaunay triangulations: a near-quadratic bound for unit speed motions. *Journal of the ACM*, 62(3):25, 2015.

[97] Natan Rubin. On topological changes in the Delaunay triangulation of moving points. *Discrete & Computational Geometry*, 49(4):710–746, 2013.

[98] John Sammon. A Non-linear Mapping for Data Structure Analysis. *IEEE Transactions on Computers*, C-18(5):401–409, 1969.

[99] Michael Segal. On piercing sets of axis-parallel rectangles and rings. In *Proc. 5th European Symposium on Algorithms (ESA)*, pages 430–442, 1997.

[100] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-tree: a dynamic index for multi-dimensional objects. In *Proc. 13th Very Large Data Bases Conference (VLDB)*, pages 507–518, 1987.

[101] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

[102] Micha Sharir and Emo Welzl. Rectilinear and polygonal p-piercing and p-center problems. In *Proc. 12th Symposium on Computational Geometry (SoCG)*, pages 122–132, 1996.

[103] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[104] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. 15th International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[105] James Sylvester. A question in the geometry of situation. *Quarterly Journal of Mathematics*, 1:79, 1857.

[106] Alexandru Telea. Personal communication, March 2019, March 21, 2019.

[107] Joshua Tenenbaum, Vin De Silva, and John Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[108] Godfried Toussaint. Solving geometric problems with the rotating calipers. In *Proc. 1983 Mediterranean Electrotechnical Conference (MELECON)*, volume 83, A10, 1983.

[109] Eugene Tyrtyshnikov. *A brief introduction to numerical analysis*. Springer, 2012.

[110] Stef van den Elzen, Danny Holten, Jorik Blaas, and Jarke van Wijk. Dynamic Network Visualization With Extended Massive Sequence Views. *IEEE transactions on visualization and computer graphics*, 20(8):1087–1099, 2014.

[111] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[112] Thomas van Dijk, Martin Fink, Norbert Fischer, Fabian Lipp, Peter Markfelder, Alexander Ravsky, Subhash Suri, and Alexander Wolff. Block Crossings in Storyline Visualizations. *Journal of Graph Algorithms and Applications*, 21(5):873–913, 2017.

[113] Manik Varma and Debajyoti Ray. Learning the discriminative power-invariance trade-off. In *Proc. 11th International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[114] John Wenskovitch, Ian Crandell, Naren Ramakrishnan, Leanna House, Scotland Leman, and Chris North. Towards a systematic combination of dimension reduction and clustering in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):131–141, 2018.

[115] Uri Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo/`, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.

[116] Uri Wilensky. Netlogo flocking model. `http://ccl.northwestern.edu/netlogo/models/Flocking/`, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1998.

[117] James Wilkinson. Error analysis. In *Encyclopedia of Computer Science*. Wiley, 2003.

[118] Wolfram Research, Inc. Mathematica, Version 12.0. Champaign, IL, 2019.

[119] Jules Wulms, Juri Buchmüller, Wouter Meulemans, Kevin Verbeek, and Bettina Speckmann. Spatially and Temporally Coherent Visual Summaries. *CoRR*, abs/1912.00719, 2019.

[120] Jin Xie, Guoxion Dai, Fan Zhu, Edward Wong, and Yi Fang. DeepShape: deep-learned shape descriptor for 3D shape retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1335–1345, 2017.

[121] Hao Zhang, Alexander Berg, Michael Maire, and Jitendra Malik. SVM-KNN: discriminative nearest neighbor classification for visual category recognition. In *Proc. 19th Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2126–2136, 2006.

[122] Yu Zhong. Intrinsic shape signatures: a shape descriptor for 3d object recognition. In *Proc. 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–696, 2009.

[123] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

Summary

Stability of Geometric Algorithms

A large amount of data that is collected nowadays is *time-varying*: air traffic, stock prices and weather are all examples of every-day data that changes over time. To aid humans in their decision making, this data has to be analyzed quickly and communicated effectively. Algorithms and visualizations together play an important part in both these tasks; hence a variety of techniques is already developed and widely available. For time-varying data it is especially important to not only communicate data at one point in time, but also to show the evolution of the data over time. To preserve temporal patterns in visualizations, it is important that they are *stable*: small changes in the data should result in small changes in the visualization. Thus far, the lack of theoretical tools to analyze and develop stable algorithms has hindered the development of stable visualizations.

In this thesis we set out to tackle the shortage of theoretical tools by introducing a framework for analyzing the stability of algorithms. More specifically, the framework allows us to better understand the trade-offs between stability and traditional criteria for evaluating algorithms, such as solution quality and running time. For our framework we subdivide algorithms for time-varying data into *stateless*, *state-aware* and *clairvoyant algorithms*, which respectively have access to data at the current time step, data calculated from previous time steps or data at every time step. Depending on the type of algorithm, different levels of stability can be achieved. The framework provides three definitions for measuring stability that each address different aspects. The *event stability* of a problem counts the number of times the combinatorial structure of the output changes. This is closely related to the efficiency of so-called kinetic data structures. The *topological* and *Lipschitz stability* both expect the output of an algorithm to change continuously, by imposing a topology or metric on the output space of the algorithm. Lipschitz stability additionally limits how fast an output can change depending on the change in the input. Because of this constraint, Lipschitz stability comes closest to our intuitive definition of stability, namely small changes in the input should lead to small changes in the solution, and is therefore the preferred type of analysis. However, Lipschitz stability analysis is often prohibitively challenging or infeasible. The other types of stability analysis simplify the stability requirements, making the analysis significantly easier. Although these types of stability analysis do not fully capture all aspects of stability, they do offer useful insight into the interplay between problem instances, solutions, and the optimization function. These insights are invaluable for the development of stable algorithms.

We first show how to use the framework by analyzing the stability of the kinetic *Euclidean minimum spanning tree* problem. The input for this problem consists of a set of moving points, which should be connected by straight lines between the points, while minimizing the total line length. We show how to improve the event stability, while approximating an optimal solution. Furthermore, we give upper and lower bounds on the topological stability for different ways of introducing continuity on spanning trees, and show how to find a simple lower bound on Lipschitz stability when the output changes slowly with respect to the input.

Second, we turn to the kinetic *k-center* problem. Here the input again is a set of moving points, which are covered by a set of k disks or squares whose size should be minimized. For a natural way of introducing continuity, it is impossible to achieve Lipschitz stability for this problem, when k is at least 3. Hence, we analyze only the topological stability. Furthermore, we propose a clairvoyant algorithm to calculate an upper bound on the approximation ratio required for a topologically stable solution to an instance of the k-center problem.

Third, we analyze orientation-based *shape descriptors* on a set of moving points. As the points move, the shape descriptor should give a low-complexity description of the point set. The size of orientation-based shape descriptors is minimized by being in a certain orientation. We investigate the topological and Lipschitz stability of three such shape descriptors, first principal component, oriented bounding box, and covering strip. We start by proving that no topologically stable stateless algorithm can exist that approximates any of these three shape descriptors, and we therefore turn to state-aware algorithms. To analyze the Lipschitz stability we introduce a natural kind of state-aware algorithm, called a *chasing algorithm.* We prove that such an algorithm, which always moves the stable solution towards an optimal solution, approximates an optimal solution well while changing at a speed proportional to the changes in the input.

Finally, we consider an application that requires stability. Inspired by our work on shape descriptors, we develop stable methods to generate a visual summary, a visualization method that shows changes in time-varying data in a static way. The visual summaries we study use one-dimensional representations, such as *linearizations*, of the data at every time step. We propose stable versions of popular dimensionality-reduction techniques, such as principal components, to create 1D representations that do not only improve the summaries visually, but are also measurably of higher quality. An extensive quantitative evaluation of the existing and newly introduced methods shows that stable techniques improve on stability, without sacrificing the ability of the individual linearizations to represent the underlying data truthfully.

Samenvatting

Stabiliteit van Geometrische Algoritmen

Een grote hoeveelheid van alle data die tegenwoordig wordt verzameld is *tijdsafhankelijk*: luchtverkeer, aandeelkoersen en het weer zijn voorbeelden van alledaagse data die verandert over tijd. Om mensen te helpen in hun beslissingen, moet deze data vlug worden geanalyseerd en effectief worden gecommuniceerd. Algoritmen en visualisaties spelen samen een belangrijke rol in deze taken; daarom zijn een divers aantal technieken reeds ontwikkeld and beschikbaar voor gebruik. Voor tijdsafhankelijke data is het in het bijzonder belangrijk om niet alleen data op één bepaald punt in de tijd te communiceren, maar ook de verandering van de data over tijd te tonen. Om deze tijdgerelateerde patronen in visualisaties te behouden, is het belangrijk dat deze *stabiel* zijn: kleine veranderingen in de data moeten resulteren in kleine veranderingen in de visualisatie. Tot zover heeft het gebrek aan theoretische middelen om stabiele algoritmen te analyseren en ontwikkelen, de ontwikkeling van stabiele visualisaties verhinderd.

In deze thesis gaan we dit gebrek aan theoretische middelen aanpakken door een raamwerk te introduceren om de stabiliteit van algoritmen te analyseren. Concreet gezien, dit raamwerk stelt ons in staat om de afwegingen tussen stabiliteit en traditionele criteria voor het evalueren van algoritmen, zoals de kwaliteit van de uitvoer en de looptijd, beter te begrijpen. Binnen ons raamwerk verdelen we algoritmen voor tijdsafhankelijke data onder in *staatloze*, *staatbewuste* en *helderziende algoritmen*, die respectievelijk toegang hebben tot de data op het huidige moment in de tijd, data berekend met de voorgaande tijdstappen, en data op elk moment in de tijd. Afhankelijk van het soort algoritme, kunnen verschillende niveaus van stabiliteit worden behaald. Het raamwerk verstrekt drie definities om stabiliteit te meten, die zich elk op andere aspecten richten. De *gebeurtenisstabiliteit* van een probleem telt het aantal keren dat de combinatorische structuur van de uitvoer verandert. Dit is sterk gerelateerd aan de efficiëntie van zogenoemde kinetische datastructuren. De *topologische* en *lipschitz-stabiliteit* verwachten beide dat de uitvoer van een algoritme continu verandert, door het opleggen van een topologie of metriek op de uitvoer ruimte van het algoritme. Bovendien beperkt lipschitz-stabiliteit hoe snel de uitvoer kan veranderen, afhankelijk van de verandering in de invoer. Door deze beperking komt lipschitz-stabiliteit het dichtst bij onze intuïtieve definitie van stabiliteit, namelijk dat kleine veranderingen in de invoer moeten leiden tot kleine veranderingen in de uitvoer, en is daarom het soort analyse dat de voorkeur heeft. Echter, lipschitz-stabiliteitsanalyse is vaak onoverkomelijk uitdagend of onhaalbaar. De andere soorten stabiliteitsanalyse vereenvoudigen de stabiliteitsvoorwaarden, en maken de analyse significant makkelijker. Hoewel deze soorten stabiliteitsanalyse niet alle aspecten van stabiliteit volledig vangen, bieden ze wel nuttige inzichten in de wisselwerking tussen probleeminstanties, oplossingen en de optimalisatiefunctie. Deze inzichten zijn waardevol voor de ontwikkeling van stabiele algoritmen.

Als eerste laten we zien hoe het raamwerk gebruikt kan worden om de stabiliteit van het kinetische *euclidische minimaal opspannende boom* probleem te analyseren. De invoer voor dit probleem bestaat uit een groep bewegende punten, die moeten worden verbonden door rechte lijnstukken tussen de punten, terwijl de totale lijnlengte geminimaliseerd wordt. We laten zien hoe de gebeurtenis stabiliteit verbeterd kan worden, en tegelijkertijd een optimale oplossing kan worden benaderd. Verder geven we boven- en ondergrenzen op de topologische stabiliteit, voor verschillende manieren waarop opspannende bomen continu kunnen veranderen, en laten we zien hoe een simpele ondergrens op de lipschitz-stabiliteit gevonden kan worden wanneer de uitvoer langzaam verandert ten opzichte van de invoer.

Als tweede richten we ons op het kinetische *k-center* probleem. Hier is de invoer alweer een groep bewegende punten, die bedekt worden door een groep van k schijven of vierkanten, waarvan de grootte geminimaliseerd moet worden. Voor een natuurlijke manier waarop continuïteit geïntroduceerd kan worden, is het onmogelijk om lipschitz-stabiliteit te behalen, wanneer k ten minste 3 is. Daarom analyseren we enkel de topologische stabiliteit. Bovendien bieden we een helderziend algoritme aan om een bovengrens te berekenen op de verhouding tussen een topologisch stabiele benadering en een optimale oplossing voor een instantie van het k-center probleem.

Als derde analyseren we oriëntatie gebaseerde *vormbeschrijvingen* op een groep van bewegende punten. Terwijl de punten bewegen, geeft de vormbeschrijving een lage complexiteit weergave van de groep punten. De grootte van de oriëntatie gebaseerde vormbeschrijving wordt geminimaliseerd door in een bepaalde oriëntatie te zijn. We onderzoeken de topologische en lipschitz-stabiliteit van drie zulke vormbeschrijvingen, eerste hoofdcomponent, georiënteerde begrenzende vak, en bedekkende strook. We bewijzen eerst dat er geen topologisch stabiel staatloos algoritme kan bestaan dat een van de drie vormbeschrijvingen benadert, en we richten ons om die reden op staatbewuste algoritmen. Om de lipschitz-stabiliteit te analyseren introduceren we een natuurlijk soort algoritme dat staatbewust is, genaamd een *achtervolgend algoritme*. We bewijzen dat dit soort algoritme, dat altijd de stabiele oplossing richting een optimale oplossing beweegt, een optimale oplossing goed benadert terwijl het verandert met een snelheid proportioneel aan de verandering in de invoer.

Tenslotte beschouwen we een toepassing die stabiliteit vereist. Geïnspireerd door ons werk aan vormbeschrijvingen, ontwikkelen we stabiele methodes voor het genereren van een visuele overzicht, een visualizatie methode die veranderingen in tijdsafhankelijke data op een statische manier laat zien. De visuele overzichten die we bestuderen gebruiken eendimensionale representaties, zoals *lineariseringen*, van de data op elk moment in de tijd. We bieden stabiele versies van populaire dimensionaliteitsreductie technieken, zoals hoofdcomponenten, om 1D representaties te creëren die niet alleen de overzichten visueel verbeteren, maar ook meetbaar van hogere kwaliteit zijn. Een uitgebreide kwantitatieve evaluatie van de bestaande en nieuw geïntroduceerde methodes laat zien dat de stabiele technieken de stabiliteit verbeteren, zonder het vermogen van de individuele lineariseringen om de onderliggende data waarheidsgetrouw weer te geven, op te offeren.

Curriculum Vitae

Jules Wulms was born on June 25, 1992 in Sittard, the Netherlands. He finished his secondary education in 2010 at Trevianum Scholengroep in Sittard (*summa cum laude*). During his secondary education, he acquired a Cambridge proficiency in English qualification. He then studied computer science at TU Eindhoven in Eindhoven, the Netherlands, and spent the winter semester of 2013 at Mälardalens Högskola in Västerås, Sweden. This resulted in a Bachelor's degree in Computer Science and Engineering (*cum laude*) in 2014, and a Master's degree in Computer Science and Engineering (*cum laude*) in 2016. His Master's thesis, on the topic of lower bounds for kernelization algorithms based on protrusion replacement, was supervised by Bart Jansen. In September 2016, he started as a PhD student at TU Eindhoven, under the supervision of Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. The main results of his research have been presented in this thesis.

Titles in the IPA Dissertation Series since 2017

M.J. Steindorfer. *Efficient Immutable Collections*. Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors*. Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT)*. Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data*. Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities*. Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too*. Faculty of Science, Mathematics and Computer Science, RU. 2017-08

A.M. Şutîi. *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod*. Faculty of Mathematics and Computer Science, TU/e. 2017-09

U. Tikhonova. *Engineering the Dynamic Semantics of Domain Specific Languages*. Faculty of Mathematics and Computer Science, TU/e. 2017-10

Q.W. Bouts. *Geographic Graph Construction and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2017-11

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems*. Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

F. Yang. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi*. Faculty of Mathematics and Computer Science, TU/e. 2018-11

L. Swartjes. *Model-based design of baggage handling systems*. Faculty of Mechanical Engineering, TU/e. 2018-12

T.A.E. Ophelders. *Continuous Similarity Measures for Curves and Surfaces*. Faculty of Mathematics and Computer Science, TU/e. 2018-13

M. Talebi. *Scalable Performance Analysis of Wireless Sensor Network*. Faculty of Mathematics and Computer Science, TU/e. 2018-14

R. Kumar. *Truth or Dare: Quantitative security analysis using attack trees*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

M.M. Beller. *An Empirical Evaluation of Feedback-Driven Software Development*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

M. Mehr. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems.* Faculty of Mathematics and Computer Science, TU/e. 2018-17

M. Alizadeh. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations.* Faculty of Mathematics and Computer Science, TU/e. 2018-18

P.A. Inostroza Valdera. *Structuring Languages as Object-Oriented Libraries.* Faculty of Science, UvA. 2018-19

M. Gerhold. *Choice and Chance - Model-Based Testing of Stochastic Behaviour.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

A. Serrano Mena. *Type Error Customization for Embedded Domain-Specific Languages.* Faculty of Science, UU. 2018-21

S.M.J. de Putter. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow.* Faculty of Mathematics and Computer Science, TU/e. 2019-01

S.M. Thaler. *Automation for Information Security using Machine Learning.* Faculty of Mathematics and Computer Science, TU/e. 2019-02

Ö. Babur. *Model Analytics and Management.* Faculty of Mathematics and Computer Science, TU/e. 2019-03

A. Afroozeh and A. Izmaylova. *Practical General Top-down Parsers.* Faculty of Science, UvA. 2019-04

S. Kisfaludi-Bak. *ETH-Tight Algorithms for Geometric Network Problems.* Faculty of Mathematics and Computer Science, TU/e. 2019-05

J. Moerman. *Nominal Techniques and Black Box Testing for Automata Learning.* Faculty of Science, Mathematics and Computer Science, RU. 2019-06

V. Bloemen. *Strong Connectivity and Shortest Paths for Checking Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-07

T.H.A. Castermans. *Algorithms for Visualization in Digital Humanities.* Faculty of Mathematics and Computer Science, TU/e. 2019-08

W.M. Sonke. *Algorithms for River Network Analysis.* Faculty of Mathematics and Computer Science, TU/e. 2019-09

J.J.G. Meijer. *Efficient Learning and Analysis of System Behavior.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-10

P.R. Griffioen. *A Unit-Aware Matrix Language and its Application in Control and Auditing*. Faculty of Science, UvA. 2019-11

A.A. Sawant. *The impact of API evolution on API consumers and how this can be affected by API producers and language designers*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12

W.H.M. Oortwijn. *Deductive Techniques for Model-Based Concurrency Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-13

M.A. Cano Grijalba. *Session-Based Concurrency: Between Operational and Declarative Views*. Faculty of Science and Engineering, RUG. 2020-01

T.C. Nägele. *CoHLA: Rapid Co-simulation Construction*. Faculty of Science, Mathematics and Computer Science, RU. 2020-02

R.A. van Rozen. *Languages of Games and Play: Automating Game Design & Enabling Live Programming*. Faculty of Science, UvA. 2020-03

B. Changizi. *Constraint-Based Analysis of Business Process Models*. Faculty of Mathematics and Natural Sciences, UL. 2020-04

N. Naus. *Assisting End Users in Workflow Systems*. Faculty of Science, UU. 2020-05

J.J.H.M. Wulms. *Stability of Geometric Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2020-06