Proceedings of the 28th Fall Workshop on Computational Geometry (FWCG 2018).

Queens College, City University of New York, Flushing, NY

Program Committee:

1. Chao Chen (Stony Brook University)
2. Hu Ding (Michigan State University)
3. Mayank Goswami (Chair, Queens College, City University of New York)
4. Jonathan Lenchner (IBM research Africa)
5. Joseph S.B. Mitchell (Stony Brook University)
6. Miguel A. Mosteiro (Pace University)
7. Megan Owen (Lehman College, City University of New York)
8. Christiane Schmidt (Linkoping University, Sweden)
9. Jack Snoeyink (Univeristy of North Carolina)

Fibres of Failure: Classifying errors in predictive processes

Leo Carlsson Gunnar Carlsson Mikael Vejdemo-Johansson

***Abstract*—Predictive models are used in many different fields of science and engineering and are always prone to make faulty predictions. These faulty predictions can be more or less malignant depending on the model application.**

We describe Fibres of Failure (FiFa), a method to classify failure modes of predictive processes using the Mapper algorithm from Topological Data Analysis. Our method uses Mapper, an algorithm from Topological Data Analysis (TDA), to build a graph model of input data stratified by prediction error. Groupings found in high-error regions of the Mapper model then provide distinct failure modes of the predictive process that can be further analyzed in subsequent steps. We demonstrate two ways to use the failure mode groupings: either to produce a correction layer that adjusts predictions by similarity to the failure modes; or to inspect members of the failure modes to illustrate and investigate what characterizes each failure mode. We demonstrate FiFa on two scenarios; a Convolutional Neural Network (CNN) predicting MNIST images with added noise, and an energy model predicting the end-point temperature of molten steel in an Electric Arc Furnace (EAF) producing clean steel.

I. Introduction

Predictive processes are used across many different fields in both science and engineering. All predictive processes produce some sort of error which makes them less practical in use. However, most predictive processes produce consistent error of single or multiple distinct types. If one could exploit these consistent errors then it would be possible to improve the predictions as an additional step on top of the predictive process.

We introduce Fibres of Failure (FiFa) as a method to classify consistent error from predictive processes, and analyze these error types both qualitatively and quantitatively. In the qualitative case we focus on domain-specific reasons as to why the consistent error(s) persist. In the quantitative case we exploit the consistent bias to adjust predicted values, as an additional step, closer to their true values.

The FiFa method builds on Mapper, an algorithm from Topological Data Analysis that constructs a graph (or simplicial complex) model of arbitrary data.

Our running examples to demonstrate the FiFa method will be first a Convolutional Neural Network (CNN) model predicting hand-written digits from the MNIST data set, trained on clean images but used on noisy images; and second an Electric Arc Furnace (EAF) energy model predicting the end-point temperature of molten steel.

This project has received support from: Hugo Carlssons Stiftelse för Vetenskaplig Forskning; a PSC-CUNY Award, jointly funded by The Professional Staff Congress and The City University of New York; and a donation of the Titan X Pascal GPU by the NVIDIA Corporation.

II. Preliminaries

We concern ourselves with failures in predictive processes. Given some observable process that given known input values X produces some distribution of observable values $Y \sim \mathscr{D}(X)$, a predictive process can be considered to be a method for creating a probability distribution of predicted values $\hat{Y} \sim \hat{\mathscr{D}}(X)$.

With past observations X_i, Y_i we can establish a measure of prediction error $\mathscr{E}_i = \mathscr{E}(Y_i, \hat{Y})$. In our work, the focus is on finding regions of input space that tend to produce large prediction errors: X_i such that $|\mathscr{E}_i|$ is large.

To illustrate and evaluate our approach, we will study two settings:

1) A CNN model is trained on the MNIST dataset, achieving a high performance in predicting correct digit classifications from handwritten digits. We then expose the model to severely noise corrupted versions of the MNIST digits and analyze the resulting failure. Here, the model produces a probability distribution on the set $\{0,1,2,3,4,5,6,7,8,9\}$ and we take as a measure of error the predicted probability of the correct class: this value will be low in case of an error, and high in case of a correct prediction.
2) Data from an Electric Arc Furnace. The furnace uses a temperature model to predict internal temperature T_p of the steel in order to stop the furnace after all steel has melted but before before the molten steel reaches elevated temperatures that dramatically increases the wear and tear of the furnace. Before pouring, a temperature measure is taken to establish a ground truth measurement of temperature T_m. The prediction error measure is $\Delta T = T_p - T_m$.

A. Mapper

Mapper (Singh et al, 2007) is an algorithm that constructs a graph (more generally a simplicial complex) model for a point cloud data set. The graph is constructed systematically from some well defined input data.

Let X be a finite metric space. The following steps construct the *Mapper complex*:

1) Choose a collection of maps $f_1, \ldots, f_k : X \to \mathbb{R}$, or equivalently some $f : X \to \mathbb{R}^k$. These are usually referred to as *lenses* or *filters*.

2) Choose a covering $\mathbb{U} = \{U_1, \dots\}$ of $\mathbb{R}^k$: an overlapping partition of possible filter value combinations.
3) Pull the covering back to a covering $\mathbb{V}$ of X, where $V_i \in V = f^{-1}(U_i)$.
4) Refine the covering $\mathbb{V}$ to a covering $\hat{\mathbb{V}}$ by clustering each V_i.
5) Create the *nerve complex* of the covering $\hat{\mathbb{V}}$: as vertices of the complex we choose the indexing set of $\hat{\mathbb{V}}$, and a simplex $[i_0, \dots, i_j]$ is included if $\hat{V}_{i_0} \cap \dots \cap \hat{V}_{i_j} \neq \emptyset$.

The filters act as measures of enforced separation: data points with sufficiently different values for the filter function are guaranteed to be separated to distinct vertices in the Mapper complex, while the nerve complex construction ensures that connectivity information is not lost in the process.

More detailed expositions can be found in (Singh et al, 2007) and (Carlsson, 2009)

For our work we are using the Ayasdi implementation of Mapper.

III. Proposed method

A. Mapper on prediction failure

The *filters* in the Mapper function have the effect of ensuring separation of features in the data that are separated by the filter functions themselves. In the setting of prediction failures, we leverage this feature to create Mapper models that enforce a separation on prediction errors, allowing the subsequent analysis to identify contiguous regions of input space with consistent and large prediction errors.

We name the process of using Mapper with prediction error as a filter in order to classify prediction failures the *Fibres of Failure* method, and the resulting Mapper model we name a FiFa *model*.

B. Extracting subgroups

Subgroups of the FiFa model with tight connectivity in the graph structure and with homogeneous and large average prediction failure per component cluster provide a classification of failure modes. These can be selected either manually, or using a community detection algorithm.

When selecting failure modes manually, a visualization such as in Figure 2 is most helpful. Here, flares (tightly connected subgraphs emanating from a core, such as Group 40) or tightly connected components, loosely connected to surrounding parts of the graph, are the most compelling characterizations of a good failure mode subgroup. When extracting subgroups manually, the intent is always to extract groups where the prediction error is as close to constant as possible.

C. Quantitative: Model correction layer

Once failure modes have been identified, one way to use the identification is to add a correction layer to the predictive process. Use a classifier to recognize input data similar to a known failure mode, and adjust the predictive process output according to the behavior of the failure mode in available training data.

1) Train classifiers: For our illustrative examples, we demonstrate several "one vs rest" binary classifier ensembles where each classifier is trained to recognize one of the failure modes (extracted subgroups) from the Mapper graph. We demonstrate performance of FiFa for model correction using Linear SVM, Logistic Regression, and Naive Bayes classifiers.

2) Evaluate bias: A classifier trained on a failure mode may well capture larger parts of test data than expected. As long as the space identified as a failure mode has consistent bias, it remains useful for model correction: by evaluating the bias in data captured by a failure mode classifier we can calibrate the correction layer.

3) Adjust model: The actual correction on new data is a type of ensemble model, and has flexibility on how to reconcile the bias prediction with the original model prediction – or even how to reconcile several bias predictions with each other. For the CNN example in this paper we choose to override the CNN prediction with the observed ground truth in the failure mode from the training data used to create the classifier. For regression tasks, such as the EAF energy model case, we used the average of the failure mode training group as an offset, motivated by Type S error, to subtract from the model prediction.

D. Qualitative: Model inspection

Identifying distinct failure modes and giving examples of these is valuable for model inspection and debugging. Statistical methods, such as Kolmogorov-Smirnov testing, can provide measures of how influential any one feature is in distinguishing one group from another and can give notions of what characterizes any one failure mode from other parts of input space. With examples and distinguishing features in hand, we can go back to the original model design and evaluate how to adapt the model to handle the failure modes better.

Much of the work in interpretability for machine learning provides tools to inspect examples, and for providing a model explanation for a specific example. These work well in conjunction with FiFa to find explanations for the identified failure modes.

IV. Experiments

A. CNN model on MNIST data

1) Protocol: We created a CNN model with a topology shown in Figure 1. The network topology and parameters was chosen arbitrarily with the only condition that it performs well on the original MNIST test data set. The activation functions was 'Softmax' for the classification layer and 'ReLU' for all other layers. The optimizer was Adadelta with learning rate $lr = 1.0$, $\rho = 0.95$, and $\varepsilon = 1e-7$ (Zeiler, 2012). We trained the model on 60,000 clean MNIST training images through 12 epochs and tested it on 10,000 clean MNIST images. The accuracy on the test-set of 10,000 clean MNIST images was 99.05%. We created 10,000 corrupt MNIST images using 25% random binary flips on the clean test images[source for code in

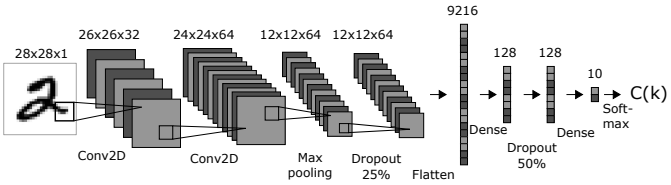


Fig. 1. The topology for the CNN model. The numbers display the dimension of each layer in the model. The abbreviations, such as Conv2D, describes the specific transformations performed between layers in the model. The activation functions for the classification layer was 'Softmax' and for the other layers 'ReLU'. The optimizer used was 'Adadelta' (Zeiler, 2012).

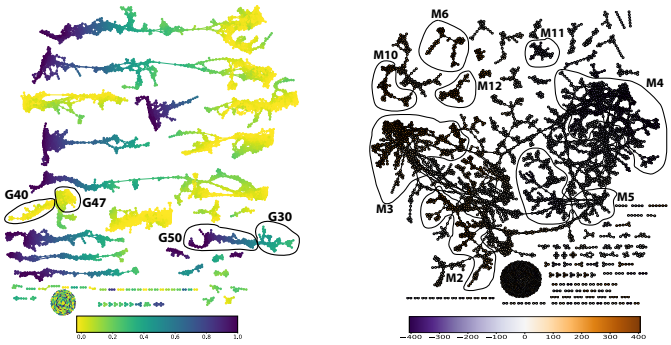


Fig. 2. Left: the Mapper graph for the CNN on MNIST dataset colored with probability of predicting the ground truth digit. The colorbar is for interpreting the values of the coloring. The circled nodes and edges are the groups Group30, Group40, Group47, and Group50. The 5-fold Mapper graphs are shown in the Supplement. Right: The Mapper graph for the EAF energy model dataset colored with ΔT. The color bar is for interpreting the ΔT values. The 8 manually picked groups are circled with names.

supplement]. The accuracy on the corrupt MNIST images was 40.45%.

To create the Mapper graph we used the following:

- **Filters:** Principal Component 1, probability of Predicted digit, probability of Ground truth digit, and Ground truth digit. Our measure of predictive error is the probability of Ground truth digit.
- **Metric:** Variance Normalized Euclidean
- **Variables:** 9472 network activations: all activations after the Dropout layer that finishes the convolutional part in the network and before the softmax layer that provides the final predictions.
- **Instances:** In addition to the test set with 10,000 images, we created 10,000 corrupted images. We performed 5-fold cross-validation for calibration on this test set of 20,000 images.

We purposely omitted the activations from the Dense-10 layer as input variables because of the direct reference to the probabilities for both the ground truth digit and the predicted digit.

The following variables were used in filter functions or in the subsequent analysis, but were *not used* to create the FiFa model:

- **10 activations** from the Dense-10 layer, which consists of the probabilities for each digit, 0-9.

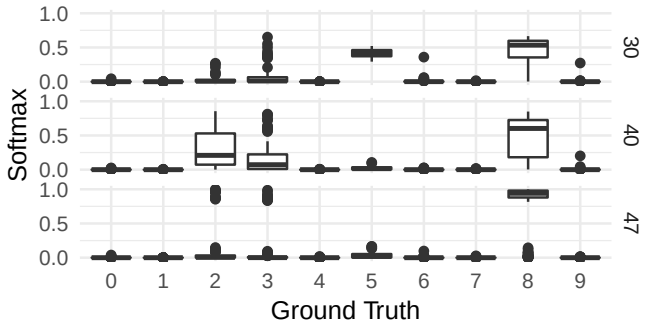


Fig. 3. The failure modes for a ground truth of 5. We see the distributions of predictions for the three failure modes: only group 30 attaches any significant likelihood to the digit 5 at all, while all three favor 8. For group 40, the digits 2 and 3 are also commonly suggested, while this happens somewhat more rarely in groups 30 and 47.

- **784 pixel values** representing the flattened MNIST image of size 28x28x1.
- **6 variables**: prediction by the CNN model, ground truth digit, corrupt or original data (binary), correct or incorrect prediction(binary), probability of the Predicted digit (highest value of the Dense-10 layer), and probability of ground truth digit.

Hence, the total number of variables in our analysis were 10272.

To extract failure modes from the FiFa model we used a supervised community detection method to find groups of approximately constant prediction error.

From partitioned groups, we retain as failure modes those groups that have at least 15 data points and have less than 99.05% correct predictions, which is the accuracy of the CNN model on the original MNIST test data.

We evaluated the usefulness of the extracted failure modes by training classifiers on detecting failure mode membership for activation sets from the network. These were evaluted on a set of 10,000 new corrupted images using the same type of 25% binary flips on the original MNIST test datasets. Between Linear SVM, Logistic Regression and Naive Bayes, the resulting performance was similar between all three options. For the two linear types, less than 1% of clean images were picked up by the classifiers, and prediction accuracy on corrupted images increased by almost 20%pt.

Focusing on failure to detect the digit 5, we identified three important failure modes. Their prediction distributions can be seen in Figure 3. For a qualitative analysis, we inspected saliency maps (Simonyan et al, 2013) for highly influential activations for our failure modes. These saliency maps, when investigating these three failure modes include on the one hand several activations that seem to code for 5-ishness, and on the other hand several activations that seem to pick up on noise closing up loops in the shape of the 5s. We would refer to our figshare repository for illustrations of these saliency maps: https://doi.org/10.6084/m9.figshare.6476426 .

B. EAF energy model

1) Protocol: For the EAF energy model case, we used the following parameter to create the Mapper graph:

- **Filters:** Principal Component 1, Principal Component 2, ΔT, and T_{m}.
- **Metric:** Variance Normalized Euclidean
- **Variables:** 77 variables, logged before the temperature measurement.
- **Instances:** We used 5163 data points that was randomly shuffled and split into 80/20 training and test data. The 4130 training data points was used to create the Mapper graph as well as the piecewise linear models. The remaining 1033 test data points was used to evaluate the piecewise linear models.

The use of ΔT as one of the lenses guarantees that Mapper separates regions of high ΔT from regions of low ΔT while the use of pre-measurement data makes the model usable for online corrections. An additional 24 variables, logged after the temperature measurement, was used to analyze the network. Hence, a total of 101 variables were used.

Automated extraction of failure modes failed for the EAF data. Instead, we extracted subgroups manually using the following heuristics:

1) An average ΔT of above 50°C or below -50°C.
2) Nodes have consistently large $|\Delta T|$, which distinguished the group from the neighboring clusters.
3) As large group as possible while still maintaining heuristic 1 and 2

Using the 4130 training data points, we trained classifiers to recognize failure mode membership.

To illustrate a quantitative application, we added a correction layer, adjusting the temperature prediction by the mean ΔT from the training set. Limiting our attention to the 3 most extreme of the 8 identified groups, the adjustment changed the prediction error on the test set by:

- M4: $-219°\mathrm{C} \rightarrow 13°\mathrm{C}$ (std.dev. 56.1)
- M10: $173°\mathrm{C} \rightarrow 44°\mathrm{C}$ (std.dev. 49.2)
- M11: $-242°\mathrm{C} \rightarrow -51°\mathrm{C}$ (std.dev. 42.6)

For a qualitative evaluation, we identified high impact variables and were able to identify as especially impactful – for the group we labelled M3 two different identifiers of the raw material type in the furnace charge and one of the measures of expected energy consumption: having the steel type out of balance led to a large underestimation of energy consumption, which produced too low an energy estimate.

V. Summary

The Fibres of Failure (FiFa) approach identifies distinct failure modes by using the topological method Mapper for robust grouping of observations with enforced separations parametrized by an error measure on the data.

When applying the method to noisy digits displayed to a CNN model trained on noise-free MNIST digits we could see an improvement by almost 20%pt on accuracy on corrupted image data, and when applying the method to temperature prediction in Electric Arc Furnaces for steel smelting we could improve temperature prediction with up to approximately 200°C.

Furthermore, in both the CNN model and the temperature prediction cases, analyzing features that distinguish the failure modes from non-failing cases led to specific observations of factors contributing to the failures: loop-closing noise for noisy MNIST task and material compositions leading to optimistic energy use predictions for the furnace. Used in this way, the failure modes extracted by FiFa acts as a guide for where to focus inspection work to best leverage understanding for the data.

A. Acknowledgements

This project is in collaboration with Outokumpu Stainless AB and with Ayasdi Inc. We would like to thank both companies and our contacts: Jesper Janis, Gunnar Lindstrand, Pär Ljungqvist and Devi Ramanan for support, access to data and software, and fruitful discussions.

References

Carlsson G (2009) Topology and data. American Mathematical Society 46(2):255–308

Simonyan K, Vedaldi A, Zisserman A (2013) Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:13126034

Singh G, Mémoli F, Carlsson G (2007) Topological Methods for the Analysis of High Dimensional Data Sets and 3d Object Recognition. In: SPBG, pp 91–100, URL http://comptop.stanford.edu/preprints/mapperPBG.pdf

Zeiler MD (2012) Adadelta: An adaptive learning rate method. arXiV

Approximating Global Optimum for Probabilistic Truth Discovery ⋆

Shi Li, Jinhui Xu, and Minwei Ye

State University of New York at Buffalo
{shil,jinhui,minweiye}@buffalo.edu

Overview

Truth discovery has received a great deal of attention in recent years in databases, data crowdsourcing, machine learning and data mining. It emerges from various practical scenarios such as copying detection, data fusion and conflicting information resolving on the web. In a typical scenario, the unknown truth for one or multiple objects can be viewed as a vector in a high-dimension space. The information about the truth vector may come from multiple sources. Those sources may be inaccurate, conflicting or even biased from the beginning if they come from subjective evaluation. Our goal is to infer the truth vector from these noisy information.

A naive method for this problem is to take the average of all the vectors from sources as the the ground truth (for coordinates correspondent to categorical data, take the majority vote). However, this approach, which inherently treats all sources as equally important, is vulnerable to unreliable and malicious sources. Such sources can provide information that pulls the average away from the truth. A more robust type of approaches is to give weights to sources to indicate their reliability and use the weighted average or weighted majority as the ground truth. However, since the weights are often unknown, the goal of finding the ground truth is coupled with the task of reliability estimation. This type of approaches is referred as a *truth discovery* approach. Among all, there are two competing and sometimes complementary frameworks that are widely accepted and used for different data types.

Weight-based Truth Discovery In this framework, both the truth and the weights are treated as variables. An objective function is defined on these variables. A typical setting might be like this [4]:

$$\text{Given } \{p_i\}_{i=1}^n \subset \mathbb{R}^d, \quad \text{find } x \in \mathbb{R}^d, \{w_i\}_{i=1}^n \subset \mathbb{R}^d$$
$$\text{to minimize } \sum_{i=1}^n w_i \|x - p_i\|^2 \quad \text{s.t. } \sum_{i=1}^n e^{-w_i} = 1$$

The problem is usually solved by alternating minimization algorithm. In each iteration, the algorithm fixes one set of variables (either the truth variables, or

⋆ A preliminary version of this work has appeared in the 24th International Computing and Combinatorics Conference (COCOON'18)

the weight variables) and optimizes the other. This procedure continues until a stable solution is reached. Many existing methods follow this framework and justify themselves by experimenting with different types of real-world datasets. However, none of these methods provides any theoretical guarantee regarding the quality of solution. Recently, Ding et al. [2] gave the first algorithm that achieves a theoretical guarantee (*i.e.,* a $(1+\epsilon)$-approximation) for the model above. Later, Huang et al. [6] further improved the running time to near quadratic.

Probabilistic Truth Discovery Probabilistic models lie in a different category of models for truth discovery. They were also studied extensively in the literature. Instead of giving weights to indicate the reliability of all sources, these models assume that the information for each source is generated independently from some distribution that depends on the truth and the reliability of the source. Then the goal under these models is to find the truth that maximizes the likelihood of the generated information from all sources. The probabilistic models have been shown to outperform the weight-based methods on numerical data. They also prevail other models in the case where sources come from subjective evaluation. For the quality of the optimization, [5] gave an iterative algorithm with guaranteed fast convergence to a local optimum.

Model and result

We propose a probabilistic truth discovery model, reformulate it as an optimization problem and give a polynomial-time approximation scheme to solve it.

We first set the stage for the problem. The unknown truth can be represented as a d dimensional vector p^*, as justified in [4]. There are n sources, and the observation/evaluation made by the i-th source is denoted as p_i which also lies in the d dimensional space $\mathbb{R}^d$. In our model, we assume that each observation/evaluation is a random variable following a multi-variate Gaussian distribution centered at the truth p^* with covariance $\sigma_i^2 I_d$.[1] Each unknown parameter $\sigma_i \geq 0$ represents the reliability of the source; the smaller the variance, the more reliable the source is.

We formulate the problem as finding the $(p^*, \sigma = (\sigma_i)_{i\in[n]})$ that maximizes the likelihood of the random procedure generating $\{p_i\}_{i=1}^n$.We impose a hyper-parameter $\sigma_0 > 0$ and require $\sigma_i \geq \sigma_0$ for every $i \in [n]$. It is naturally interpreted as an upper bound of the reliability of all sources.

Given the set of observation $P = \{p_i\}_{i=1}^n \subset \mathbb{R}^d$ under this probabilistic model and a hyper-parameter σ_0, we need to find a point x that maximizes the following

[1] For categorical data, the Gaussian distribution may cause fractional answers, which can be viewed as a probability distribution over possible truths. In practice, variance for different coordinates of the truth vector may be different and there might be some non-zero covariance between different coordinates; however, up to a linear transformation, we may assume the covariance matrix is $\sigma_i^2 I_d$.

likelihood function:

$$\prod_{i=1}^{n} \mathcal{N}(p_i \mid x, \sigma_i^2 I_d) = \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi}\sigma_i}\right)^d \exp\left[-\frac{\|p_i - x\|^2}{2\sigma_i^2}\right].$$

Taking negative logarithm, we obtain the following optimization problem:

$$\min_{x\in\mathbb{R}^d,\sigma} \left\{ \frac{nd}{2}\ln(2\pi) + \sum_{i=1}^{n}\left(d\ln\sigma_i + \frac{\|p_i - x\|^2}{2\sigma_i^2}\right)\right\}, \quad \text{s.t.} \quad \sigma_i \geq \sigma_0, \forall i \in [n]. \tag{1}$$

After some scaling and simplification, the problem reduces to:

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{n} f_{truth}(\|x - p_i\|) \quad \text{where} \quad f_{truth}(\ell) = \begin{cases} \ell^2 & 0 \leq \ell < 1 \\ 1 + \ln \ell^2 & \ell \geq 1 \end{cases}. \tag{2}$$

which is a special case of a more general class of optimization problems as follows,

$$\text{Given } \{p_1, p_2, \cdots, p_n\} \subset \mathbb{R}^d, \text{ find } x \in \mathbb{R}^d \text{ to minimize } \sum_{i=1}^{n} f(\|x - p_i\|),$$

where f is a function satisfying the following three properties.

Property 1. (Regularity) f is a continuous, non-negative, monotonically increasing function.

Property 2. (Sub-proportionality)[2] $\exists \alpha \geq 1 : f(kx) \leq k^\alpha f(x)$ for any $k \geq 1$,$x \geq 0$. We say α is the *proportional degree* of f if it is the smallest α satisfies such property.

Property 3. The function f can be computed in polynomial time with respect to the size of the input. The inverse of f, defined as $f^{-1}(y) = \sup_x\{x : f(x) = y\}$, should also be able to calculate in polynomial time w.r.t to the size of x when $y \leq 2f(x)$.

This general problem encloses as special cases the classic 1-median and 1-mean problems, and the more general problem of minimizing p-th power of distances. Moreover, by considering the corresponding functions with an upper-threshold, i.e, $f(\ell) = \min\{\ell, B\}$, $f(\ell) = \min\{\ell^2, B\}$ and $f(\ell) = \min\{\ell^p, B\}$, one can capture the outlier versions of all these problems.

We give a sampling-based method that solves the above optimization problem up to a factor of $1+\epsilon$ for any $\epsilon > 0$ in quadratic running time. Thus, it not only solves our truth discovery problem but also gives a unified approach to solve all the above problems under this framework.

Theorem 1. *Let $0 < \epsilon \leq 1$. Let P be a set of n points in $\mathbb{R}^d$ and $G(x) = \sum_{p\in P} f_{truth}(\|x - p\|)$. A $(1+\epsilon)$-approximate solution can be obtained in time $O(2^{(1/\epsilon)^{O(1)}}d + n^2 d)$. In addition, if we replace f_{truth} by any other functions that satisfy property 1,2,3, the same result applies.*

[2] Also referred as polynomial growing function or Log-Log Lipschitz function in literature.

Our Techniques

One property that we *do not* impose on the function f is convexity. Without the convexity property, iterative approaches such as gradient descent and EM do not guarantee the global optimality. General coreset technique which reduces the size of the problem will not work, either. The dimensionality is not reduced by those techniques so that the problem is still hard even for the coreset.

Instead of using methods in continuous optimization or general sampling technique, our algorithm is based on the elegant method Badoiu, Har-Peled and Indyk developed to give fast algorithms for many clustering problems [1, 3]. Roughly speaking, [1] showed that a small set of sample points X can guarantee that the affine subspace $\text{span}(X)$ contains a $(1+\epsilon)$ approximate solution for these clustering problems. Therefore both the size and the dimensionality can be reduced. The approximate solution is then obtained by grid search inside $\text{span}(X)$.

Directly applying [1] does not work for general cost function. The key issue is to settle the parameter (call it L) that determines how close the affine subspace should be to the optimal point. This parameter also determines the grid size in the final grid search step. L can't be too large, otherwise the final result of the grid search won't be close enough to the optimal and therefore won't be an approximate solution. It can't be too small either: smaller L implies more sample points and the size of the grid might be exponential with respect to the input size. We first settled the existence of such L in the general function case. Then we provided a method to search the value of L.

References

1. M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 250–257. ACM, 2002.
2. H. Ding, J. Gao, and J. Xu. Finding global optimum for truth discovery: Entropy based geometric variance. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 51. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
3. A. Kumar, Y. Sabharwal, and S. Sen. Linear time algorithms for clustering problems in any dimensions. In *International Colloquium on Automata, Languages, and Programming*, pages 1374–1385. Springer, 2005.
4. Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1187–1198. ACM, 2014.
5. H. Xiao, J. Gao, Z. Wang, S. Wang, L. Su, and H. Liu. A truth discovery approach with theoretical guarantee. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1925–1934. ACM, 2016.
6. H. D. Ziyun Huang and J. Xu. Faster algorithm for truth discovery via range cover. In *Proceedings of Algorithms and Data Structures Symposium (WADS 2017)*, pages 461– 472, 2017.

Time Window Fréchet and Metric-Based Edit Distance for Passively Collected Trajectories

Jiaxin Ding
Stony Brook University

Jie Gao
Stony Brook University

Steven Skiena
Stony Brook University

1 INTRODUCTION

Advances in localization techniques and wireless technologies have allowed for the collection of a huge volume of trajectories of pedestrians and vehicles. The rich knowledge in human mobility data provides great opportunities to mine interesting patterns, that can be useful for numerous applications, including but not limited to traffic management, urban planning, and the scheduling of autonomous vehicles. In many scenarios, mobile users' trajectories are collected passively, i.e., inferred from connection traces with WiFi access points, cellular towers, or through transactions of credit cards or transit cards in public transportation systems. These records consist of users' locations and the corresponding time stamps, so the sequence of records produces a good approximation of mobile user trajectory. Prior studies of trajectories collected passively include the study of human mobility traces of over 100, 000 mobile phone users [26] and the study of about 95, 000 users in a large university campus [46].

Trajectories collected through passive sensing are unique in many ways. First the collected trajectories respect the density of wireless checkpoints (WiFi, cellular towers, etc) and have much lower resolution than GPS traces. Missing data is the norm, for example, in regions without WiFi access points. It is an interesting problem to properly handle the time-stamp information. First, time stamp labels cannot be ignored but it is preferred to allow flexibility in handling time stamps. Accurate spatial temporal information (i.e., the location of a person at a particular time) can be considered sensitive and private. With long term trajectory data, frequent locations [26, 40], co-locations [22], and specific patterns [20, 36] of users can be easily learned, which can be used to identify a user, breach their social ties and locate their whereabouts at any time. Thus location/time-stamp data in published data sets is often intentionally perturbed or generalized [38]. Further, although human mobility shows great regularity and repetition, people have a fair amount of flexibility in daily routines. For example, it is common that a user goes to work every day but the time of leaving home may fluctuate.

With passively collected trajectories, one can discover interesting mobility patterns by clustering trajectories into groups of similar ones, for which we need a measure of similarity. There has been much work on distances to measure similarity between curves. For example, Hausdorff distance measures the maximum distance of all points on one curve to their nearest point on the other curves. Fréchet distance measures the minimum of maximum length between all possible pairing/coupling of points on the two curves.

These distances for curves when applied to data mining in passively collected human trajectory data are not ideal. First, all these distances focus on the *shape* of the curves but ignore the time stamp information. But time dimension is important for understanding group mobility and traffic. Two individuals, traveling along the same route but at totally different time frames shall not be considered similar. The duration spent at a location is a crucial attribute for important places for a user and the patten of visiting a sequence of locations is useful to infer the semantics of the trip.

In this paper we develop alternative similarity measures. We explored in two directions.

- *Time-window Fréchet distance.* In the first variant, we introduce time-window Fréchet distance, where only pairs of points within a time window δ can be coupled. Further, we show an algorithm to compute the time window Fréchet distance with running time related to the complexity of the input curves.
- *Metric-based edit distance.* In the second variant, we take uniform sampling along the time dimension and use a combinatorial representation as a string of cells/towers/APs visited by the user and the edit distance – the minimum number of changes (insertions or deletions) to turn one sequence to another. We consider a variant of edit distance by incorporating the underlying metric in the calculation of the edit distance. Specifically, the cost of inserting or deleting a symbol is based on the change of the metric distance before and after the operation. This will be able to handle the issue of missing data. We also discuss algorithms for computing the metric based edit distance.

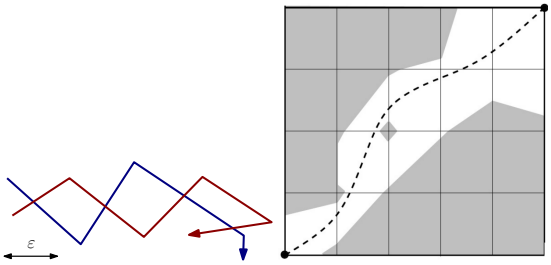
Next, we talk about clustering motion trajectories into meaningful clusters. We use the notion of k-gather clustering [3], which is to minimize the radius of any cluster, such that each trajectory is in a cluster of at least k traces. The requirement of k is needed to define group motion and often adopted to define popular/meaningful traffic patterns. There is also additional benefit of k-anonymity [42] if only a summary of each cluster is reported/shared with the public – one cannot tell a specific user from a group of k trajectories. It is known that k-gathering for points in a metric space is NP-hard and there is a 2-approximation algorithm [3]. We extend the hardness proof to the case of trajectories under edit distance, metric based edit distance and the Jaccard distance.

Related work see Appendix D.

2 TIME-WINDOW METRICS

2.1 Fréchet Distance

The Fréchet distance is one of the most popular distance measure of two curves in space. It can be intuitively understood as a man traversing a finite curved path while walking his dog on a leash, with the dog traversing a separate path. Both the man and the dog need to walk forward but can take any speed at any time. The Fréchet distance is the minimum length of the leash needed to finish the walk. An approximation and simpler version of the Fréchet



(a) Two curves (b) Free Space

Figure 1: Given ε, the white area in the left figure denotes the free space. The horizontal axis corresponds to the red curve and the vertical axis corresponds to the blue curve.

distance between polygonal curves is the discrete Fréchet distance, which considers only positions of the leash whose endpoints are located at vertices of two polygonal curves.

A curve in $\mathbb{R}^2$ is a continuous map $[0,1] \to \mathbb{R}^2$. A re-parameterization of $[0,1]$ is a continuous, non-decreasing, surjection $[0,1] \to [0,1]$.

Definition 2.1 (Fréchet distance). Let A, B be two given curves in $\mathbb{R}^2$, and α, β be re-parameterizations of A, B. The Fréchet distance $\delta_F(A,B)$ is defined as

$$\delta_F(A,B) = \inf_{\alpha,\beta} \max_{s\in[0,1]} \{d(A(\alpha(s)), B(\beta(s)))\},$$

where $d(\cdot,\cdot)$ is the distance between two points.

It is often easier to understand the Fréchet distance between polygonal curves by the free space diagram. A polygonal curve A is a continuous map $[0,n] \to \mathbb{R}^2$, $A(i) = a_i \in \mathbb{R}^2$ is the ith vertex. The map from interval $[i, i+1]$ to the ith line segment $A(i)A(i+1)$ of the curve is affine,

$$A(s) = (1+i-s)a_i + (s-i)a_{i+1}, s \in [i, i+1].$$

We also represent A as a sequence of its vertices $[a_0, a_1, \cdots, a_n]$.

Definition 2.2 (Free space [4]). Free space between two polygonal curves A and B for a given distance ε is defined as

$$D_{\le\varepsilon}(A,B) = \left\{(s,s') \in [0,n]\times[0,m] \middle| d(A(s), B(s')) \le \varepsilon\right\}.$$

The free space is a two-dimensional region in the parameter space consisting of all points on the two curves with distance at most ε. A free space diagram is the free space along with vertical lines corresponding to vertices in A and horizontal lines corresponding to vertices in B, as shown in Figure 1. These vertical and horizontal lines divide the free space into *cells*. A cell $C_{ij} = [i, i+1]\times[j, j+1]$ corresponds to all possible pairing between line segment $A(i)A(i+1)$ and $B(j)B(j+1)$. The Fréchet distance between A and B is at most ε if there is a path that is monotone in both x (horizontal) and y (vertical) dimension in the free space $D_{\le\varepsilon}(A,B)$ from $(0,0)$ to (n,m). Given ε, we can decide whether there exists such a path in $O(nm)$ and we can find the Fréchet distance using parametric search in $O(nm\log(nm))$ [4].

We can define the discrete Fréchet distance on polygonal curves.

Definition 2.3 (Coupling/Traversal). Given two polygonal curves represented by a sequence of vertices $A = [a_0, a_1, \ldots, a_n], B = [b_0, b_1, \ldots, b_m]$, we define a coupling $\beta = [c_0, c_1, \ldots, c_l]$ as a sequence of pairs of points in A and B, $c_k = (c_k[0], c_k[1])$, where $c_k[0] \in A, c_k[1] \in B$, such that

- $c_0 = (a_0, b_0)$, $c_l = (a_n, b_m)$,
- if $c_k = (a_i, b_j)$, $c_{k+1} \in \{(a_i, b_{j+1}), (a_{i+1}, b_j), (a_{i+1}, b_{j+1})\}$.

The distance of a coupling is the maximum distance among the pairs c_k in β: $\max_{(a_i,b_j)\in\beta} d(a_i, b_j)$. The coupling is sometimes called traversal in the literature.

Definition 2.4 (The discrete Fréchet distance [23]). The discrete Fréchet distance between A and B is defined as the minimum of the maximum widths of all possible couplings of A, B:

$$\delta_{dF}(A,B) = \min_{\beta} \max_{(a_i,b_j)\in\beta} d(a_i, b_j).$$

It has been shown in [23] that the discrete Fréchet distance provides an upper bound for the Fréchet distance and the difference between these two distances is bounded by the longest edge length of the polygonal curves. The discrete Fréchet distance is sensitive to sampling rate of the curve.

2.2 Time-Window Fréchet Distance

In this section, we introduce the time window Fréchet distance to analyze human mobility trajectories. In the original Fréchet distance, there are no constraints on the couplings with which we obtain the Fréchet distance. This loses important information in the temporal dimension for comparing time-stamped trajectories.

A trajectory is a continuous map from time $t \in [0,1]$ to $\mathbb{R}^2$: $[0,1] \to \mathbb{R}^2$. W.l.o.g., we normalize time into the range $[0,1]$ and assume all trajectories start at time 0 and end at time 1.

Definition 2.5 (Time window Fréchet distance). Let A, B be two trajectories, and α, β be the re-parameterization of A, B. The time window Fréchet distance is defined as

$$\delta^{\sigma}_{tF}(A,B) = \inf_{\alpha,\beta} \max_{s\in[0,1]} \{d(A(\alpha(s)), B(\beta(s)))\}$$

for any $s \in [0,1]$, we have $|\alpha(s) - \beta(s)| < \sigma$, where σ is a parameter specifying which points on two trajectories can be paired.

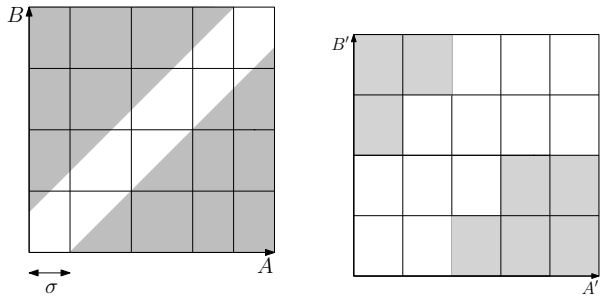
Since we add constraints on the original Fréchet distance, a valid re-parametrization for time window Fréchet distance is a valid coupling for Fréchet distance. Hence, we have the following lemma.

Lemma 2.6. (1) $\delta^{\sigma}_{tF}(A,B) \ge \delta_F(A,B)$.
(2) $\delta^{\sigma}_{tF}(A,B) \ge \delta^{\sigma'}_{tF}(A,B)$, *if* $\sigma \le \sigma'$.

Trajectories in the real world settings are often produced by discrete samples taken by various localization techniques. Without further information, we assume that the mobile entities move along the polygonal curves determined by the sample points. In this section we consider two scenarios:

(1) **Constant Speed.** A mobile entity travels at constant speed between two consecutive sample points,
(2) **Varying Speed.** A mobile entity may travel at varying speed from a sample point to the next.

Given two trajectories A, B with discrete sample points, $A(t_0) = a_0, A(t_1) = a_1, \cdots, A(t_n) = a_n, B(t'_0) = b_0, B(t'_1) = b_1, \cdots, B(t'_m) = b_m$, $a_i, b_j \in \mathbb{R}^2$, we first show the difference of the time window Fréchet distance under constant speed and varying speed conditions and then present algorithms.



(a) The constraints on free space diagram of trajectory A, B under constant speed assumption

(b) The constraints on free space diagram of polygonal curve A', B' under arbitrary speed assumption

Figure 2: Constraints on the free space diagram for time window Fréchet distance given σ.

Constant Speed. If a mobile entity travels at constant speed between two consecutive sample points, we can run interpolation to get the location of the mobile entity at time t.

In the free space diagram, we simply use time as axes. Use t to refer time used for trajectory A and t' be time of B. All pairs of t, t' within time window σ, $|t - t'| \leq \sigma$, is demonstrated in Figure 2(a), where the horizontal axis is t of A and the vertical axis is t' of B. A valid re-parametrization with time window constraints, is represented as a monotone path within the white area decided by $|t - t'| \leq \sigma$ in the free space diagram. Hence, we have the following lemma.

Lemma 2.7. *Given time window σ, $\delta^{\sigma}_{tF}(A, B) \leq \varepsilon$ iff there is a monotone path from $(0, 0)$ to $(1, 1)$ within the space decided by $|t - t'| \leq \sigma$ in the free space diagram $D_{\leq\varepsilon}(A, B)$, where t, t' are time of A and B.*

Varying Speed. If we allow a mobile entity travel at varying speed in the direction from a sample point to the next, the problem boils down to setting constraints only on how sample points of A, B are paired. The set of sample locations of B, which $A(t_i)$ can be paired with, contains those that are within time interval $[t_i - \sigma, t_i + \sigma]$, and the first one outside this interval:

$$\{B(t'_j) \big| |t_i - t'_j| \leq \sigma \text{ or } (t'_j < t_i - \sigma, t'_{j+1} \geq t_i - \sigma) \text{ or } (t'_j > t_i + \sigma, t'_{j-1} \leq t_i + \sigma)\}. \tag{1}$$

Lemma 2.8. *Given time window σ, $\delta^{\sigma}_{tF}(A, B) \leq \varepsilon$ iff there is a monotone path from $(0, 0)$ to (n, m) within the union of all valid cells in $D_{\leq\varepsilon}(A, B)$.*

Analysis can be found in Appendix A.1.

Algorithm and Running Time Analysis. We use the free space diagram to find our time-window Fréchet distance, where we enforce all monotone paths to stay within the region defined by the time window. Now, we focus on analysing the Fréchet distance under the above two assumptions. We denote by $C(n, m, \sigma)$ the number of cells containing space satisfying time window constraint σ in both above assumptions of constant and varying speeds. To find all such cells, since points on trajectories are already in chronological order, we can do a linear scan over trajectories in $O(n + m) + C(n, m, \sigma)$. Given ε, we can check whether $\delta^{\sigma}_{tF}(A, B) \leq \varepsilon$ in time $C(n, m, \sigma)$, similar to [12]. With Cole's parametric search [19] in [4], the complexity of finding the time window Fréchet distance is $O(C(n, m, \sigma) \log C(n, m, \sigma))$. Since for each sample point on the trajectories, there is at least one corresponding cell containing the space satisfying the time window constraints, $C(n, m, \sigma) \geq n$. All above, the overall complexity is $O(C(n, m, \sigma) \log C(n, m, \sigma))$.

Remark 1. *We can also apply the time-window constraint on discrete Fréchet distance and dynamic time warping, as shown in Appendix A.2.*

3 STRING REPRESENTATION AND METRIC-BASED EDIT DISTANCE

For passively collected trajectories, when a mobile entity get connected to or approximated to a labeled checkpoint, such as cellular towers, WiFi Access Points, or other stations, the appearance of the mobile entity is collected. A trajectory is represented as a sequence of checkpoints visited in order. If we associate each checkpoint by a unique character, we get a *string representation* of the mobile entity's trajectory. Such trajectories contain rich information and save tremendous storage compared with trajectories of frequent GPS sample points.

3.1 Edit Distance and Metric-Based Edit Distance

With the string representation of a trajectory, a natural metric is the edit distance. The distance between two strings $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_m$ is the smallest number of character insertions and deletions that convert A to B. One can compute the edit distance by using dynamic programming in time $O(nm)$.

For trajectories, it would make sense to differentiate the insertion of a nearby location versus the insertion of a far away location. The definition of 'nearby' or 'far-away' location can be application dependent. For example, one can examine the Euclidean distance between two locations. Alternatively, it might be interesting to define such distance by the functionality (e.g., by the district partitioning in a city, by the type of buildings which a location belongs to, etc). We assume that this distance is defined by the metric $d(\cdot, \cdot)$.

Given d, we propose a *metric-based edit distance* with insertion and deletion cost as following. Let U be the set of all characters representing locations. For $x, y, z \in U$, we define the insertion cost to insert z between x and y, as the difference of taking the detour through z rather than going straight:

$$\text{INS}(x, y, z) = d(x, z) + d(y, z) - d(x, y). \tag{2}$$

Similarly, we define the deletion cost to delete z between x and y symmetrically:

$$\text{DEL}(x, y, z) = d(x, z) + d(y, z) - d(x, y) \tag{3}$$

To define the insertion and deletion cost before the first character and after the last character of a string, we add character S and T to the beginning and end of each trajectory string. Here S, T are dummy nodes with distance $M = \infty$ to all other locations. If z stays on the line segment between x, z, $\text{INS}(x, y, z) = 0$, $\text{DEL}(x, y, z) = 0$. This makes sense as the shape of the trajectory does not change and the insertion/deletion of y properly handles possible missing

data in this case. Besides, the cost of inserting or deleting a sequnce of y between x and z has the same cost with inserting or deleting one y between x and z. In this case, redundant data is also handled.

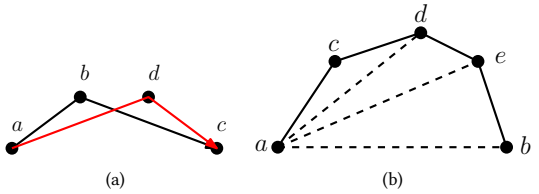


Figure 3: (a) The metric-based edit distance between trajectory abc and adc is $\min\{|ab|+2|bd|+|dc|-|ad|-|bc|, |ab|+|bc|+|ad|+|dc|-2|ac|\}$. The former is the cost of inserting d first and then deleting b, while the latter is the cost of deleting b first and then inserting d. (b) The cost of deleting cde between a and b is always $|ac|+|cd|+|de|+|eb|-|ab|$ for all the orders to delete cde.

Definition 3.1 (Metric-based edit distance). Given two trajectories with string representation, the *metric-based edit distance* is the minimum cost to convert one trajectory string to the other with arbitrary order of insertions and deletions, where the cost of insertion and deletion is defined by function INS and DEL.

In some edit distance definitions, one can substitute one character by another. Here we use insertion and deletion to simulate a substitution operation. An example of metric-based edit distance is shown in Figure 3(a).

Lemma 3.2. *Given two characters in a trajectory string, the total cost of deleting all characters between these two characters is not affected by the order of performing these deletions. Given two neighboring characters in a trajectory string, the total cost of inserting a sequence of characters between these two characters is also not affected by the order.*

Proof see Appendix B.1. An example is in Figure 3(b).

Theorem 3.3. *The metric-based edit distance is a metric.*

Proof see Appendix B.2.

3.2 Algorithm for Metric-based Edit Distance

The insertion and deletion cost of metric-based edit distance is decided by the neighbors. This provides us a cost highly related to the underlying distance metric of two symbols. On the other hand, it also increases the complexity of computing the distance as we need to consider different orders of insertion and deletion of symbols as these may affect the cost in later insertion and deletion operations. The good thing is that we can still run dynamic programming algorithm with running time $O(n^3m^3(n+m))$ with details provided in Appendix B.3.

We can simplify the computation if we require all insertions be done before any deletions when converting A to B. In this way the running time can be made to be in $O(nm)$. But the downside is that the distance no longer satisfies the triangle inequality. See Appendix B.4.

4 k-GATHER CLUSTERING

A common practice to process trajectories is to perform clustering. That is, trajectories that are similar to each other are grouped in one cluster. Tight and dense clusters of trajectories naturally correspond to meaningful features such as group motion, convoy, etc. A proper notion for this purpose is the k-gather problem, which requires each cluster to have at least k trajectories.

Definition 4.1 (k-gather problem[3]). The k-gather problem is to cluster n points in a metric space into a set of clusters, such that each cluster has one point as the center and at least k points. The objective is to minimize the maximum radius among the clusters, where the radius is the distance from a point in a cluster to the center of the cluster.

The k-gather problem on a general metric distance is NP-hard to compute when $k > 6$ [3]. We show that for our specific metric distances between trajectories the problem remains hard. Our gadgets construction was motivated by the original proof [3] and need to be carefully created to fit the trajectory setting.

Theorem 4.2. *The k-gather problem of trajectories on edit distance and metric-based edit distance is NP-hard, for $k > 13$.*

The reduction is from 3SAT. For the full proof see Appendix C.1.

We also prove the hardness of k-gather on Jaccard distance in Appendix C.2. For completeness, we provide a 2-approximation algorithm for the k-gather problem on these metrics based on the work [3] in Appendix C.3.

A New Cost Function for Hierarchical Cluster Trees

Dingkang Wang Yusu Wang

October 15, 2018

1 Introduction

Clustering has been one of the most important and popular data analysis methods in the modern data era, with numerous clustering algorithms proposed in the literature [1]. Theoretical studies on clustering have so far been focused mostly on the *flat clustering* algorithms, e.g, [2, 3, 7, 8, 9]. However, there are many scenarios where it is more desirable to perform *hierarchical clustering (HC)*, which recursively partitions data into a hierarchical collection of clusters. Hierarchical clustering can provide a more thorough view of the cluster structure behind input data across all levels of granularity simultaneously.

Most hierarchical clustering algorithms used in practice are developed in a *procedure manner*: For example, the family of *agglomerative methods* build a HC-tree bottom-up by starting with all data points in individual clusters, and then repeatedly merging them to form bigger clusters at coarser levels. The family of *divisive methods* instead partition the data in a top-down manner, starting with a single cluster, and then recursively dividing it into smaller clusters. While many of these algorithms work well in different practical situations, it is in general not clear what the output HC-tree aims to optimize. This lack of optimization understanding of the HC-tree also makes it hard to decide which hierarchical clustering algorithm one should use given a specific type of input data.

This disadvantage was recently studied by Dasgupta in [6]. Specifically, given a similarity graph G, he proposed an intuitive cost function for any HC-tree, and defined an optimal HC-tree for G to be one that minimizes this cost. Dasgupta showed that the optimal tree under this objective function has many nice properties and is indeed desirable. Furthermore, while it is NP-hard to find the optimal tree, he showed that a simple heuristic using an α_n-approximation of the sparsest graph cut will lead to an algorithm computing a HC-tree whose cost is an $O(\log^{3/2} n)$-approximation of the optimal cost. The approximation factor has since been improved to $O(\sqrt{\log n})$ in several subsequent work [4, 5, 10].

Our work. For a fixed graph G, the value of Dasgupta's cost function can be used to differentiate "better" HC-trees (with smaller cost) from "worse" ones (with larger cost), and the HC-tree with the smallest cost is optimal. However, we observe that this cost function, in its current form, does not indicate whether an input graph has a strong hierarchical structure or not, or whether one graph has "more" hierarchical structure than another graph.

We propose a new cost function to address this issue and study its properties and algorithms. In particular, by reformulating Dasgupta's cost function, we observe that for a fixed graph, there exists a *base-cost* which reflects the minimum cost one can hope to achieve. Based on this observation, we develop a new cost $\rho_G(T)$ to evaluate how well a HC-tree T represents an input graph G. An optimal HC-tree for G is the one minimizing $\rho_G(T)$. The new cost function has several interesting properties:

(i) For any graph G, a tree T minimizes $\rho_G(T)$ for a fixed graph G if and only if it minimizes Dasgupta's cost function. Furthermore, hardness results and the existing approximation algorithm developed in [4] still apply to our cost function.

(ii) For any positively weighted graph G with n vertices, the optimal cost $\rho^*_G := min_T \rho_G(T)$ is bounded with $\rho^*_G \in [1, n-2]$. The optimal cost ρ^*_G intuitively indicates how much HC-structure the graph G has.

(iii) The new formulation enables us to develop an $O(n^4 \log n)$-time algorithm to test whether an input graph G has a perfect HC-structure (i.e, $\rho^*_G = 1$) or not, as well as computing an optimal tree if $\rho^*_G = 1$. If an input graph G is what we call the δ-perturbation of a graph G^* with a perfect HC-structure, then in $O(n^3)$ time we can compute a HC-tree T whose cost is a (δ^2+1)-approximation of the optimal one.

Finally, we study the behavior of our cost function for a random graph G generated from an edge probability matrix P. Under mild conditions on P, we show that the optimal cost ρ^*_G concentrates on a certain value.

All missing proofs can be found in the full paper attached.

2 A New Cost Function for HC-trees and Properties

Our input is a set of n data points $V = \{v_1, \ldots, v_n\}$ as well as their pairwise similarity, represented as a $n \times n$ weight matrix W with $w_{ij} = W[i][j]$ representing the similarity between points v_i and v_j.

Given a set of data points $V = \{v_1, \ldots, v_n\}$, a *hierarchical clustering tree (HC-tree)* is a rooted tree $T = (V_T, E_T)$ whose leaf set equals V. We also say that T is a HC-tree *spanning* V. Given any tree node $u \in V_T$, $T[u]$ represents the subtree rooted at u, and leaves$(T[u])$ denotes the set of leaves contained in the subtree $T[u]$. Given any two points $v_i, v_j \in V$, we use $\text{LCA}_T(i,j)$ to represent the lowest common ancestor of leafs v_i and v_j in T. The following cost function to evaluate a HC-tree T w.r.t. a similarity graph G was introduced in [6]:

$$cost_G(T) = \sum_{\{i,j\} \in E} w_{ij} |\text{leaves}(T[\text{LCA}(i,j)])|.$$

An optimal HC-tree T^* is defined as one that minimizing $cost_G(T)$. Intuitively, to minimize the cost, pairs of nodes with high similarity should be merged (into a single cluster) earlier.

To introduce our new cost function, it is more convenient to take the matrix view where the weight w_{ij} is defined for all pairs of nodes v_i and v_j. First, a triplet $\{i,j,k\}$ means three distinct indices $i \neq j \neq k \in [1,n]$. We say that *relation* $\{i,j|k\}$ *holds in* T, if the lowest common ancestor $\text{LCA}(i,j)$ of v_i and v_j is a proper descendant of $\text{LCA}(i,j,k)$; *relation*$\{i|j|k\}$ *holds in* T, if they are merged at the same time.

Definition 1. *Given any triplet* $\{i,j,k\}$ *of* $[1,n]$, *the* cost of this triplet (induced by T) *is* $triC_{T,G}(i,j,k) = w_{ik} + w_{jk}$ *if relation* $\{i,j|k\}$ *holds (similarly for other two symmetric relations), and* $= w_{ij} + w_{jk} + w_{ik}$ *if relation* $\{i|j|k\}$ *holds. The* total-cost of tree T w.r.t. G *is*

$$totalC_G(T) = \sum_{i \neq j \neq k \in [1,n]} triC_T(i,j,k).$$

The following claim shows the relation between $totalC_G(T)$ and $cost_G(T)$.

Claim 1. $totalC_G(T) = \sum_{(i,j)\in E} w_{ij}(|\text{leaves}(T[\text{LCA}(i,j)])| - 2) = cost_G(T) - 2\sum_{(i,j)\in E} w_{ij}$.

Definition 2. *Given a n-node graph G associated with similarity matrix W, for any distinct triplet $\{i,j,k\} \subset [1,n]$, define its* min-triplet cost *to be*

$$minTriC_G(i,j,k) = \min\{w_{ij}+w_{ik}, w_{ij}+w_{jk}, w_{ik}+w_{jk}\}.$$

The base-cost of similarity graph G *is*

$$baseC(G) = \sum_{i\neq j\neq k\in[1,n]} minTriC_G(i,j,k).$$

To differentiate from Dasgupta's cost function, we call our new cost function the ratio-cost.

Definition 3 (Ratio-cost function)**.** *Given a similarity graph G and a HC-tree T, the* ratio-cost of T w.r.t. G *is defined as*

$$\rho_G(T) = \frac{totalC_G(T)}{baseC(G)}.$$

The optimal tree *for G is a tree T^* such that $\rho_G(T^*) = \min_T \rho_G(T)$; and its ratio-cost $\rho_G(T^*)$ is called the* optimal ratio-cost ρ^*_G.

Graphs with perfect HC-structure. Consider any triplet $\{i,j,k\}$, and assume w.l.o.g that w_{ij} is the largest among the three pairwise similarities. If there exists a "perfect" HC-tree T, then it should first merge v_i and v_j as before merging them with v_k. We say that this relation $\{i,j|k\}$ (and the tree T) is *consistent* with (similarties of) this triplet. If there is a HC-tree T consistent with relations of all triplets, then $\rho^*_G = 1$. We say this graph G has a perfect HC-structure.

While Dasgupta's cost function can be arbitrarily large (up to $O(n^3)$), it turns out that the optimal cost is always tightly bounded.

Theorem 1. *(i) Given a similarity graph $G = (V,E,w)$ with w being symmetric, having non-negative entries, we have that $\rho^*_G \in [1, n-2]$ where $n = |V| \geq 3$.*
(ii) For a connected unweighted graph $G = (V,E)$ with $n = |V|$ and $m = |E|$, we have that $\rho^*_G \in [1, \frac{n^2-2n}{2m-n}]$.

3 Algorithms

While in general, it remains open how to approximate ρ^*_G (as well as $cost_G(T^*)$) to a factor better than $\sqrt{\log n}$, we now show that we can check whether a graph has perfect HC-structure or not, and compute an optimal tree if it has, in polynomial time. We also provide a polynomial-time approximation algorithm for graphs with near-perfect HC-structures.

The high level framework of our recursive algorithm $\mathsf{BuildPerfectTree}(\widehat{G})$ is given below and output a HC-tree $\widehat{T}$ spans a subset of vertices from $\widehat{V} = V(\widehat{G})$. $\widehat{G}_A$ (resp. $\widehat{G}_B$) in the algorithm denotes the subgraph of $\widehat{G}$ spanned by vertices in $A \subseteq \widehat{V}$ (resp. in $B \subseteq \widehat{V}$). It can be proven that the output tree $\widehat{T}$ spans *all* vertices $V(\widehat{G})$, if and only if $\widehat{G}$ has a perfect HC-structure (in which case $\widehat{T}$ will also be an optimal tree).

$\mathsf{BuildPerfectTree}(\widehat{G})$ /* Input: graph $\widehat{G} = (\widehat{V}, \widehat{E})$. Output: a binary HC-tree $\widehat{T}$ */

Set (A,B)= $\mathsf{validBipartition}(\widehat{G})$; If($A = \emptyset$ or $B = \emptyset$) Return($\emptyset$)
Set T_A = $\mathsf{BuildPerfectTree}(\widehat{G}_A)$; T_B=$\mathsf{BuildPerfectTree}(\widehat{G}_B)$
Build tree $\widehat{T}$ with T_A and T_B being the two subtrees of its root. Return($\widehat{T}$)

We say that (A, B) is a *partial bi-partition of* $\widehat{V}$ if $A \cap B = \emptyset$ and $A \cup B \subseteq \widehat{V}$; and (A, B) is a *bi-partition of* $\widehat{V}$ *(or* $\widehat{G}$*)* if $A \cap B = \emptyset$, $A, B \neq \emptyset$, and $A \cup B = \widehat{V}$.

Let $\widehat{V} = \{x_1, \ldots, x_{\hat{n}}\}$ and as before, for simplicity we sometimes use an index i of a node x_i to denote this node.

Definition 4 (Triplet types)**.** *A triplet* $\{x_i, x_j, x_k\}$ *with edge weights* w_{ij}, w_{ik} *and* w_{jk} *is*
type-1: *if the largest weight, say* w_{ij}*, is strictly larger than the other two; i.e,* $w_{ij} > w_{ik}, w_{jk}$*;*
type-2: *if exact two weights, say* w_{ij} *and* w_{ik}*, are the largest; i.e,* $w_{ij} = w_{ik} > w_{jk}$*;*
type-3: *otherwise, where all three weights are equal; i.e,* $w_{ij} = w_{ik} = w_{jk}$*.*

Definition 5 (Valid partition)**.** *A partition* $(S_1, \ldots, S_m)$*,* $m > 1$*, of* $\widehat{V}$ *(i.e,* $\cup S_i = \widehat{V}$*,* $S_i \neq \emptyset$*, and* $S_i \cap S_j = \emptyset$*) is* valid w.r.t. $\widehat{G}$ *if (i) for any type-1 triplet* $\{x_i, x_j, x_k\}$ *with* $w_{ij} > \max\{w_{ik}, w_{jk}\}$*, either all three vertices belong to the same subset; or* x_i *and* x_j *are in one subset, while* x_k *is in the other; and (ii) for any type-2 triplet* $\{x_i, x_j, x_k\}$ *with* $x_{ij} = x_{ik} > x_{jk}$*, it cannot be that* x_j *and* x_k *are from the same subset, while* x_i *are in the other one.*

If this partition is a bi-partition, then it is also called a valid bi-partition*.*

The goal of procedure validBipartition$(\widehat{G})$ is to compute a valid bi-partition if it exists. This step can be finished in $O(\hat{n}^3 \log \hat{n})$. This means that each depth level during our recursive algorithm takes $O(\sum_i n_i^3 \log n_i) = O(n^3 \log n)$ time. The total complexity is $O(n^4 \log n)$ as stated in theorem 2.

Theorem 2. *Given a similarity graph* $G = (V, E)$ *with* n *vertices, we can check whether it has a perfect HC-structure, as well as compute an optimal HC-tree if it has, in* $O(n^4 \log n)$ *time.*

Graphs with almost perfect HC-structure. In practice, a graph with perfect HC-structure could be corrupted with noise. We introduce a concept of graphs with an almost-perfect HC-structure, and present a polynomial time algorithm to approximate the optimal cost.

Definition 6 (δ-perfect HC-structure)**.** *A graph* $G = (V, E, w)$ *has* δ-perfect HC-structure*,* $\delta \geq 1$*, if there exists weights* $w^* : E \to \mathbb{R}$ *such that (i) the graph* $G^* = (V, E, w^*)$ *has perfect HC-structure; and (ii) for any* $e = (u, v) \in E$*, we have* $\frac{1}{\delta} w(e) \leq w^*(e) \leq \delta \cdot w(e)$*. In this case, we also say that* $G = (V, E, w)$ *is a* δ-perturbation of graph $G^* = (V, E, w^*)$*.*

Theorem 3. *Suppose* $G = (V, E, w)$ *is a* δ*-perturbation of a graph* $G^* = (V, E, w^*)$ *with perfect HC-structure. Then we have (i)* $\rho^*_G \leq \delta^2$*; and (ii) we can compute a HC-tree* T *s.t.* $\rho_G(T) \leq (1 + \delta^2) \cdot \rho^*_G$ *(i.e, we can* $(1 + \delta^2)$*-approximate* ρ^*_G*) in* $O(n^3)$ *time.*

4 Ratio-cost function for Random Graphs

Definition 7. *Given a* $n \times n$ *symmetric matrix* P *with each entry* $\mathrm{P}_{ij} = \mathrm{P}[i][j] \in [0, 1]$*,* $G = (V = \{v_1, \ldots, v_n\}, E)$ *is a* random graph generated from P *if there is an edge* (v_i, v_j) *with probability* P_{ij}*. Each edge in* G *has unit weight.*

The expectation-graph $\overline{G} = (V, \overline{E}, w)$ *refers to the weighted graph where the edge* (i, j) *has weight* $w_{ij} = \mathrm{P}_{ij}$*.*

The main result is as follows.

Theorem 4. *Given an* $n \times n$ *edge probability matrix* P*, assume each entry* $\mathrm{P}_{ij} = \omega(\sqrt{\frac{\log n}{n}})$*, for any* $i, j \in [1, n]$*. Given a random graph* $G = (V, E)$ *sampled from* P*, let* T^* *denote the optimal HC-tree for* G *(w.r.t. ratio-cost), and* $\overline{T}^*$ *an optimal HC-tree for the expectation-graph* $\overline{G}$*. Then we have that with probability larger than* $1 - n^{-\varepsilon}$ *for some constant* $\varepsilon > 0$*,*

$$\rho^*_G = \rho_G(T^*) = (1 + o(1)) \frac{totalC_{\overline{G}}(\overline{T}^*)}{\mathbb{E}baseC(G)}.$$

References

[1] C. C. Aggarwal and C. K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[3] M. Balcan and Y. Liang. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

[4] M. Charikar and V. Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 841–854, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[5] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu. Hierarchical clustering: Objective functions and algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 378–397, 2018.

[6] S. Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 118–127, New York, NY, USA, 2016. ACM.

[7] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. Approximation schemes for clustering problems. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 50–58, New York, NY, USA, 2003. ACM.

[8] D. Ghoshdastidar and A. Dukkipati. Consistency of spectral hypergraph partitioning under planted partition model. *Ann. Statist.*, 45(1):289–315, 02 2017.

[9] K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *Ann. Statist.*, 39(4):1878–1915, 08 2011.

[10] A. Roy and S. Pokutta. Hierarchical clustering via spreading metrics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2316–2324. Curran Associates, Inc., 2016.

Far-Away Spanning Trees

Jie Gao*, Mayank Goswami†, Rebecca Schley†, Shih-yu Tsai*, and Hao-Tsung Yang*
*Stony Brook University
†City University of New York

I. Introduction

The problem of computing minimum spanning trees is a classical textbook problem. But not much is known about how to find spanning trees that have 'diversity'. Using trees that do not share a lot of edges means better reliability under potential failures or attacks. If one tree is destroyed, one can quickly switch to a different tree. In many network tasks, it is desirable to have multiple good solutions and to possibly alternate between them to avoid being tracked or attacked by adversaries [1], [2].

Given a graph G, a spanning tree is a connected subgraph of G with no cycles. If the edges are weighted, a minimum spanning tree is a spanning tree with minimum total weight. There are two problems we focus on in this paper. First, we look at k spanning trees that are far away from each other – that is, the maximum number of edges shared by any two trees is minimized. Secondly, we define k far-away α-approximate spanning trees, which are selected within the family of spanning trees with total weight at most $\alpha \cdot |\text{MST}|$ and the maximum number of edges that any two trees share is minimized. We provide algorithms in the two problems which achieve 2-approximation and $(2+\epsilon)$-approximation respectively.

II. k Far-Away Spanning Trees

Given an undirected graph $G = (V, E)$ and an integer k, let us consider the set of all the spanning trees Ω_1 of G. We are going to construct k spanning trees $T_1, \ldots, T_k \in \Omega_1$ such that

$$\min_{i \neq j} d(T_i, T_j)$$

is maximum, where d is the Hamming distance. Let d^* be the optimal such value.

$$d^* = \max_{T_1, \cdots, T_k \in \Omega_1} (\min_{i \neq j} d(T_i, T_j)).$$

We call such spanning trees which share the fewest number of edges with each other "far-away spanning trees."

A. Algorithm

We sequentially construct these k far-away spanning trees by finding the farthest spanning tree from these constructed trees, i.e. the minimum Hamming distance among all the constructed trees is maximized. First, let T_1 be an arbitrary spanning tree. Next, by implementing Algorithm 1 we find the spanning tree that is farthest from T_1. Call this tree T_2. As we continue to iterate the algorithm, we will obtain tree T_{i+1} that is farthest to $T_1, ...T_i$. By running this algorithm k times, we will get k spanning trees as the solution.

In Algorithm 1, we denote M as some large constant greater than 1. The edge-weight can be interpreted as "how close to one of the input trees." For example, if an output spanning tree has all edges with weight 1, it means the output spanning tree shares no edges with any other input trees and thus is the "farthest" tree. The intuition is to pick light edges one by one without creating a cycle until the spanning tree is constructed (lines 6 to 8). If there is a case in which the picked edge belongs to some of the input trees T_j, the weight of all other edges in T_j is increased by M such that the algorithm will be likely to pick next edges which are from other input trees (lines 9, 10). In this way, the algorithm will "fairly" pick edges among all the input trees.

Algorithm 1 Generate $i+1$-th farthest tree

1: **procedure** $i+1$-TH-FARTHEST$(V, E, \{T_1, \ldots, T_i\})$
2: Set each edge $e \in E$ with weight w_e, where

$$w_e = \begin{cases} M & \text{if } e \in T_j \text{ for some } j \\ 1 & \text{otherwise} \end{cases}$$

3: Order all edges in increasing order with respect to their weight, storing the sequence into S.
4: Denote X as the set of picked edges, started with $X = \emptyset$.
5: **repeat**
6: Pop edge f, which has the smallest weight in S.
7: **if** adding f does not create a cycle in the subgraph induced by X **then**
8: Add f into X.
9: For each input tree T_j, if $f \in T_j$, the edge-weight of all other edges in T_j is increased by M.
10: Reorder S with respect to the updated weights.
11: **end if**
12: **until** There are $n-1$ edges in X
13: Return the tree constructed by the edge set X.
14: **end procedure**

Remark. *Algorithm 1 does not generate an optimal solution.*

As a demonstration, let G be the graph in Figure 1. We wish to generate two far-away spanning trees using Algorithm 1. On the first iteration, the algorithm could generate the spanning tree consisting of black edges in Figure 1. On the second iteration, the algorithm must choose a previously used edge

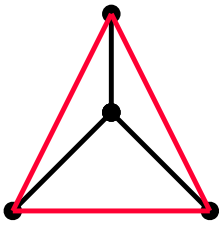


Fig. 1: Graph G with the first generated tree

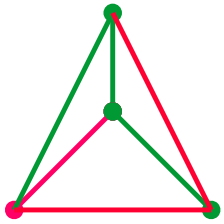


Fig. 2: Two disjoint spanning trees

to connect the center node. The Hamming distance between these two trees is 2. The optimal solution for two far-away spanning trees on this graph is represented by the green and red edges in Figure 2, which have the Hamming distance 3, which is greater than the two trees generated by the algorithm.

Though Algorithm 1 does not achieve the optimal solution, we prove that the approximation factor is 2 in the following.

B. 2-approximation

Lemma II.1. *Given a set of i spanning trees, Algorithm 1 generates the $i+1$-th spanning tree which is the farthest tree from the existing set. That is, for any spanning tree R,*

$$\min_{j=\{1,\ldots,i\}} d(R,T_j) \leq \min_{j=\{1,\ldots,i\}} d(T_{i+1},T_j)$$

Proof. To prove this lemma we have two claims. To begin, we define a "free edge" as one not already used by any of the previously generated spanning trees. A "non-free edge" is one that has already been used in a previously generated spanning tree. The first claim is that the algorithm constructs T_{i+1} by choosing as many "free" edges as possible that do not introduce a cycle. The second claim is that the algorithm chooses "non-free" edges uniform randomly.

To prove the first claim, note that Algorithm 1 will choose free edges before choosing non-free edges. Before it begins selecting any non-free edges it will have constructed a forest in G. This forest partitions the vertices of G into c components. Clearly, each component is a tree. Now, for the sake of contradiction, assume that a spanning tree R can be constructed using the same edge set as T_{i+1} except that R uses an available free edge that T_{i+1} does not use. The free edge either connects two components or is an edge inside one of the components. However, if the edge connected two components, the algorithm would have chosen the free edge because it does not introduce a cycle. If the edge is inside one of the components, then it has selected more edges in that component which makes that component as a non-tree, also a contradiction.

To prove the second claim, consider the point at which Algorithm 1 has chosen all free edges and constructed the forest of c components in G. To complete the spanning tree, Algorithm 1 must choose $c-1$ non-free edges that connect the components. Recall that once the algorithm chooses any edge in a tree, all edges in that tree are given a weight of M. In each successive iteration of the algorithm, all edges used in previously constructed trees have weight M. Since each tree is a spanning tree, each tree has an edge that connects any two components in the forest. Because all edges in previously constructed trees have the same weight, the algorithm will choose such an edge uniform randomly to connect any two of the components.

We have proven both claims. Therefore, given i spanning trees, Algorithm 1 constructs the $i+1$-th tree which is the farthest-away tree. □

Theorem II.2. *Algorithm 1 generates k spanning trees that have minimum Hamming distance d from each other that is at least $\frac{1}{2}d^*$, where d^* is the optimal solution.*

Proof. From Lemma II.1, we know that tree T_i generated at the i^{th} iteration of the algorithm is the furthest tree from $T_1,\cdots,T_{i-1}$. This gives us several useful properties. First, $\min_{j\in\{1,\cdots,i-1\}} d(T_i,T_j)$, the minimum Hamming distance between the current tree and the previously generated trees, is nondecreasing as i increases. Otherwise, tree T_j whose Hamming distance is increasing is actually further away than T_{j-1} from the set of previously generated trees $\{T_1,...T_{j-2}\}$. Second, the minimum Hamming distance between any two trees in $\{T_1,\cdots,T_k\}$ is exactly the minimum Hamming distance between T_k and $\{T_1,\cdots,T_{k-1}\}$.

Define:

$$d = \min_{j\in\{1,\cdots,k-1\}} d(T_k,T_j).$$

Consider the metric space with the metric on the space being the Hamming distance. The prior observation implies that if we draw balls of radius d centered at each of $T_1,...T_{k-1}$, then these $k-1$ balls include all of the spanning trees. Notice that there exist k spanning trees $T_1^*,\ldots,T_k^*$ which comprise the optimal set of spanning trees achieving the maximum minimum Hamming distance. There are $k-1$ balls that contain the k spanning trees $T_1^*,\ldots,T_k^*$. By the Pigeonhole Principle, two spanning trees must lie in the same ball. W.L.O.G, let them be T_1^*,T_2^*. Let the center of this considered ball be T_j. By the definition of d^* and the triangle inequality, we get

$$d^* \leq d(T_1^*,T_2^*) \leq d(T_1^*,T_j) + d(T_j,T_2^*) \leq 2d.$$

□

III. k FAR-AWAY α-SPANNING TREES

For this problem, consider an undirected weighted graph $G=(V,E)$ with edge weights c_e, an integer k, and a budget α, where $\alpha \geq 1$. We define an α-spanning tree as a tree whose edge weights sum to at most α times the weight of the minimum spanning tree of G. We consider the set of all the spanning trees Ω_α of G. The problem is to construct k α-spanning trees $T_1,...,T_k \in \Omega_\alpha$ such that

$$\min_{i,j\in\{1,2,,,k\},i\neq j} d(T_i,T_j)$$

is maximum, where d is the Hamming distance function. Let d_α^* be the optimal distance for this problem. That is,

$$d_\alpha^* = \max_{T_1,\cdots,T_k\in\Omega_\alpha} (\min_{i\neq j} d(T_i,T_j)).$$

Notice that it would not be interesting if d^*_α is pretty small, (e.g. $d^*_\alpha = 1$), since there is no much difference if we just output any k arbitrary α spanning trees. Therefore, we assume that the farthest α-spanning tree to any one α-spanning tree has distance more than $\frac{2(n-1)}{3}$.

A. Algorithm Based on Multi-Criteria Spanning Tree

We give an algorithm that returns an approximate solution to the k far-away α-spanning tree problem. Similar to Section II-A with the same notations, we design Algorithm 2 and sequentially construct these k far-away α-spanning trees by finding the "almost farthest" α-spanning tree from these constructed trees (We will define "almost farthest" later). That is, let T_1 be an arbitrary α-spanning tree and T_{i+1} is the "almost farthest" from $T_1, T_2, \cdots T_i$. As we continue to iterate Algorithm 2 k times, we will get k α-spanning trees.

Algorithm 2 Generate the $i+1$-th almost farthest α-spanning tree

1: **procedure** ITH-ALMOST-FARTHEST($V, E, \{T_1, \ldots, T_i\}$)
2: With the constructed trees $T_1, ...T_i$, let the edge set of T_j be E_j
3: **if** There is a disjoint α-spanning tree from all constructed trees $T_1, ...T_i$ **then**
4: Return the disjoint spanning tree
5: **else**
6: **for** h from 1 to $|V| - 1$ **do**
7: Run Algorithm 3 with inputs $T_1, ...T_i$, $\alpha/(1+\epsilon)$, and h.
8: **if** Algorithm 3 returns an α-spanning tree **then**
9: Return the found solution
10: **end if**
11: **end for**
12: **end if**
13: **end procedure**

Algorithm 2 firstly check if there is an α-spanning tree that is disjoint to all other input trees $T_1, \cdots, T_i$ (line 3,4). This part can be implemented by removing all edges from the input graph and run Kruskal's algorithm [3] to see the generated tree is either an α-spanning tree. If it is not the case, Algorithm 2 iteratively runs Algorithm 3 (line 6 to 11), which gives an α-spanning tree that is "almost farthest" to all input trees. Paramter h is interpreted as the maximum number of sharing edges for the output tree T_{i+1} to any input tree T_j, where $T_j \in \{T_1, \cdots, T_i\}$. To define the term "almost farthest", we are given a fixed ϵ, where $1 > \epsilon > 0$. We said that an α-spanning tree is almost farthest to trees $T_1, \ldots, T_i$ if the minimum Hamming distance from this tree to trees $T_1, \ldots, T_i$ is at least $\frac{1}{1+\epsilon}$ times the furthest possible Hamming distance. That is, a tree T is almost farthest to trees $T_1, \ldots, T_i$ if

$$\min_{j\in\{1,\ldots,i\}} d(T, T_j) \geq \frac{\max_{T'\in\Omega_\alpha}(\min_{j\in\{1,\ldots,i\}} d(T', T_j))}{1+\epsilon}.$$

Thus, we claim Algorithm 3 output a tree whose minimum Hamming distance at least $n-1-(1+\epsilon)h^*_c$ to the constructed trees, where h^*_c is the maximum number of sharing edges between the farthest α-spanning tree to any input spanning tree.

Algorithm 3 is a modified version of the PTAS for finding multi-criteria spanning trees in Chapter 11.3 [4] with the objective function being zero and the parameter settings as follows.

$$\begin{array}{lcl} c_1 & = & c_e \\ L_1 & = & \frac{\alpha}{1+\epsilon}|MST| \\ c_{2,j} & = & \text{the identity function for } E_j \\ L_{2,j} & = & h \end{array}$$

The multi-criteria spanning tree problem considered here is:

$$\begin{array}{lll} \text{minimize} & 1 & \\ \text{subject to} & \sum_{e\in E} x_e = |V| - 1, & \\ & \sum_{e\in E(S)} x_e \leq |S| - 1, & \forall S \subset V \\ & \sum_{e\in E} c_{2,j}(e) x_e \leq L_{2,j}, & \forall 1 \leq j \leq i \\ & \sum_{e\in E} c_1(e)\ x_e \leq L_1, & \forall S \subset V \\ & x_e \geq 0, & \forall e \in E \end{array} \quad (1)$$

Given the results of the above linear program, Algorithm 2 will generate the almost furthest α-spanning tree, where $|MST|$ is denoted as the weight of minimum spanning tree.

Algorithm 3 Generate a multi-criteria spanning tree

1: **procedure** MC-TREE($V, E, \{T_1, \ldots, T_i\}, \alpha/(1+\epsilon), h$)
2: Among all edges in the optimal solution, we will guess all the edges with edge weight $c_1(e) \geq \frac{\epsilon\alpha}{(1+i)(1+\epsilon)}|MST|$ or $c_{2,j}(e) \geq \frac{\epsilon}{1+i}h$ for some $j \in \{1, \cdots, i\}$.
3: Include all guessed edges in the solution and contract them. Delete all other edges from G that satisfy at least one of the aforementioned thresholds.
4: Update L_1 by removing the included edges total c_1-cost. Similarly, update $L_{2,j}$ as well.
5: **if** there is an optimal extreme point solution x to the updated linear programming 1 **then**
6: Remove every edge e with $x_e = 0$.
7: Select any spanning tree in the support graph.
8: Return the solution with the union of guessed edges and selected tree.
9: **end if**
10: **end procedure**

Theorem III.1. *Algorithm 2 runs in polynomial time with fixed values ϵ and k.*

Proof. Algorithm 2 iteratively runs Algorithm 3 for at most $n-1$ times. When ϵ and k are fixed, the number of all possible guesses is bounded in $m^{\frac{k^2}{\epsilon}}$. Thus, the total running time is $O(n \cdot m^{\frac{k^2}{\epsilon}})$. □

B. $2(1+\epsilon)$-approximation

Lemma III.2. *Given i α-spanning trees, Algorithm 1 generates the $i+1$-th α-spanning tree which is an almost farthest α-spanning tree. That is,*

$$\min_{j\in\{1,\cdots,i\}} d(T_{i+1},T_j) \geq \frac{\max_{T'\in\Omega_\alpha}\min_{j\in\{1,\cdots,i\}} d(T',T_j)}{1+\epsilon}.$$

Proof. Let the parameter h of the found $(i+1)$-th tree to be h'. Then the PTAS guarantee for the linear program 1 ensures that the tree is an α-spanning tree and that it shares at most $(1+\epsilon)h'$ edges with any previously constructed tree. Hence, its Hamming distance, $d = \min_{j\in\{1,\cdots,i\}} d(T_{i+1},T_j)$, is at least $n-1-(1+\epsilon)h'$.

On the other hand, let

$$d^* = \max_{T'\in\Omega_\alpha} \min_{j\in\{1,\cdots,i\}} d(T',T_j),$$

which means the farthest α-spanning tree from these constructed i trees has the optimal Hamming distance d^* and it has exactly $n-1-d^*$ maximum edges shared with any previous tree. When the parameter h is $n-1-d^*$, this optimal tree is a feasible solution to the linear program 1. By the PTAS guarantee, Algorithm 3 in turn is guaranteed to produce a tree when h is $n-1-d^*$. It is possible that the algorithm finds a tree earlier than the time when h is $n-1-d^*$, so the parameter h of the found tree, h', is at most $n-1-d^*$.

By the prior observations, the minimum Hamming distance of the $(i+1)$-th produced tree is at least $\frac{d^*}{1+\epsilon}$ as the following sequence of inequalities said.

$$\begin{array}{lll} d & \geq n-1-(1+\epsilon)h' & \text{by the first observation} \\ & \geq n-1-(1+\epsilon)(n-1-d^*) & \text{by the second observation} \\ & \geq \frac{d^*}{1+\epsilon} & \text{by our assumption and} \\ & & 0<\epsilon<1 \end{array}$$

□

Theorem III.3. *Algorithm 1 gives k α-spanning trees that have minimum Hamming distance d_h that is at least $\frac{1}{2(1+\epsilon)}d^*_\alpha$.*

Proof. The analysis is similar to the 2-approximation analysis for the far-away spanning tree. The only difference is the radius of the balls centered at trees $T_1,\ldots,T_{k-1}$. By Lemma III.2, setting the radius $(1+\epsilon)d_h$ will ensure these balls cover all the α-spanning trees, including the trees in the optimal set $\{T^*_1,\ldots,T^*_k\}$. If the minimum Hamming distance d_h is between T_i and $T_{i'}$ with $i<i'$, then the ball of radius $(1+\epsilon)d_h$ centered at T_{i-1} contains all the α-spanning trees excepts $T_1,\cdots,T_{i-1}$. However, these balls cover $T_1,\cdots,T_{i-1}$ as well since they are centered at $T_1,\cdots,T_{k-1}$. Hence, we will have

$$d^*_\alpha \leq d(T^*_1,T^*_2) \leq d(T^*_1,T_j)+d(T_j,T^*_2) \leq 2(1+\epsilon)d_h.$$

□

REFERENCES

[1] Clayton W Commander, Panos M Pardalos, Valeriy Ryabchenko, Stan Uryasev, and Grigoriy Zrazhevsky. The wireless network jamming problem. *Journal of Combinatorial Optimization*, 14(4):481–498, 2007.

[2] Ahmad Bilal Asghar and Stephen L Smith. Stochastic patrolling in adversarial settings. In *American Control Conference (ACC), 2016*, pages 6435–6440. IEEE, 2016.

[3] Thomas H Cormen. Section 24.3: Dijkstra's algorithm. *Introduction to algorithms*, pages 595–601, 2001.

[4] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

Computing Trajectory with Clearance for an Articulated Probe*

Ovidiu Daescu† Kyle Fox† Ka Yaw Teo†

Abstract

We consider an extension of the articulated probe trajectory planning problem introduced in [1] by requiring to compute a probe trajectory with a given clearance from obstacles or to report that no such trajectory exists. The probe is modeled as two line segments ab and bc, with a joint at the common point b, where bc is of fixed length r and ab is of arbitrarily large (infinite) length. Initially, ab and bc are collinear. Given a set of obstacles in the form of n line segments and a target point t, the probe is to first be inserted in straight line, followed possibly by a rotation of bc, so that in the final configuration c coincides with t, all while avoiding intersections with the obstacles. We prove that, for any constant $\delta > 0$, a feasible probe trajectory with a clearance δ can be determined in $O(n^2 \log n)$ time using $O(n^2)$ space.

1 Introduction

Consider the following *trajectory (or motion) planning problem*. An articulated needle-like probe is modeled in $\Re^2$ as two line segments, ab and bc, joined at point b. Line segment bc may rotate at point b. The length of line segment ab can be arbitrarily large (infinitely long), while line segment bc has a fixed length r (e.g., unit length). A two-dimensional workspace is defined as the region bounded by a circle S, which encloses a set P of n disjoint line segment obstacles (see Figure 1). Let t be a point in the free space (i.e., inside S and outside the obstacles).

In the beginning, the probe assumes a straight configuration, that is, line segments ab and bc are collinear, with $b \in ac$. We call this an *unarticulated* configuration. Starting from outside S, the unarticulated probe, represented by straight line segment abc, may be inserted into S as long as no obstacle is intersected by abc. After the insertion, line segment bc may be rotated at point b up to $\pi/2$ radians in either direction, provided that line segment bc does not collide with any obstacle. If a rotation is performed, then we have an *articulated* configuration of the probe.

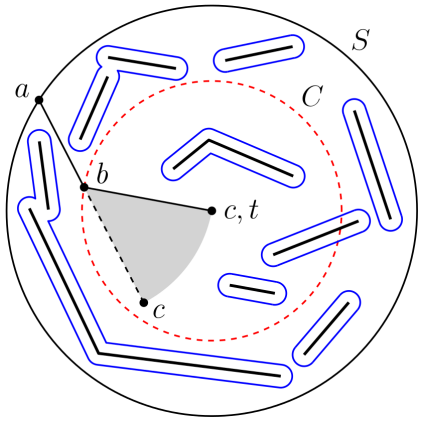


Figure 1: Planning a trajectory of a given clearance from obstacles for an articulated probe. After a straight insertion of line segment abc, in order to reach point t in the midst of obstacles, line segment bc may be required to rotate from its intermediate position (black dashed line) to the final position (black solid line). Each obstacle line segment is "dilated" by a distance $\delta > 0$ using Minkowski sum to ensure that any computed feasible probe trajectory has a clearance of at least δ from the obstacles.

A *feasible probe trajectory* consists of an initial insertion of straight line segment abc, possibly followed by a rotation of line segment bc at point b, such that point c ends at the target point t, while avoiding obstacles in the process of insertion and rotation.

Because bc may only rotate up to $\pi/2$ radians, it is an easy observation that for any feasible probe trajectory, point b is the first intersection of segment ab with a circle C of radius r centered at point t. As illustrated in Figure 1, segment bc may rotate about point b, and the area swept by segment bc is a sector of a circle (a portion of a disk enclosed by two radii and an arc) with a radius r, a center located on C, and the endpoint of one of its two bounding radii located at point t.

When no clearance from the obstacles is required, a feasible solution can be obtained using an $O(n^2 \log n)$ time, $O(n \log n)$ space algorithm as presented in [1].

In this work, we address a more general version of the problem, which asks to find a feasible probe trajectory of a given clearance δ from the obstacles for any constant $\delta > 0$ (a *δ-clearance probe trajectory* for short). A feasible probe trajectory is claimed to have a clearance

*This work is an extension of the paper [1] presented at the *30th Canadian Conference on Computational Geometry*, 2018.

†Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA. {ovidiu.daescu, kyle.fox, ka.teo}@utdallas.edu

δ from the obstacles if and only if every point of the trajectory is of at least distance δ from its nearest obstacle. We describe an algorithm that finds a δ-clearance probe trajectory in $O(n^2 \log n)$ time using $O(n^2)$ space.

2 Motivation

In the field of robotics, a simple articulated probe such as one defined herein is useful for reaching targets previously unattainable through a straight path by circumventing surrounding obstacles. Finding a collision-free probe trajectory with a certain clearance from the obstacles is of practical relevance, particularly in minimally invasive robotic surgery, where the probe is required to maintain a safe distance from any surrounding critical structures.

3 Solution approach

Our algorithm follows the general framework of the solution proposed in [1], which begins with the following observation.

Lemma 1 *There exists a feasible probe trajectory such that the probe assumes either I) an **unarticulated** final configuration (i.e., a straight line segment abc with $c = t$) that is tangent to an obstacle, or II) an **articulated** final configuration (i.e., line segments ab and bc are not colinear and $c = t$) that is tangent to an obstacle outside C and another obstacle inside or outside C.*

Based on the observation stated in Lemma 1, the set of extremal feasible probe trajectories with a given clearance δ can be obtained using the following approach.

For each obstacle line segment s of P, we define $H(s, \delta) = s \oplus B_\delta$ to be the dilation of s by a distance δ, where B_δ is a closed disk of radius δ, and $s \oplus B_\delta$ denotes the *Minkowski sum* of s and B_δ. A dilated line segment $H(s, \delta)$ has the shape of a *stadium* – a rectangle with two semi-circles attached to its sides (see Figure 1). Let $Q = \{H(s, \delta) | s \in P\}$ denote the resulting set of dilated line segments. Notice that the total number of vertices (and edges) of the dilated line segments of Q is $O(n)$.

For the purpose of analysis and clarity, the dilated line segments of Q are divided into those lying inside C and those lying outside C. Since the boundary of a dilated line segment may intersect C at most four times, a dilated line segment may be partitioned by C into at most four open (piecewise) curves, each of which may consist of line segments and circular arcs. Let Q_{in} be the set of curves lying inside C with $n_{in} = |Q_{in}|$; let Q_{out} be the set outside C with $n_{out} = |Q_{out}|$. Note that $n_{in} + n_{out} = O(n)$.

The complexity of each dilated line segment of Q is $O(1)$; that is, the tangent line from a point to a dilated line segment, as well as the common tangent lines of two dilated line segments, can be computed in $O(1)$ time. For the sake of brevity, a dilated line segment obstacle is henceforth referred to as simply an *obstacle*.

The main added difficulty in our case, when compared to [1], is that after enlarging the obstacle line segments with a disk of radius δ to account for the required clearance, we have to work with obstacles with circular arc edges. That imposes a change on the data structures needed to handle various operations, particularly for visibility computation and circular arc queries, as detailed next.

Case I. δ-clearance unarticulated probe trajectory

We compute the set R of $O(n)$ rays, each of which i) originates at point t, ii) is tangent to an obstacle of Q, and iii) does not intersect any obstacle of Q. Each ray $\gamma \in R$ represents an extremal δ-clearance unarticulated probe trajectory.

The problem of finding the set R of rays (as well as some others that follow) can be reduced to the following radial visibility problem.

Given a fixed point t and a real number r, let C be a circle of radius r centered at t. Given a set Q of n obstacles of constant complexity inside C, return the portion of C visible from t.

A simple algorithm is suggested as follows for solving the radial visibility problem. At first, find the two tangent lines from t to each obstacle in $O(n)$ time. Assume that t is located at (0, 0), and let θ be the angle of any said tangent line relative to the x-axis. Then, the pair of tangent lines to an obstacle defines a (possibly empty) occluded θ-interval $I = (\theta_s, \theta_f)$, within which circle C is invisible from t due to the obstruction by the obstacle, where $0 \leq \theta_s \leq \theta_f < 2\pi$.

The resulting $O(n)$ occluded θ-intervals, possibly overlapping, can be sorted by increasing value of θ_s in $O(n \log n)$ time, and can then be merged in $O(n)$ time to yield a set of non-overlapping occluded θ-intervals. The complement of these occluded θ-intervals is the set of visibility θ-intervals that indicates the portion of C visible from t.

Lemma 2 *Given a fixed point t and a real number r, let C be a circle of radius r centered at t. Given a set Q of n obstacles of constant complexity inside C, the portion of C visible from t can be determined in $O(n \log n)$ time.*

The set R of rays are given by the endpoints of the visibility θ-intervals. Thus, based on Lemma 2, the total time required to compute R is $O(n \log n)$.

Lemma 3 *The set of extremal δ-clearance unarticulated probe trajectories can be determined in $O(n \log n)$ time.*

Case II. δ-clearance articulated probe trajectory

For ease of exposition, the two subcases of Case II, depending on whether an articulated final configuration is tangent to 1) an obstacle outside C and an obstacle inside C, or 2) two obstacles outside C, are considered separately.

Subcase 1. In order to find a feasible probe trajectory with an articulated final configuration that is tangent to an obstacle outside C and another obstacle inside C, we first determine a feasible articulated *final* configuration in the following manner.

We compute the set R_{in} of rays, each of which i) originates at point t, ii) is tangent to an obstacle of Q_{in}, and iii) does not intersect any obstacle of Q_{in}. For each ray $\gamma_{in} \in R_{in}$, find the intersection point b of γ_{in} and C in $O(1)$ time, and compute the set R_{out} of rays, each of which i) originates at point b, ii) is tangent to an obstacle of Q_{out}, and iii) does not intersect any obstacle of Q_{out}. A pair of rays $\gamma_{in} \in R_{in}$ and $\gamma_{out} \in R_{out}$ intersecting at a point b defines a feasible final configuration that is tangent to an obstacle outside C and an obstacle inside C.

According to Lemma 2, R_{in} and R_{out} (for each ray $\gamma_{in} \in R_{in}$) can be obtained in time $O(n_{in} \log n_{in})$ and $O(n_{out} \log n_{out})$, respectively. Given that $|R_{in}|$ is bounded by $O(n_{in})$, the worst-case running time for finding a pair of rays γ_{in} and γ_{out} intersecting at a point b is $O(n_{in} \log n_{in} + n_{in} n_{out} \log n_{out})$.

After finding a probe trajectory with a feasible articulated final configuration, we examine the feasibility of its associated *intermediate* configuration (i.e., the probe configuration after inserting straight line segment abc into S and before rotating line segment bc).

For each computed point b, we consider a circle B of radius r centered at b, and find the visibility θ-intervals in $O(n_b \log n_b)$ time (see Lemma 2), where n_b is the number of obstacles lying within B. Let n_s be the number of obstacles within distance $2r$ from point t. Note that $n_b \leq n_s$ for any point $b \in C$. Recall that the size of R_{in} is bounded by $O(n_{in})$ (i.e., the upper bound on the number of distinct points b computed). Thus, the total time required to find the visibility θ-intervals for all computed points b is $O(n_{in} n_s \log n_s)$.

After finding the visibility θ-intervals for a point b, one can determine if a given radius of circle B intersects with any obstacle inside B in $O(\log n_s)$ time by using a binary search. Hence, it takes $O(\log n_s)$ time to determine if a segment bc (of an intermediate configuration) intersects with any obstacle. Since there could be $O(n_{in} n_{out})$ such segments bc, the worst-case running time for finding a feasible final configuration (that is tangent to an obstacle inside C and another obstacle outside C) with a feasible intermediate configuration is $O(n_{in} \log n_{in} + n_{in} n_{out} \log n_{out}) + O(n_{in} n_s \log n_s + n_{in} n_{out} \log n_s)$.

Subcase 2. In order to find a feasible probe trajectory with an articulated final configuration that is tangent to two obstacles outside C, we first determine a feasible intermediate configuration using the following procedure.

We compute the set R of rays, each of which i) originates from some point on circle S, ii) is a common tangent line between two obstacles of Q_{out}, iii) does not intersect any obstacle of Q_{out}, iv) intersects C, and v) goes at least distance r beyond the intersection point with C without intersecting any obstacle of Q.

R can be obtained by using the *visibility complex* of Q. The visibility complex is a two-dimensional subdivision in which each cell corresponds to a collection of rays with the same visibility properties [3]. For a simple scene of n obstacles with constant complexity, the visibility complex can be computed in $O(n \log n + k)$ time using $O(k)$ space, where k is the size of the visibility complex (or the corresponding tangent visibility graph). In the worst case, $k = O(n^2)$. After the visibility complex of Q is built, we can find the set R of rays (i.e., the set of bitangent lines that satisfy the obstacle-free restriction above) by simply traversing the cells of the visibility complex in $O(n^2)$ time.

After finding a probe trajectory with a feasible intermediate configuration, we determine if its associated final configuration is feasible.

Lemma 2 can be applied as follows in determining whether a segment bt (of a final configuration) intersects with any obstacle. The visibility θ-intervals are computed with respect to circle C centered at point t in $O(n_{in} \log n_{in})$ time. Given that $O(n_{out}^2)$ such segments bt are to be examined (using binary searches), the worst-case running time for finding a feasible intermediate configuration (that is tangent to two obstacles outside C) with a feasible final configuration is $O(n^2) + O(n_{in} \log n_{in} + n_{out}^2 \log n_{in})$.

An articulated probe trajectory with both a feasible final configuration and a feasible intermediate configuration is feasible if and only if the area swept by segment bc after the initial insertion (i.e., a circular sector) is not intersected by any obstacle. Thus, the remainder of Case II entails a circular sector intersection problem, detailed in the next section.

4 Circular sector intersection queries

The general circular sector intersection query problem can be formally stated as follows.

Given a set Q of n obstacles, preprocess it so that, for a query circular sector σ, one can efficiently determine whether σ intersects Q.

For our purposes, it suffices to solve a special case of this problem where the radius of the circular sector is fixed to r and one endpoint of the circular arc of the sector is fixed at t.

Recall that a pair of feasible final and intermediate configurations for an articulated probe trajectory have been found in the previous section. Thus, both radii of the query circular sector are certainly not intersected by any obstacle of Q. Therefore, an obstacle can only intersect with a circular sector by i) intersecting the sector's arc, or ii) lying completely inside the sector. Hence, our case of the circular sector intersection problem reduces to the following two problems – i) circular arc intersection query, and ii) circular sector emptiness query.

Circular arc intersection queries. Consider the following circular arc intersection problem.

Problem 1 *Given a set Q of n line segments and semi-circular arcs, preprocess it so that, for a query circular arc γ that originates at a fixed point t and has a fixed radius r, one can efficiently determine if γ intersects Q.*

Problem 1 can be solved by using a similar data structure (i.e., lower envelopes) as constructed in [1]. In brief, for any $\theta \in [0, 2\pi)$, a unique circle D of radius r centered at some point $p \in C$ can be defined such that the angle of tp relative to the x-axis is θ. Note that circle D always passes through t. A lower envelope can then be computed to represent the length of the circular arc of D from t to the first intersection with a line segment (or a semi-circular arc) of Q. Given that two semi-circular arcs (or an arc and a line segment, or two line segments) of Q can intersect at most two times, the size of the lower envelope is bounded by the fourth-order Davenport-Schinzel sequence, which is $O(n \cdot 2^{\alpha(n)})$, where $\alpha(n)$ is the inverse of the Ackermann function. The lower envelope can be computed in $O(n\alpha(n) \log n)$ time [2, 4]. A binary search can be performed on the lower envelope to determine if a query circular arc γ intersects Q. Thus, the following result is obtained.

Lemma 4 *A set Q of n line segments and circular arcs can be preprocessed in $O(n\alpha(n) \log n)$ time into a data structure of size $O(n \cdot 2^{\alpha(n)})$ so that, for a query circular arc γ that originates at a fixed point t and has a fixed radius r, one can determine whether γ intersects Q in $O(\log n)$ time.*

Circular sector emptiness queries. Our special case of the circular sector emptiness problem can be stated as follows.

Problem 2 *Given a set Q of n points in the plane, preprocess it so that, for a query circular sector σ of fixed radius r whose arc has an endpoint at a fixed point t, one can efficiently determine whether σ contains any point of Q.*

Problem 2 can be solved as previously described in [1], and the result is summarized in the following lemma.

Lemma 5 *A set Q of n points in the plane can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n \log n)$ so that, given a query circular sector σ with radius r and an endpoint of its arc located at t, one can determine whether σ contains any point of Q in $O(\log n)$ time.*

Recall that n_s is the number of obstacles within distance $2r$ from point t. In Case II, given that $O(n_{in}n_{out} + n_{out}^2)$ queries are to be processed in the worst case and we only need to worry about obstacles lying sufficiently close to t, the following result is obtained.

Lemma 6 *A δ-clearance articulated probe trajectory can be determined in time $O(n_{in} \log n_{in} + n_{in}n_{out} \log n_{out}) + O(n_{in}n_s \log n_s + n_{in}n_{out} \log n_s) + O(n^2) + O(n_{in} \log n_{in} + n_{out}^2 \log n_{in}) + O((n_{in}n_{out} + n_{out}^2) \log n_s)$ using $O(n^2 + n_s \log n_s)$ space.*

Theorem 7 *A δ-clearance probe trajectory can be determined in $O(n^2 \log n)$ time using $O(n^2)$ space.*

5 Conclusion

We presented an efficient algorithm for computing a δ-clearance probe trajectory in $O(n^2 \log n)$ time. Our algorithm can be easily extended to the case of polygonal obstacles, where we can exploit output sensitive algorithms with respect to the number of polygons. Our algorithm has the same time complexity as when no clearance is required; however, the space complexity increases from $O(n \log n)$ to $O(n^2)$. We believe that it is possible to reduce the space usage to $O(n \log n)$ by using an incremental construction of the visibility complex. Finally, an open problem remains as to how the space of the feasible solutions can be characterized.

References

[1] O. Daescu, K. Fox, and K. Teo. Trajectory planning for an articulated probe. In *30th Canadian Conference on Computational Geometry*, pages 296–303, 2018.

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

[3] M. Pocchiola and G. Vegter. The visibility complex. *International Journal of Computational Geometry & Applications*, 6(3):279–308, 1996.

[4] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, 1995.

Computing Simple Polygonizations of Disjoint Line Segments is NP-Complete *

Hugo A. Akitaya[†] Matias Korman[†] Mikhail Rudoy[‡] Diane L. Souvaine[†]
Csaba D. Tóth[§†]

1 Introduction

Simple polygons are foundational for Computational Geometry. A natural problem is to find **simple polygonizations of point sets**, i.e., given a point set S in the plane, a simple polygonizations of S is a simple polygon P whose vertex set is S. It is easy to see that, unless all points are collinear, every point set S has a simple polygonization. There are several results on the upper and lower bounds on the number of simple polygonizations of point sets (see [1] and [10] for the currently best known bounds, and [3] for a survey on this and related problems).

A natural generalization of this problem is finding simple polygonizations of line segments: given a set S of line segments in the plane, find a simple polygon whose vertex set is the set of endpoints of segments in S, and whose edge set contains S. The problem can also be phrased as a graph augmentation problem: Can a geometric graph be augmented to a simple Hamiltonian cycle? Deciding whether such simple polygon exists is NP-complete [8] in the general case. However the hardness proof provided requires that segments in the input share endpoints. Rappaport [8] conjectured in 1989 that the problem remains hard even if the input consists of disjoint line segments, but the problem remained open since then. In the special case that every segment has at least one endpoint on the boundary of the convex hull, an $O(n \log n)$ time algorithm is available [9]. In this paper, we confirm the conjecture and show that deciding whether a given set of disjoint line segments admits a simple polygonizations is NP-complete.

Further Related Previous Work. Many different variations of the polygonization problem have been considered in the literature. Hoffmann and Tóth [2] proved that the vertex visibility graph of a set S of n disjoint segments in the plane is Hamiltonian (as long as not all segments lie on a line). This implies a slightly weaker result: there exists a simple polygon with $2n$ vertices in which every segment in S is an edge, an internal diagonal, or an external diagonal. Ishaque et al. [4] proved that n disjoint segments can be augmented to a 2-regular planar straight-line graph if n is even. A **circumscribing polygon**, for a set S of n disjoint segments in the plane, is a simple polygon with $2n$ vertices in which every segment in S is either an edge or an internal

*Research supported in part by the NSF awards CCF-1422311 and CCF-1423615. MK was partially supported by MEXT KAKENHI No. 17K12635.

[†]Department of Computer Science, Tufts University, Medford, MA, USA.
[‡]MIT CSAIL, 32 Vassar St., Cambridge, MA 02139, USA. Now at Google.
[§]Department of Mathematics, California State University Northridge, Los Angeles, CA, USA.

diagonal. Urabe and Watanabe [11] constructed a set of 16 disjoint segments that does not admit a circumscribing polygon. A circumscribing polygon is guaranteed to exist when (i) every segment has at least one endpoint on the boundary of the convex hull [5], or (ii) no segment intersects the supporting line of any other segment [6]. It is also known that every set of n disjoint segments has a subset of $\Omega(n^{1/3})$ segments that admits a circumscribing polygon [7]. However, no nontrivial upper bound is known.

2 Hardness for Disjoint Segments

The reduction in [8] is from Hamiltonian Path in Planar Cubic Graphs (HPPCG) and produces an instance containing axis-aligned line segments. Each segment shares at least one endpoint with another segment and endpoints are located in the integer grid. In practice, a polygonal chain formed by such segments must be contained in any simple polygonization of the input.

Theorem 1. *It is NP-complete to decide whether a set S of disjoint line segments admits a simple polygonization, even if S contains only segments with 4 orientations.*

Proof. We reduce from finding simple polygonizations of line segments. Our input is an instance produced by the reduction in [8]. We label an endpoint in this instance **useful** if it is the endpoint of a single segment. We prove that we can incrementally replace chains of two segments by disjoint line segments while preserving the existence or non-existence of a solution. Our strategy is to replace a pair of segments p_1p_2 and p_2p_3 that share an endpoint p_2 by seven disjoint segments p_1p_2', p_4p_5, p_6p_7, p_8p_9, $p_{10}p_{11}$, $p_{12}p_{13}$, and p_2p_3. Refer to Figure 1. We call this construction a **connection gadget**. We must prove two properties: (i) any simple polygonization of the modified input will contain a chain using only the seven segments in the gadget and connecting p_1 to p_3; (ii) if p_1 (resp., p_3) is labeled useful, the set of useful endpoints visible from p_1 (resp., p_3) is the same before and after the replacement.

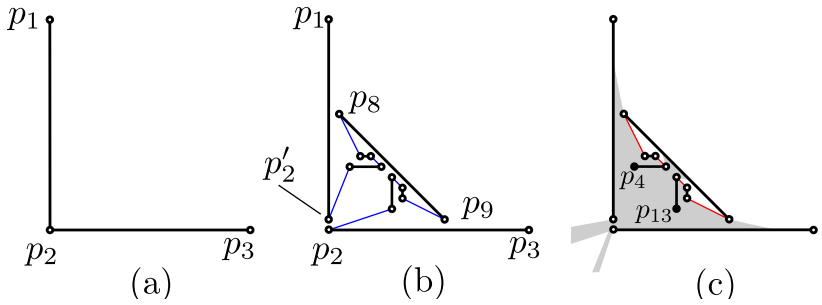


Figure 1: (a) Two line segments p_1p_2 and p_2p_3. (b) Connection gadget that simulates (a) using seven disjoint line segments. The polygonal path shown with black and blue line segments is $[p_1, p_2', p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_2, p_3]$. (c) The union of the visibility region of the solid black points p_4 and p_{13}.

We first give an overview and then give the details of the proof. The main property of the connection gadget is that the points p_6 and p_7 (resp., p_{10} and p_{11}) can only see p_4, p_5, and p_8 (p_9, p_{12}, and p_{13}). Then any solution must connect p_8 with p_6 or p_7, and p_6 or p_7 to p_4 or p_5, or else there would be a cycle of length 4 disconnected from the rest of the solution. The same argument

applies symmetrically to $p_9, \ldots, p_{13}$. Fig. 1(c) shows the forced connections in red. We choose the position of the segments so that points p_4, p_5, p_{12}, and p_{13} can only see points p_6, p_{11}, p_2, and p_2'. Then p_2' must connect to p_4 or p_5, and p_2 must connect to p_{12} or p_{13}, or else there would be a cycle of length 10 disconnected from the rest of the solution. Then, property (i) holds. Property (ii) also holds if we place points $p_4, \ldots, p_{13}$ close to p_2 so that the visibility of p_1 and p_3 are not affected.

We now show where to place the segments in the connection gadget. Recall that our input contains endpoints on integer coordinates. We assume that the bounding box of the construction is in the first quadrant and contains the origin. We describe the gadget for the orientation shown in Figure 1. Other orientations can be obtained by reflections. We first describe the position of auxiliary lines. Refer to Fig. 2(a). The dashed line ℓ_1 connects $p_2 + (0, \frac{1}{4})$ and $p_2 + (\frac{1}{4}, 0)$, and will contain p_9 and p_8. Let ℓ_2 (resp., ℓ_3) be the line passing through $(0, y(p_2) - \frac{3}{4})$ (resp., $(0, y(p_2) - \frac{1}{2})$) and p_2. Let a (resp., b) be the intersection point between ℓ_2 (resp., ℓ_3) and ℓ_1, and ℓ_4 be the line through $(0, y(p_2) - \frac{1}{4})$ and b or the line through $p_2 + (-\frac{1}{4}, 0)$ and b, whichever has larger slope. Let t be the intersection of ℓ_4 and ℓ_2. We place p_2' at the intersection between ℓ_4 and $x = x(p_2)$. We will place points $p_4, \ldots, p_7$ and $p_{10}, \ldots, p_{13}$ in the triangle defined by ℓ_1, ℓ_2 and ℓ_4: Δabt. This guarantees that these points will not be able to see any endpoint not in the gadget.

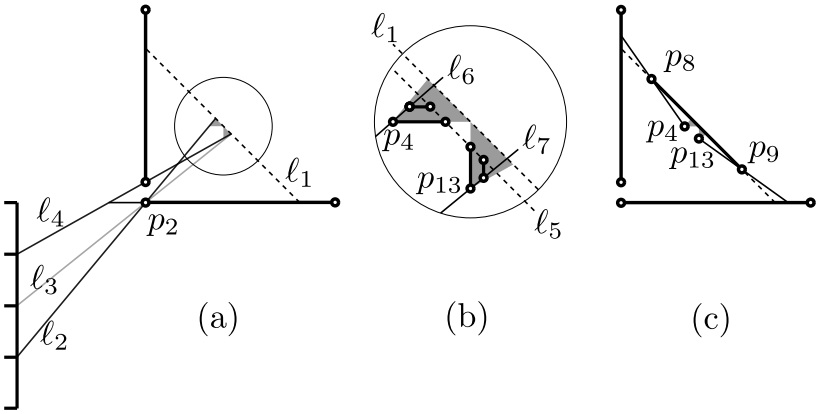


Figure 2: The figures are not drawn to scale. (a) The construction of the dashed line ℓ_1 and the lines ℓ_2, ℓ_3, and ℓ_4; (b)The placement of the segments $p_4, \ldots, p_7$, and $p_{10}, \ldots, p_{13}$; (c)The placement of the line segment p_8, p_9.

Refer to Fig. 2(b). Let c be the midpoint of the segment ab. Place p_4 (resp., p_{13}) on the intersection of ℓ_2 and $y = y(c)$ (resp., ℓ_4 and $x = x(c)$). Let d be the minimum between the distances between ℓ_1 and p_4, and ℓ_1 and p_{13}. We define another line ℓ_5 being parallel to and below ℓ_1, and the distance between ℓ_1 and ℓ_5 is $\frac{d}{2}$. Place p_5 (resp., p_{12}) at the intersection between ℓ_5 and $y = y(c)$ (resp., $x = x(c)$). Let ℓ_6 (resp., ℓ_7) be the line through p_2' and p_4 (resp., p_2 and p_{13}). Place the segment p_6p_7 (resp., $p_{10}p_{11}$) parallel to p_4p_5 (resp., $p_{12}p_{13}$) and inside the triangle defined by ℓ_5, $y = y(c)$, and ℓ_6 (resp., ℓ_5, $x = x(c)$, and ℓ_7). The construction guarantees that p_2 and p_2' cannot see segments p_6p_7 or $p_{10}p_{11}$. Refer to Fig. 2(c). Place p_8 (resp., p_9) at the intersection of ℓ_1 and the line through p_4 and $p_2 + (0, \frac{1}{2})$ (resp., the line through p_{13} and $p_2 + (\frac{1}{2}, 0)$). The positioning of p_8p_9 blocks the visibility of $p_4, \ldots, p_7$ and $p_{10}, \ldots, p_{13}$ so that they can't see any point above ℓ_1. This concludes the construction for one connection gadget.

After replacing every shared endpoint by a connection gadget, the instance contains disjoint line segments of four directions. The construction satisfies properties (i) and (ii), and the correctness

of the proof is implied by these properties. All coordinates of new points are solutions of linear equations, and thus the coordinates of endpoints can be described by polynomials. The membership in NP is proven in [8]. □

3 Open Problems

- In Section 2 we established NP-hardness, even when the input consists of segments with four directions. Does the hardness still hold if segments have only three (or even fewer) distinct directions?
- Does every set of disjoint axis-parallel segments admit a circumscribing polygon?
- Let $f(n)$ be the maximum integer such that every set of n disjoint segments contains $f(n)$ segments that admit a circumscribing polygon. It is known that $f(n) < n$ [11]. Is it possible that $f(n) = \Omega(n)$? Is there a nontrivial upper bound?
- Let $g(n)$ be the maximum integer such that every set of n disjoint segments contains $g(n)$ segments that can be extended to disjoint rays (i.e., half-lines). It is not difficult to show that $g(n) \leq f(n)$. Pach and Rivera-Campo [7] proved that $g(n) = \Omega(n^{1/3})$, and there is an easy construction that yields $g(n) = O(\sqrt{n})$. What is the asymptotic growth rate of $g(n)$?

References

[1] Alfredo García, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of K_N. *Comput. Geom.*, 16(4):211–221, 2000.

[2] Michael Hoffmann and Csaba D. Tóth. Segment endpoint visibility graphs are Hamiltonian. *Comput. Geom.*, 26(1):47–68, 2003.

[3] Ferran Hurtado and Csaba D. Tóth. Plane geometric graph augmentation: A generic perspective. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 327–354. Springer, New York, 2013.

[4] Mashhood Ishaque, Diane L. Souvaine, and Csaba D. Tóth. Disjoint compatible geometric matchings. *Discrete & Computational Geometry*, 49(1):89–131, 2013.

[5] Andranik Mirzaian. Hamiltonian triangulations and circumscribing polygons of disjoint line segments. *Comput. Geom.*, 2:15–30, 1992.

[6] Joseph O'Rourke and Jennifer Rippel. Two segment classes with Hamiltonian visibility graphs. *Comput. Geom.*, 4:209–218, 1994.

[7] János Pach and Eduardo Rivera-Campo. On circumscribing polygons for line segments. *Comput. Geom.*, 10(2):121–124, 1998.

[8] David Rappaport. Computing simple circuits from a set of line segments is NP-complete. *SIAM Journal on Computing*, 18(6):1128–1139, 1989.

[9] David Rappaport, Hiroshi Imai, and Godfried T. Toussaint. Computing simple circuits from a set of line segments. *Discrete & Computational Geometry*, 5(3):289–304, 1990.

[10] Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *J. Comb. Theory, Ser. A*, 120(4):777–794, 2013.

[11] Masatsugu Urabe and Mamoru Watanabe. On a counterexample to a conjecture of Mirzaian. *Comput. Geom.*, 2:51–53, 1992.

Challenges in Reconstructing Shapes from Euler Characteristic Curves

Brittany Terese Fasy*† Samuel Micka† David L. Millman† Anna Schenfisch*
Lucia Williams†

October 18, 2018

Abstract

Shape recognition and classification is a problem with a wide variety of applications. Several recent works have demonstrated that topological descriptors can be used as summaries of shapes and utilized to compute distances. In this abstract, we explore the use of a finite number of Euler Characteristic Curves (ECC) to reconstruct plane graphs. We highlight difficulties that occur when attempting to adopt approaches for reconstruction with persistence diagrams to reconstruction with ECCs. Furthermore, we highlight specific arrangements of vertices that create problems for reconstruction and present several observations about how they affect the ECC-based reconstruction. Finally, we show that plane graphs without degree two vertices can be reconstructed using a finite number of ECCs.

1 Introduction

Shape comparison and classification is a common task in the field of computer science, with applications in graphics, geometry, machine learning, and several other research fields. The problem has been well-studied in $\mathbb{R}^3$, with several approaches described in the survey [6]. One relatively new approach to the problem involves utilizing topological descriptors to represent and compare the shapes. In [7], Turner et al. proposed the use of the zero- and one-dimensional persistence diagrams from lower-star filtrations to compare triangulations of S^k in $\mathbb{R}^d$, for $d > k$. We call the mapping of a shape to to a parameterized set of diagrams the *persistent homology transform* (PHT). Their main result (Cor. 3.4 of [7]) showed that the persistent homology transform (PHT) is injective for comparing triangulations of $\mathbb{S}^2$ or $\mathbb{S}^1$ embedded in $\mathbb{R}^3$ (or triangulations of $\mathbb{S}^1$ in $\mathbb{R}^2$), and thus can be used to distinguish different shapes. Turner et al. also extend the idea of the PHT to the Euler Characteristic Curve (ECC) and describe the Euler Characteristic Transform (ECT), a topological summary that records changes in the Euler Characteristic across a height parameter, again from all directions. Finally, using experimental results, the authors show that the PHT and ECT performed well in clustering tasks. In [2], Crawford et al. extend this work by proposing the smooth Euler Characteristic Transform (SECT), a functional variant of the ECT with favorable properties for analysis. They show that features derived from the SECT of tumor shapes are better predictors of clinical outcomes of patients than other traditional features.

The proof of injectivity (i.e., that a shape can be reconstructed from the PHT or the ECC) uses an infinite set of a directions; however, using an infinite set of directions is infeasible for computational purposes. Thus, both [2, 7] use sampling a finite set of directions for the height filtrations in order to apply the technique to shape comparison. In [1], Belton et al. present an algorithm for reconstructing plane graphs using a quadratic (hence, finite) number of persistence diagrams. Simultaneous to that result, other researchers also attempted to give a finite number of directions sufficient to fully determine a shape. Both [3] and [5] give upper bounds on the number of directions needed to determine a hidden shape in $\mathbb{R}^d$. In order to do this, they make assumptions about the curvature and geometry of the input shape. In our work, by contrast, we restrict to plane graphs, but make no restrictions on curvature.

Here, we attempt to extend the work of [1] on the PHT to the ECT. However, difficulties arise when using ECCs because they do not encode information about every vertex from every direction, as a persistence diagram does when on-diagonal points are in-

*Depart. of Mathematical Sciences, Montana State U.
†School of Computing, Montana State U.
{brittany.fasy, david.millman, annaschenfisch}@montana.edu
{samuel.micka, lucia.williams}@msu.montana.edu

cluded. We show that, while the number of directions needed to give an ECT unique to the input graph is linear in the number of vertices of the graph, it is difficult to determine which directions generate the necessary ECCs. As we will see, the main difficulty lies with the presence of degree two vertices.

2 Background

In this paper, we focus on a subset of finite simplicial complexes that are composed of only edges and vertices and are provided with a planar straight-line embedding in $\mathbb{R}^2$. We refer to these simplicial complexes as *plane graphs*. We refer the reader to [4] for a general background on persistent homology, and only present the necessary content here.

Assumptions Let K be a plane graph. In what follows, we assume that the vertices of K have distinct x- and y-coordinates from one another. Furthermore, we assume that no three vertices are collinear.

Lower-Star Filtration Let $\mathbb{S}^1$ be the unit sphere in $\mathbb{R}^2$. Consider $s \in \mathbb{S}^1$, i.e., a direction vector in $\mathbb{R}^2$; we define the *lower-star filtration* with respect to s. Let $h_s : K \to \mathbb{R}$ be defined for a simplex $\sigma \subseteq K$ by $h_s(\sigma) = \max_{v \in \sigma} v \cdot s$, where $x \cdot y$ is the inner (dot) product and measures height in the direction of unit vector y. Intuitively, the height of σ with respect to s is the maximum "height" of all vertices in σ. Then, for each $h \in \mathbb{R}$, the subcomplex $K_h := h_s^{-1}((-\infty, h])$ is composed of all simplices that lie entirely below or at the height h, with respect to the direction s. The lower-star filtration is sequence of subcomplexes K_h, where h increases from $-\infty$ to ∞; notice that K_h only changes when h is the height of a vertex of K.

When we observe a difference between $K_{h-\epsilon}$ and $K_{h+\epsilon}$, we know that we have encountered a vertex. As in [1], we define a structure to encode what we know about this vertex in $\mathbb{R}^2$. Given $s \in \mathbb{S}^1$, and a height $h \in \mathbb{R}$, the *filtration line at height* h is the line, denoted $\ell(s, h)$, perpendicular to direction s and at height h in direction s. Given a finite set of vertices $V \subset \mathbb{R}^2$, the *filtration lines of* V are the set of lines

$$\mathbb{L}(s, V) = \{\ell(s, h) \mid \exists v \in V \text{ s.t. } h = v \cdot s\}.$$

Further, $\mathbb{L}(s, V)$ will contain $|V|$ lines if and only if no two vertices have the same height in direction s. Our assumptions guarantee distinct vertex heights only for $(0, 1)$, $(0, -1)$, $(1, 0)$, and $(-1, 0)$, referred to as the *cardinal directions*. In [1], every line in $\mathbb{L}(s, V)$ can be read off of the persistence diagram, as every simplex corresponds to either a birth or the death of a homology class. Next, we observe that we cannot witness all such lines for another topological descriptor, the Euler Characteristic Curve.

Euler Characteristic Curves The Euler characteristic of a plane graph $K = (V, E)$ is $|V| - |E|$. The *Euler Characteristic Curve* (ECC) is the piecewise step function of the Euler characteristic, whose domain is subcomplexes of a filtration defined by some parameterization of K. In this paper, the parameter is the *height* of a lower-star filtration. Specifically, we define $\chi_s^K : \mathbb{R} \to \mathbb{Z}$ to be the function that maps a height h to the Euler Characteristic of K_h. Every change in the ECC corresponds to a filtration line from that direction, but not vice versa. For example, if an edge and vertex appear at the same height, then the ECC does not change. We now refine our definition of filtration lines:

$$\mathbb{W}(s, V) = \{\ell(s, h) \mid \exists \epsilon_0 > 0 \text{ s.t. } \forall \epsilon \in (0, \epsilon_0), \chi_s^K(h - \epsilon) \neq \chi_s^K(h + \epsilon)\}.$$

This set corresponds to the subset of vertices in V that are *witnessed* from s through the ECC χ_s^K. As such, we refer to these lines as *witnessed lines*. We note that the only time that a vertex is not witnessed is if the vertex is included in the filtration at the same time as an edge because the vertex being added will be cancelled out by the inclusion of the edge. Furthermore, we note that v lying on a filtration line from s does not necessarily imply that v is witnessed from s, i.e., it could lie on a witness line for another vertex if they lie at the same height from s.

3 Towards Vertex Reconstruction

We are interested in reconstructing a plane graph from ECCs from a finite number of directions. While three directions was sufficient for reconstructing vertices using persistence diagrams, ECCs contain strictly less information in each direction. We observe the existence of a linear number of directions that allows to fully reconstruct the vertices of a plane graph:

Proposition 1 (ECC Existence)**.** *Given a plane graph $K = (V, E)$ with $|V| = n$, there exist $3n$ directions that can be used to reconstruct all vertices in V.*

The proof of this claim may be found in Appendix A. We note that while $3n$ directions are sufficient, this bound is likely not tight.

Initially, attempting to use the techniques in [1] seems promising for plane graph reconstruction using ECCs, i.e., we can define a correspondence between

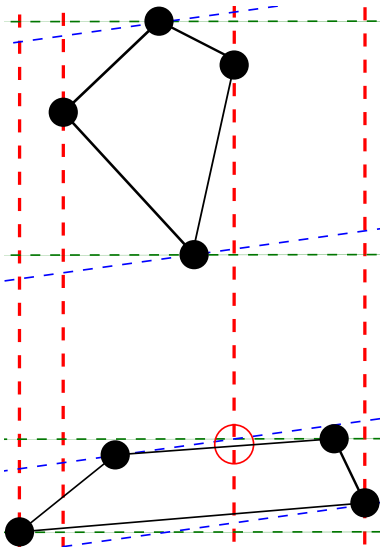


Figure 1: Scenario where a degree two vertex not witnessed by the cardinal directions can create a three-way line intersection where a vertex does not exist. The three-way intersection without a vertex is circled in red.

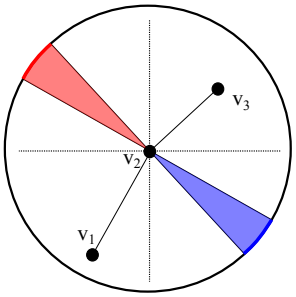


Figure 2: Case where v_1, v_2, and v_3 are nearly collinear. As the vertices approach collinear the region on $\mathbb{S}^1$ containing directions which will witness v_2 grows arbitrarily small.

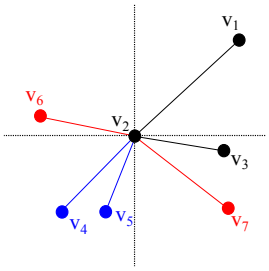


Figure 3: Different scenarios of edge embeddings for degree two vertices. We consider v_2 a degree two vertex when considering, exclusively, the sets of edges $\{(v_1, v_2), (v_3, v_2)\}$, $\{(v_4, v_2), (v_5, v_2)\}$, or $\{(v_6, v_2), (v_7, v_2)\}$. These three sets of edges highlight situations in which v_2 can be witnessed in different ways.

three-way witness line intersections (from carefully chosen directions) and vertices. However, certain types of vertices introduce difficulties. For example, consider Figure 1. A degree two vertex is not witnessed by any of the witness lines from the cardinal directions $(1, 0), (0, 1), (-1, 0)$ and $(0, -1)$. However, we would like to generate a correspondence between three-way intersections of witness lines and non degree two vertices. If we use the technique described in Theorem 5 of [1] to choose such a direction, that direction creates a witness line that causes a three-way intersection not corresponding to a vertex. In fact, when degree two vertices are introduced to the plane graph, several problems arise. We discuss these problems in detail in Section 4.

4 Degree Two Challenges

Degree two vertices introduce several complications in finding witness directions, because degree two vertices can have an arbitrarily small region on $\mathbb{S}^1$ from which they can be witnessed. For example, in Figure 2 the vertices v_1, v_2, and v_3 are nearly collinear. In order to witness v_2, we must choose directions from within the red region, where a decrease in the ECC will be observed, or from the blue region, where an increase in the ECC will be observed. However, these these regions becomes arbitrarily small as v_1, v_2 and v_3 approach collinear.

As mentioned earlier, degree two vertices can also introduce additional ambiguities when witnessing non-degree two vertices. Recall the example found in Figure 1 and the discussion in Section 3.

Despite these difficulties, several situations exist in which degree two vertices can be witnessed. The following propositions summarize these scenarios. Proofs are provided in Appendix A. For clarity, we discuss quadrants as though v_2 is located at the origin. However, note that the following propositions also apply to arrangements with similar orientations and angles.

Proposition 2 (Same Quadrant)**.** *If v_1 and v_3 lie in the same quadrant, such as the vertices v_4 and v_5 in Figure 3, then v_2 will be witnessed in ECCs from every one of the cardinal directions.*

Proposition 3 (Neighboring Quadrants)**.** *If v_1 and v_3 lie in neighboring quadrants, such as vertices v_1 and v_3 in Figure 3, then v_2 will be witnessed in ECCs from exactly two of the four cardinal directions.*

Proposition 4 (Degree Two Bounded Angle)**.** *If* $angle((v_1, v_2), (v_2, v_3)) < \frac{\pi}{2}$ *then* v_2 *will be witnessed in ECCs from at least two of the four cardinal directions.*

The above propositions show scenarios for which degree two vertices can be witnessed using cardinal directions. However, degree two vertices pose particular problems when the edges lie in non-neighboring quadrants, such as the edges (v_6, v_2) and (v_7, v_2) in Figure 3 or (v_1, v_2) and (v_2, v_3) in Figure 2. Then, when degree two vertices are not included in a plane graph K, a constant number of ECCs can be used to determine the embeddings of the vertices.

5 A Special Case

If a plane graph contains no degree two vertices, the graph can be reconstructed using a finite number of ECCs. Let K denote a plane graph with vertex and edge sets V and E respectively. Recall from [1] that three way filtration line intersections from carefully chosen directions correspond to a vertex location for plane graphs using persistence diagrams. We show that this result still holds for reconstructing plane graphs using ECCs, if they do not contain degree two vertices. The proofs of the following lemmas and theorem can be found in Appendix A.

First, we provide a lemma that yields insight into how non-degree two vertices are witnessed.

Lemma 1 (Linear Witness Lines)**.** *Let* K *be a plane graph in* $\mathbb{R}^2$ *with vertices* V *such that for all* $v \in V$*,* $deg(v) \neq 2$ *and denote* $|V| = n$*. Let* ℓ *be a line in* $\mathbb{R}^2$ *such that any line parallel to* ℓ *intersects at most one vertex in* V*. Let* $s \in \mathbb{S}^1$ *be chosen perpindicular to* ℓ*. Then,*

$$|\mathbb{W}(s, V) \cup \mathbb{W}(-s, V)| = n$$

By generalizing the results of Lemma 1, we introduce the the following Lemma to generate n^2 potential vertex locations in $\mathbb{R}^2$, where n is the number of vertices.

Lemma 2 (Witness Line Intersections)**.** *Recall the cardinal directions* $(0,1), (1,0), (0,-1), (-1,0) \in \mathbb{S}^1$*. If for all* $v \in V$*,* $deg(v) \neq 2$ *then*

$$|\mathbb{W}((0,1), V) \cup \mathbb{W}((0,-1), V)| = n, \text{ and}$$
$$|\mathbb{W}((1,0), V) \cup \mathbb{W}((-1,0), V)| = n.$$

Utilizing these n horizontal and n vertical witness lines, we are able to pick two additional directions to generate three-way filtration line intersections using a technique similar to the one described in Theorem 5 of [1]. Then, the following theorem holds as well.

Theorem 1 (ECC Vertex Reconstruction)**.** *Let* $K = \langle V, E \rangle$ *be a plane graph with vertices* V *and edges* E*. If for all* $v \in V$*,* $deg(v) \neq 2$ *then the locations of all vertices can be determined using six ECCs in* $O(n \log n)$ *time.*

The proof of Theorem 1 is found in Appendix A, but note that the result follows using similar arguments to those found in Theorem 5 of [1].

6 Discussion and Future Work

We have shown that, for any known plane graph K, we can choose a linear number of directions to fully describe K using only ECCs from those directions. However, when K is unknown, determining such a set is difficult. We emphasize that although there is an infinite number of directions in which the vertices of a plane graph can be witnessed by an ECC, the presence of degree two vertices can restrict these directions to an arbitrarily small subset of $\mathbb{S}^1$.

Our ultimate goal is to further develop the theory on determining the minimal set of directions necessary to reconstruct shapes. We are currently investigating upper bounds on the number of directions needed to reconstruct a plane graph from ECCs. Additionally, we are exploring what assumptions we can place on the underlying shape in order to overcome the challenges of degree two vertices. For example, we observe that if the number of vertices $|V| = n$ is known, then the intersection of $m > n$ filtration lines determines the location of all vertices. Another simplifying assumption is that minimum angle between any three vertices, ϵ, is known. Then, we can avoid some of the issues described in Section 4 by employing pairs of directions whose difference in angle is less than ϵ. Finally, we would like to extend our work to more general shapes embedded in $\mathbb{R}^d$.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. CCF 1618605 (authors BTF and SM) and Grant No. DBI 1661530; BTF and AS acknowledge the support of NIH and NSF under Grant No. NSF-DMS 1664858.

References

[1] Robin Lynne Belton, Brittany Terese Fasy, Rostik Mertz, Samuel Micka, David L Millman,

Daniel Salinas, Anna Schenfisch, Jordan Schupbach, and Lucia Williams. Learning simplicial complexes from persistence diagrams. *arXiv preprint arXiv:1805.10716*, 2018.

[2] Lorin Crawford, Anthea Monod, Andrew X. Chen, Sayan Mukherjee, and Ral Rabadn. Functional data analysis using a topological summary statistic: The smooth Euler characteristic transform. arXiv:1611.06818, 2016.

[3] Justin Curry, Sayan Mukherjee, and Katharine Turner. How many directions determine a shape and other sufficiency results for two topological transforms. arXiv:1805.09782, 2018.

[4] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction.* American Mathematical Society, 2010.

[5] Robert Ghrist, Rachel Levanger, and Huy Mai. Persistent homology and Euler integral transforms. arXiv:1804.04740, 2018.

[6] Johan W. H. Tangelder and Remco C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441, Dec 2007.

[7] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

A Proofs

Proof of Proposition 1 (ECC Existence)

Proof. Let $v \in V$ be a vertex in K. First, we show that each vertex is witnessed from an infinite number of directions $\mathbb{S}^1$. If $deg(v) = 0$, v is witnessed from any direction for which it lies on a unique witness line (So, for all but $|V|-1$ directions). If $deg(v) = 1$ with edge (v, v') for some $v' \in V$, then v is observed from an the infinite set of directions from which v' appears after v in the lower-star filtration, and v lies on a unique witness line. If $deg(v) > 1$ with edges (v, v') and (v, v'') for $v', v'' \in V$, then v is observed from any direction from which v' and v'' appear before v in the filtration and v lies on a unique witness line. Thus, each vertex is witnessed from an infinite number of directions.

Let $\mathbb{I}_v$ be the set of directions that witness v. We can choose any three directions from $\mathbb{I}_v$ and generate a unique three-way intersection at v. Now, we need to show that a set of directions exist for each of the n vertices such that no three-way intersections exist at locations where a vertex is not located. In order to do this, we give the vertices some arbitrary ordering $v_1, v_2, \ldots, v_n$. Then, select vertices in ascending order. For the first, any three directions in $\mathbb{I}_{v_1}$ will give a single three-way intersection of witness lines. For each successive vertex v_i, there exist up to $3i^2$ witness lines. More importantly, the number of three-way witness line intersections is finite. Thus, there exist three directions in $\mathbb{I}_{v_i}$ such that none of the witness lines created by these directions intersect existing intersections. Since the x- and y-coordinates of a vertex can be determined using a three-way line intersection, we can see that there exist a set of $3n$ directions which generates exactly n three-way intersections of witness lines, revealing the location of all n vertices.

□

Proof of Proposition 2 (Same Quadrant)

Proof. If v_1 and v_3 lie in the same quadrant, then v_1 and v_3 will appear before v_2 from exactly one of the two x-axis parallel directions directions $(-1, 0)$ or $(1, 0)$ and before v_2 in exactly one of the y-axis parallel directions $(0, -1)$ or $(0, 1)$. Let $s_1 \in \{(0, 1), (0, -1)\}$ and $s_2 \in \{(1, 0), (-1, 0)\}$ be the directions that witness v_1 and v_3 before v_2. $\chi^K_{s_1}$ and $\chi^K_{s_2}$ will witness v_2 by seeing a decrease in the Euler Characteristic at the time that v_2 is first included in the filtration. Then, $-s_1$ and $-s_2$ will witness v_2 before v_1 or v_2. Since no other edges with v_2 as an endpoint exist, there will be an increase in $\chi^K_{-s_1}$ and $\chi^K_{-s_2}$ at the time that v_2 is first included in the filtration. Then, v_2 is witnessed from every cardinal direction, as required. □

Proof of Proposition 3 (Neighboring Quadrants)

Proof. Recall that no two vertices share x- or y-coordinates, then any witness line from a cardinal direction will be unique. Let s be the cardinal direction for which $v_1 \cdot s < v_2 \cdot s$ and $v_3 \cdot s > v_2 \cdot s$ and $-s$ the cardinal direction chosen such that $v_3 \cdot s < v_2 \cdot s$ and $v_1 \cdot s > v_2 \cdot s$. Then, there is no change in Euler Characteristic at v_2 from either s or $-s$, since v_2 is added at the same time as (v_1, v_2) or (v_2, v_3), respectively. Now, let w and $-w$ be the remaining two cardinal directions, where w is the direction from which we include v_2 before v_1 or v_3. Direction w witnesses v_2 because no edges are included at height v_2 from that direction. Direction $-w$ witnesses v_2 because both (v_1, v_2) and (v_2, v_3) are added along with v_2. Thus,

v_2 is witnessed from exactly two of the four cardinal directions. □

Proof of Proposition 4 (Degree Two Bounded Angle)

Proof. If $angle((v_1, v_2), (v_2, v_3)) < \frac{\pi}{2}$, then v_1 and v_3 must lie in neighboring quadrants or the same quadrant, since neither can lie on the boundary of a quadrant by assumption. If they are in the same quadrant, Proposition 2 tells us that they must be seen from all four cardinal directions. If they are in neighboring quadrants, Proposition 3 tells us that we can witness v_2 with ECCs from exactly two of the four cardinal directions. □

Proof of Lemma 1 (Linear Witness Lines)

Proof. We show that each vertex is seen by at least one of s or $-s$. Let $v \in V$ be a vertex with $deg(v) = 0$. Then, v will correspond to $\ell(s, v)$ for any arbitrary direction $s \in \mathbb{S}^1$ because χ_s^K will always increase by at least one at time $s \cdot v$. As such, v will be observed by both s and $-s$.

Let $v \in V$ be a vertex with $deg(v) = 1$ and $(v, v') \in E$ for some $v' \in V$. Then, if $s \in \mathbb{S}^1$ is chosen such that $s \cdot v' < s \cdot v$, v will not result in a change in χ_s^K. However, s was chosen such that no two vertices will be observed at the same time. As a result, no edge in E can be parallel to ℓ. Then, if $s \cdot v' < s \cdot v$ then $-s \cdot v' \geq -s \cdot v$ and an increase in χ_{-s}^K is seen at time $-s \cdot v$. This implies that v is observed by s or $-s$ but not both.

Finally, if $v \in V$ is a vertex with $deg(v) > 2$, then we must consider two cases. If, for $s \in \mathbb{S}^1$, there exists exactly one edge $(v, v') \in E$ such that $s \cdot v' < s \cdot v$, then there must exist at least two additional edges that will result in a decrease in χ_{-s}^K at time $-s \cdot v$. As such, v will be observed by at least one of the ECCs resulting from s or $-s$. On the other hand, if, for $s \in \mathbb{S}^1$, there exists either zero edges or more than one edge that appear before v in the height filtration from s, then χ_s^K will either increase (in the case where no edges appear before v) or decrease (in the case where two or more edges appear before v). Then, all non-degree two vertices result in a change in χ_s^K or χ_{-s}^K and, as such, $|\mathbb{W}(s, V) \cup \mathbb{W}(-s, V)| = n$, as required. □

Proof of Lemma 2 (Witness Line Intersections)

Proof. By Lemma 1, if s is chosen such that no two vertices are intersected by a line perpendicular to s, then $\mathbb{W}(s, V) \cup \mathbb{W}(-s, V)$ will result in n filtration lines. Recall that no two vertices in K share an x- or y-coordinate. Then, by Lemma 1, $|\mathbb{W}((0, 1), V) \cup \mathbb{W}((0, -1), V)| = n$ and $|\mathbb{W}((1, 0), V) \cup \mathbb{W}((-1, 0), V)| = n$, as required. □

Proof of Theorem 1 (ECC Vertex Reconstruction)

Proof. Using Lemma 2 we construct n horizontal and n vertical lines corresponding to vertices using four ECCs and we denote them L_H and L_V respectively. Then, we must identify an additional two directions which will, together, generate an additional n unique witness lines and exactly n three-way filtration line intersections. We choose these final directions $s_3 \in \mathbb{S}^1$ and $-s_3$ using the method described in Theorem 5 of [1]. We observe that, by Lemma 4 of [1], no two vertices will be intersected by any single line perpindicular to s_3. Then, since each vertex will be witnessed by at least one of the ECCs from $\mathbb{W}(s_3, V)$ or $\mathbb{W}(-s_3, V)$ by Lemma 1, these two directions will yield n distinct filtration lines each of which will intersect exactly one two-way intersection between lines of L_H and L_V. Then, Lemma 3 of [1] implies that these three-way intersections are the locations of the n vertices in V. The $O(n \log n)$ running time follows from the proof of Theorem 5 in [1]. □

Colored range closest-pair problem under general distance functions*

Jie Xue
University of Minnesota, Twin Cities
xuexx193@umn.edu

1 Background

The closest-pair problem, which aims to find the closest pair of points in a given dataset of points in $\mathbb{R}^2$ (or more generally $\mathbb{R}^d$), is one of the most fundamental problems in computational geometry and finds many applications in various scenarios, e.g., traffic control, similarity search, etc. A natural and important generalization of this problem is the *colored* closest-pair problem in which the given data points are colored and the point pairs of interest are only the *bichromatic* ones (i.e., those consisting of two points of different colors), namely, we want to find the closest bichromatic pair of points. This colored version has applications in analyzing categorical data, and is strongly related to Euclidean minimum spanning tree [2, 6]. The closest-pair problem and its generalization can be considered under the Euclidean metric or more general metrics (such as L_p-metrics).

The *range closest-pair* (RCP) problem, introduced in [9] for the first time, is the range-search version of the classical (single-shot) closest-pair problem which aims to store a given dataset S into some data structure which can report, for a specified query range X, the closest pair of points in $S \cap X$. The RCP problem in $\mathbb{R}^2$ (under the Euclidean metric) has been studied in prior work over the last decades, see for example [1, 7, 8, 9, 10, 13].

Compared to traditional range-search problems, the RCP problem (even the uncolored version) is much more challenging due to a couple of reasons. First, as a range-search problem, the RCP problem is *non-decomposable* in the sense that even if the dataset S can be written as $S = S_1 \cup S_2$, the closest pair of points in $S \cap X$ cannot be obtained via those in $S_1 \cap X$ and $S_2 \cap X$. This makes the decomposition-based techniques inapplicable to the RCP problem. Second, since the RCP problem concerns the pairwise distances of the points, it is difficult to apply "mapping"-based approaches to solve the problem. For example, it is well-known that 2D circular range reporting can be reduced to 3D halfspace range reporting via a lifting argument. However, this reduction does not work for the RCP problem because the lifting changes the pairwise distances of the points. For another example, consider vertical strip queries in $\mathbb{R}^2$. Range reporting for vertical strip queries is "pointless", as it is just the 1D range-reporting problem, after projecting the data points to the x-axis. Again, this projection argument does not apply to the RCP problem as it changes the pairwise distances of the points, and the RCP problem for vertical strip queries is actually nontrivial [10, 13].

*Accepted to *ACM-SIAM Symposium on Discrete Algorithms* (SODA'19). A full version of the paper is available online [12] (https://arxiv.org/abs/1807.09977).

Similarly to the single-shot closest-pair problem, the RCP problem can be naturally generalized to the *colored range closest-pair* (CRCP) problem in which we want to store a colored dataset and report the closest bichromatic pair of points contained in a query range. Surprisingly, despite of much effort made on the RCP problem, this generalization has never been considered previously. In this paper, we make the first progress on the CRCP problem. Unlike the previous work, we do not restrict ourselves to the Euclidean metric. Instead, we investigate the problem under a general metric satisfying some certain condition. This covers all L_p-metrics for $p > 0$ (including the L_∞-metric). The CRCP problem is even harder than the (uncolored) RCP problem, especially when considered under such a general metric. As such, we are interested in answering CRCP queries *approximately*. That is, for a specified query range X, we want to report a bichromatic pair of points in X whose distance is at most $(1+\varepsilon)\cdot \mathsf{Opt}$ where Opt is the distance of the closest bichromatic pair of points in X and ε is a pre-specified parameter. Our main goal is to design efficient $(1+\varepsilon)$-approximate CRCP data structures in terms of *space* and *query time*.

2 Related work

The closest-pair problem and range search are both the most fundamental problems in Computational Geometry, see [3, 11] for surveys. Approximate range search is also well-studied in the last decades, see for example [4, 5]. The RCP problem was introduced by Shan et al. [9] for the first time. Subsequently, the problem in $\mathbb{R}^2$ was studied by [1, 7, 8, 10]. The papers [7, 8, 10] considered the problem with orthogonal queries, while [1] studied halfplane queries. Recently, Xue et al. [13] improved the above results. The state-of-the-art RCP data structure for rectangle query uses $O(n\log^2 n)$ space and $O(\log^2 n)$ query time [13]. In higher dimensions, the RCP problem is still open. To our best knowledge, the only known result that can be generalized to higher dimensions is a simple data structure given in [8], which only has guaranteed *average-case* performance. All these results were limited to the Euclidean metric and uncolored case.

3 Our contributions and techniques

We investigate the CRCP problem under a general metric that is induced by a monotone norm (see the full version [12] for a formal definition); in particular, this includes all L_p-metrics for $p > 0$ and the L_∞-metric. We design $(1+\varepsilon)$-approximate CRCP data structures for orthogonal queries in $\mathbb{R}^2$ and higher dimensions, where $\varepsilon > 0$ is a pre-specified parameter. The performances of these data structures are summarized in Table 1, and we give a brief explanation below.

Our main result is two $(1+\varepsilon)$-approximate CRCP data structures for rectangle queries in $\mathbb{R}^2$; see the gray rows of Table 1. In the process of designing these data structures, we also obtain efficient data structures for strip and quadrant queries. Using similar techniques, we also achieve results in higher dimensions. Specifically, we design data structures for slab and 2-box queries in $\mathbb{R}^d$ (which are generalizations of strip and quadrant queries in $\mathbb{R}^2$ respectively) and dominance queries in $\mathbb{R}^3$. All of our data structures use *near-linear* space and *poly-logarithmic* query time (when ε is regarded as a constant). Preprocessing time is not considered in this paper, and we leave this as an open question for future work[1]. Our new results are interesting for the following reasons.

[1]Preprocessing RCP-related data structures is usually a hard task. For instance, how to build efficiently the state-of-the-art orthogonal RCP data structures in [13] is still unknown.

- Previously, only the uncolored RCP problem was studied. We make the first progress on the CRCP problem. Furthermore, we do not make any assumption on the coloring of the dataset.
- The previous work considered the RCP problem only under the Euclidean metric. Our results can be applied to a quite general class of metrics including all L_p-metrics.
- Almost all existing results on the RCP problem were restricted to $\mathbb{R}^2$. Our techniques give some results beyond that (while our main focus is still on $\mathbb{R}^2$), leading us towards better understanding of the CRCP problem in higher dimensions.

Dimension	Query	Space	Query time
$\mathbb{R}^2$	Strip	$O(\varepsilon^{-1} n \log^2 n)$	$O(\log n + \log(1/\varepsilon))$
	Quadrant	$O(\varepsilon^{-1} n \log^2 n)$	$O(\log n + \log(1/\varepsilon))$
	Rectangle	$O(\varepsilon^{-1} n \log^4 n)$	$O(\log^4 n + \varepsilon^{-1} \log^3 n + \varepsilon^{-2} \log n)$
		$O(\varepsilon^{-1} n \log^3 n)$	$O(\log^5 n + \varepsilon^{-1} \log^4 n + \varepsilon^{-2} \log^2 n)$
$\mathbb{R}^d$	Slab	$O(\varepsilon^{-1} n \log^d n)$	$O(\log n + \log(1/\varepsilon))$
	2-Box	$O(\varepsilon^{-1} n \log^d n)$	$O(\log n + \log(1/\varepsilon))$
$\mathbb{R}^3$	Dominance	$O(\varepsilon^{-1} n \log^6 n)$	$O(\varepsilon^{-2} \log^9 n + \varepsilon^{-4} \log^3 n)$

Table 1: The performances of our $(1+\varepsilon)$-approximate CRCP data structures.

Our techniques. Unfortunately, the techniques used in the (uncolored) RCP problem are inapplicable to the CRCP problem even under the Euclidean metric. Thus, we develop new techniques to solve the problem. Our first technical contribution is the notion of *RCP coresets*. Roughly speaking, an RCP coreset of a set of point pairs is a subset that approximately preserves the closest-pair information in every query range. This notion gives us a natural way to design an approximate CRCP data structure, namely, storing an RCP coreset of the set of all bichromatic pairs and searching for the answer in the coreset. This idea works only when there exists a small-size RCP coreset. We prove that if the query space (i.e., the collection of the query ranges) satisfies some nice property and the metric is induced by a monotone norm, then a small-size RCP coreset always exists. Using this result, we obtain efficient approximate CRCP data structures for both strip and quadrant queries. Our second technique is an *anchored* version of the CRCP problem, in which an anchor point o is specified with the query range and we want to report the closest o-anchored bichromatic pair (see the full version [12] for a formal definition) contained in the query range. We give an efficient (approximate) anchored CRCP data structure for rectangle queries, which works for any metric induced by a monotone norm. Based on the above results, we design our rectangle CRCP data structures. The main idea is to use range trees to reduce a rectangle CRCP query to several strip and quadrant CRCP queries and anchored CRCP queries. Using our strip/quadrant CRCP data structures and anchored CRCP data structure mentioned above, we eventually obtain the two approximate CRCP data structures for rectangle queries. Our results in higher dimensions are achieved using similar techniques and ideas.

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Workshop on Algorithms and Data Structures*, pages 1–12. Springer, 2009.

[2] Pankaj K Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete & Computational Geometry*, 6(3):407–422, 1991.

[3] Pankaj K Agarwal, Jeff Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

[4] Sunil Arya and David M Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.

[5] Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Computational Geometry*, 39(1):24–29, 2008.

[6] David Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Discrete & Computational Geometry*, 13(1):111–122, 1995.

[7] Prosenjit Gupta. Range-aggregate query problems involving geometric aggregation operations. *Nordic journal of Computing*, 13(4):294–308, 2006.

[8] Prosenjit Gupta, Ravi Janardan, Yokesh Kumar, and Michiel Smid. Data structures for range-aggregate extent queries. *Computational Geometry: Theory and Applications*, 2(47):329–347, 2014.

[9] Jing Shan, Donghui Zhang, and Betty Salzberg. On spatial-range closest-pair query. In *International Symposium on Spatial and Temporal Databases*, pages 252–269. Springer, 2003.

[10] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. *Technical Report IIIT/TR/2007/80. IIIT Hyderabad, Telangana*, 500032, 2007.

[11] Michiel Smid. *Closest point problems in computational geometry.* Citeseer, 1995.

[12] Jie Xue. Colored range closest-pair problem under general distance functions. *arXiv preprint arXiv:1807.09977*, 2018.

[13] Jie Xue, Yuan Li, Saladi Rahul, and Ravi Janardan. New bounds for range closest-pair problems. *Proceedings of the 34th International Symposium on Computational Geometry*, 2018.

Local cliques in ER-perturbed random geometric graphs

Matthew Kahle[1], Minghao Tian[2], and Yusu Wang[2]

[1]Department of Mathematics, The Ohio State University, USA
[2]Computer Science and Engineering Dept., The Ohio State University, USA

1 Introduction

Random graphs are mathematical models which have applications in a wide spectrum of domains. Erdős–Rényi graph $G(n, p)$ is one of the oldest and most-studied models for networks [15], constructed by adding edges between all pairs of n vertices with probability p independently. Many global properties of this model are well-studied by using probabilistic method [1], such as the clique number and the phase transition behaviors of connected components w.r.t. parameter p.

Another classical type of random graphs is the random geometric graph $G(\mathbb{X}_n; r)$ introduced by Edgar Gilbert in 1961 [10]. This model starts with a set of n points $\mathbb{X}_n$ randomly sampled over a metric space (typically a cube in $\mathbb{R}^d$) from some probability distribution, and edges are added between all pairs of points within distance r to each other. The Erdős–Rényi random graphs and random geometric graphs exhibit similar behavior for the Poisson degree distribution; however, other properties, such as the clique number and phase transition (w.r.to p or to r), could be very different [11, 18, 17, 13]. This model has many applications in real world where the physical locations play an important role (e.g, transportation networks [2]).

We are interested in mixed models that "combine" both types of randomness together. One way to achieve this is to add Erdős–Rényi type perturbation (percolation) to random geometric graphs. Although these graphs are related to the continuum percolation theory [14], our understanding about them so far is still limited: In previous studies, the underlying spaces are typically plane[4] and cubes [7]; the vertices are often chosen as the lattices of the space; and the results usually concern the connectivity [5] or diameter (e.g, [20]).

Cliques in graphs are important objects in many application domains (e.g, in PPI networks [19]). The clique number in Erdős–Rényi random graph has been studied extensively in 20th century [3]. The clique number in random geometric graph is a relative new topic. It has dramatically different behaviors when different ranges of r are chosen [13].

Our work. In this short paper[1], we consider a mixed model of Erdős–Rényi random graphs and random geometric graphs, and study the behavior of a local property called *edge clique number*. More precisely, we use the following *ER-perturbed random geometric graph model* previously introduced in [16]. Suppose there is a compact metric space $\mathcal{X} = (X, d)$ (as feature space) with a probability distribution induced by a "nice" measure μ supported on $\mathcal{X}$ (e.g, the uniform measure supported on an embedded smooth low-dimensional Riemannian manifold). Assume we now randomly sample n points V i.i.d from this measure μ, and build the random geometric graph $G^*_{\mathcal{X}}(r)$, which is the r-neighborhood graph spanned by V (i.e, two points $u, v \in V$ are connected if their distance $d(u, v) \leq r$). Next, we add Erdős–Rényi (ER) type perturbation to $G^*_{\mathcal{X}}(r)$: each edge in $G^*_{\mathcal{X}}(r)$ is deleted with a uniform probability p, while each "short-cut" edge between two unconnected nodes u, v is inserted to $G^*_{\mathcal{X}}(r)$ with a uniform probability q. We denote the resulting generated graph as $\widehat{G}^{p,q}_{\mathcal{X}}(r)$.

We introduce a local property called the *edge clique number* of a graph G. It is defined for each edge (u, v) in the graph, denoted as $\omega_{u,v}(G)$, as the size of the largest clique containing uv in graph G. Our main result is that $\omega_{u,v}(G)$ presents two fundamentally different types of behaviors. See Theorems 3.2, 3.7, 3.8, and 3.9.

As an application of our theoretical analysis, in Theorem 4.1, we show that by using a filtering process based

[1]The full version: http://web.cse.ohio-state.edu/~tian.394/papers/clique_number_RGG_under_ER_perturbation.pdf

on our edge clique number, we can recover the shortest-path metric of the random geometric graph $G^*_{\mathcal{X}}(r)$ within a multiplicative factor of 3, from an ER-perturbed graph $\widehat{G}^{p,q}_{\mathcal{X}}(r)$, for a significantly wider range of insertion probability q than what's required in [16], although we do need a stronger condition on the measure μ.

2 Preliminaries

Suppose we are given a compact geodesic metric space $\mathcal{X} = (X, d)$ [6]. We will consider "nice" measures on $\mathcal{X}$. Specifically,

Definition 2.1 (Doubling measure) *Given a metric space $\mathcal{X} = (X, d)$, let $B_r(x) \subset X$ denotes the closed metric ball $B_r(x) = \{y \in X \mid d(x, y) \leq r\}$. A measure $\mu : X \to \mathbb{R}$ on $\mathcal{X}$ is said to be* doubling *if every metric ball (with positive radius) has finite and positive measure and there is a constant $L = L(\mu)$ s.t. for all $x \in X$ and every $r > 0$, we have $\mu(B_{2r}(x)) \leq L \cdot \mu(B_r(x))$. We call L the* doubling constant *and say μ is an L-*doubling measure.

For our theoretical results later, we in fact need a stronger condition on the input measure, which we will specify later in Assumption-A at the beginning of Section 3.

ER-perturbed random geometric graph. Following [16], we consider the following random graph model: Given a compact metric space $\mathcal{X} = (X, d)$ and a L-doubling probability measure μ supported on X, let V be a set of n points sampled i.i.d. from μ. We build the $r-$neighborhood graph $G^*_{\mathcal{X}}(r) = (V, E^*)$ for some parameter $r > 0$ on V; that is, $E^* = \{(u, v) \mid d(u, v) \leq r, u, v \in V\}$. We call $G^*_{\mathcal{X}}(r)$ a random geometric graph generated from $(\mathcal{X}, \mu, r)$. Now we add the following two types of random perturbations:

- p-deletion: For each existing edge $(u, v) \in E^*$, we delete edge (u, v) with probability p.
- q-insertion: For each non-existent edge $(u, v) \notin E^*$, we insert edge (u, v) with probability q.

The final graph $\widehat{G}^{p,q}_{\mathcal{X}}(r) = (V, \widehat{E})$ is called a *(p, q)-perturbation of $G^*_{\mathcal{X}}(r)$*, or simply an *ER-perturbed random geometric graph*.

Definition 2.2 (Edge clique number) *Given an arbitrary graph $G = (V, E)$, for any edge $(u, v) \in E$, its* edge clique number $\omega_{u,v}(G)$ *is defined as:*

$\omega_{u,v}(G) =$ *the size of the largest clique(s) in G containing (u, v).*

Setup for the remainder of the paper. In what follows, we fix the compact geodesic metric space $\mathcal{X} = (X, d)$, the L-doubling probability measure μ, and the set of n graph nodes V sampled i.i.d from μ. The input is a (p, q)-perturbation $\widehat{G} = \widehat{G}^{p,q}_{\mathcal{X}}(r) = (V, \widehat{E})$ of a random geometric graph $G^* = G^*_{\mathcal{X}}(r)$ spanned by V with radius parameter r. For an arbitrary graph G, let $V(G)$ and $E(G)$ refer to its vertex set and edge set, respectively, and let $N_G(u)$ denote the set of neighbors of u in G.

Definition 2.3 (Good / bad-edges) *An edge (u, v) in the perturbed graph $\widehat{G}$ is a* good-edge *if $d(u, v) \leq r$. An edge (u, v) in the perturbed graph $\widehat{G}$ is a* bad-edge *if for any $x \in N_{G^*}(u)$ and $y \in N_{G^*}(v)$, we have $d(x, y) > r$.*

Our main result Theorem 3.9 roughly suggests, under certain conditions on the insertion probability q, for a good-edge (u, v) of $\widehat{G}^{p,q}_{\mathcal{X}}(r)$, with high probability, $\omega_{u,v}\left(\widehat{G}\right)$ has order $\Omega\left(\log_{1/(1-p)} \ln n\right)$; while for a bad-edge (u, v), its edge-clique number $\omega_{u,v}\left(\widehat{G}\right)$ has order $o\left(\log_{1/(1-p)} \ln n\right)$ with high probability.

All the missing proofs can be found in the full version.

3 Two different behaviors of edge clique number

For technical reasons, we need an assumption on the parameter r (for the random geometric graph $G^*_{\mathcal{X}}(r)$), as well as a condition on the measure μ where graph nodes V are sampled from.

[Assumption-A]: The parameter r and the doubling measure μ satisfy the following condition:

There exist $\mathrm{s} \geq \frac{13 \ln n}{n} \left(= \Omega(\frac{\ln n}{n})\right)$ and a constant ρ such that for any $x \in X$

(Density-cond) $\mu\left(B_{r/2}(x)\right) \geq \mathrm{s}$.
(Regularity-cond) $\mu\left(B_{r/2}(x)\right) \leq \rho \mathrm{s}$

Density-cond is equivalent to the Assumption-R in [16]. It intuitively requires that r is large enough such that with high probability each vertex v in the random geometric graph $G^*_{\mathcal{X}}(r)$ has degree $\Omega(\ln n)$. Indeed, the following is already known.

Claim 3.1 ([16]) *Under* Density-cond*, with probability at least* $1 - n^{-5/3}$*, each vertex in* $G^*_{\mathcal{X}}(r)$ *has at least* $sn/4$ *neighbors.*

3.1 Insertion-only perturbation

First, for good-edges, we have the following result.

Theorem 3.2 *Assume* Density-cond *holds. Let* $\widehat{G} = \widehat{G}^q$ *denote the final graph after inserting each edge not in* G^* *independently with probability* q*. Then, with high probability, for each good-edge* (u, v) *in* $\widehat{G}$*, its edge clique number satisfies that* $\omega_{u,v}(\widehat{G}) \geq sn/4$*.*

Bounding the edge clique number for bad-edges is much more challenging, due to the interaction between local edges (from random geometric graph) and long-range edges (from random insertions). To handle this, we will create a specific collection of subgraphs for $\widehat{G}$ in an appropriate manner, and bound the edge clique number of a bad-edge in each such subgraph. The property of this specific collection of subgraphs is that the union of these individual cliques provides an upper bound on the edge clique number for this edge in $\widehat{G}$. To construct this collection of subgraphs, we will use the so-called Besicovitch covering lemma which has a lot of applications in measure theory [9].

First, we introduce some notations. We use a *packing* to refer to a countable collection $\mathcal{B}$ of *pairwise disjoint* closed balls. Such a collection $\mathcal{B}$ is a *packing w.r.t. a set* P if the centers of the balls in $\mathcal{B}$ lie in the set $P \subset X$, and it is a δ*-packing* if all of the balls in $\mathcal{B}$ have radius δ. A set $\{A_1, \ldots, A_\ell\}, A_i \subseteq X$, *covers* P if $P \subseteq \bigcup_i A_i$.

Theorem 3.3 (Besicovitch Covering Lemma[12]) *Let* $\mathcal{X} = (X, d)$ *be a doubling space. Then, there exists a constant* $\beta = \beta(\mathcal{X}) \in \mathbb{N}$ *such that for any* $P \subset X$ *and* $\delta > 0$*, there are* β *number of* δ*-packings w.r.t.* P*, denoted by* $\{\mathcal{B}_1, \cdots, \mathcal{B}_\beta\}$*, whose union also covers* P*.*

We call the constant $\beta(\mathcal{X})$ above the *Besicovitch constant*. Given a set P, we say that A is *partitioned into* $A_1, A_2, \cdots, A_k$, if $A = A_1 \cup \cdots \cup A_k$ and $A_i \cap A_j = \emptyset$ for any $i \neq j$.

Definition 3.4 (Well-separated clique-partitions family) *Consider the random geometric graph* $G^* = G^*_{\mathcal{X}}(r)$*. A family* $\mathcal{P} = \{P_i\}_{i \in \Lambda}$*, where* $P_i \subseteq V$ *and* Λ *is the index set of* P_i*s, forms a* well-separated clique-partitions family *of* G^* *if:*

1. $V = \cup_{i \in \Lambda} P_i$*.*
2. $\forall i \in \Lambda$*,* P_i *can be partitioned as* $P_i = C_1^{(i)} \sqcup C_2^{(i)} \sqcup \cdots \sqcup C_{m_i}^{(i)}$ *where*
   - *(2-a)* $\forall j \in [1, m_i]$*,* $\exists \bar{v}_j^{(i)} \in V$ *such that* $C_j^{(i)} \subseteq B_{r/2}\left(\bar{v}_j^{(i)}\right) \cap V$*.*
   - *(2-b)* $\forall j_1, j_2 \in [1, m_i]$ *with* $j_1 \neq j_2$*,* $d_H\left(C_{j_1}^{(i)}, C_{j_2}^{(i)}\right) > r$*, where* d_H *is the Hausdorff distance between two sets in metric space* (X, d)*.*

We also call $C_1^{(i)} \sqcup C_2^{(i)} \sqcup \cdots \sqcup C_{m_i}^{(i)}$ *a* clique-partition of P_i (w.r.t. G^*)*, and its* size *(cardinality) is* m_i*. The* size *of the well-separated clique-partitions family* $\mathcal{P}$ *is its cardinality* $|\mathcal{P}| = |\Lambda|$*.*

By applying Theorem 3.3, we have the following lemma.

Lemma 3.5 *There is a well-separated clique-partitions family* $\mathcal{P} = \{P_i\}_{i \in \Lambda}$ *of* G^* *with* $|\Lambda| \leq \beta^2$*, where* $\beta = \beta(\mathcal{X})$ *is the Besicovitch constant of* $\mathcal{X}$*.*

Using Chernoff bound and the Assumption-A, we can also upper-bound the number of points in every $r/2$-ball centered at nodes of G^*, which further upper-bounds the number of points in each clique of a well-separated clique-partition.

Claim 3.6 *Given an* n*-node random geometric graph* $G^* = (V, E^*)$ *generated from* $(\mathcal{X}, \mu, r)$*, if* Assumption-A *holds, then with probability at least* $1 - n^{-5}$*, for every* $v \in V$*, the ball* $B_{r/2}(v) \cap V$ *contains at most* $3\rho sn$ *points.*

By using the well-separated clique-partition technique, one can prove the following theorem.

Theorem 3.7 *Suppose* Assumption-A *holds. Let* $\widehat{G} = \widehat{G}^q$ *denote the graph obtained by inserting each edge not in* G^* *independently with probability* q*. Then there exist constants* $c_1, c_2, c_3 > 0$ *which depend on the doubling constant* L *of* μ*, the Besicovitch constant* $\beta(\mathcal{X})$*, and the regularity constant* ρ*, such that for any* $\mathsf{K} = \mathsf{K}(n)$ *with* $\mathsf{K} \to \infty$ *as* $n \to \infty$*, with high probability,* $\omega_{u,v}(\widehat{G}) < \mathsf{K}$ *for any bad-edge* (u, v) *in* $\widehat{G}$*, as long as* q *satisfies* $q \leq \min\left\{c_1, c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\mathsf{K}} \cdot \frac{\mathsf{K}}{sn}\right\}$*.*

Remark. Consider for example when $\mathsf{K} = \Theta(sn)$. Then the theorem says that there exists constant c' such that if $q < c'$, then w.h.p. $\omega_{u,v} < \mathsf{K}$ (thus $\omega_{u,v} = O(sn)$) for any bad-edge (u, v). Now consider when $q = o(1)$. Then the theorem implies that w.h.p. the edge-clique number for any bad-edge is at most $\mathsf{K} = o(sn)$. This is qualitatively different from the edge-clique number for a good-edge for the case $q = o(1)$, which is $\Omega(sn)$ as shown in Theorem 3.2. By reducing this insertion probability q, this gap can be made **larger and larger**.

3.2 Deletion-only perturbation

We now assume that the input graph $\widehat{G} = \widehat{G}^p$ is obtained by deleting each edge in the random geometric graph G^* independently with probability p. An observation is that for any edge (u, v) in G^*, as $d_X(u, v) \leq r$, we have that $B_r(u) \cap B_r(v)$ must contain a metric ball of radius $r/2$. Based on this observation, we have the following theorem.

Theorem 3.8 *Assume* Density-cond *holds. Let $\widehat{G} = \widehat{G}^p$ denote the final graph after deleting each edge in G^* independently with probability p. Then, for any constant $p \in (0, 1)$, with high probability, we have $\omega_{u,v}(\widehat{G}) \geq \frac{2}{3}\log_{1/(1-p)} sn$ for all edges (u, v) in $\widehat{G}$.*

3.3 Combined Case

Theorem 3.9 (ER-perturbed random geometric graph) *Assume* Assumption-A *holds. Let $\widehat{G} = \widehat{G}^{p,q}(r)$ denote the graph obtained by removing each edge in G^* independently with constant probability $p \in (0, 1)$ and inserting each edge not in G^* independently with probability q. There exist constants $c_1, c_2, c_3 > 0$ which depend on the doubling constant L of μ, the Besicovitch constant $\beta(\mathcal{X})$, and the regularity constant ρ, such that the following holds for any $\mathsf{K} = \mathsf{K}(n)$ with $\mathsf{K} \to \infty$ as $n \to \infty$*

1. *W.h.p., for all good-edges $(u, v), \omega_{u,v}(\widehat{G}) \geq \frac{2}{3}\log_{1/(1-p)} sn$.*
2. *W.h.p., for all bad-edges (u, v), $\omega_{u,v}(\widehat{G}) < \mathsf{K}$ as long as the insertion probability q satisfies $q \leq \min\left\{c_1, c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\mathsf{K}} \cdot \frac{\mathsf{K}}{sn\sqrt{1-p}}\right\}$*

Remark. For example, assume $sn = \Theta(\ln n)$. Then for a constant deletion probability $p \in (0, 1)$, w.h.p. the edge clique number for any good-edge is **at least** $\Omega\left(\log_{1/(1-p)} sn\right) = \Omega(\ln\ln n)$. For any bad-edge uv, if the insertion probably $q = o\left(\left(\frac{1}{n}\right)^{\frac{c_3}{\ln\ln n}} \frac{\ln\ln n}{\ln n}\right)$, then its edge clique number is **at most** $o\left(\log_{1/(1-p)} sn\right) = o(\ln\ln n)$ w.h.p..

4 Recover the shortest-path metric of $G^*(r)$

In this section, we show an application in recovering the shortest-path metric structure of $G^*_{\mathcal{X}}(r)$ from an input observed graph $\widehat{G}^{p,q}_{\mathcal{X}}(r)$, which was introduced in [16].

Specifically, given two different metrics defined on the same space (Y, d_1) and (Y, d_2), we say that $d_1 \leq \alpha \cdot d_2$ if for any two points $y_1, y_2 \in Y$, we have that $d_1(y_1, y_2) \leq \alpha \cdot d_2(y_1, y_2)$. The metric d_1 is an *α-approximation* of d_2 if $\frac{1}{\alpha} \cdot d_2 \leq d_1 \leq \alpha \cdot d_2$ for $\alpha \geq 1$.

Our edge-clique based filtering process is the following.

τ-Clique filtering: Given graph $\widehat{G}$, we construct another graph $\tilde{G}_\tau$ on the same vertex set as follows: For each edge $(u, v) \in E(\widehat{G})$, we insert the edge (u, v) into $E(\tilde{G}_\tau)$ if and only if $\omega_{u,v}(\widehat{G}) \geq \tau$.

The following result can be proved by almost the same argument as that for Theorem 12 of [16], with the help of Theorem 3.9.

Theorem 4.1 *Assume* Assumption-A *holds. Let $\tilde{G}_\tau$ denote the resulting graph after τ-Clique filtering. Then there exist constants $c_1, c_2, c_3 > 0$ which depend on the doubling constant L of μ, the Besicovitch constant $\beta(\mathcal{X})$, and the regularity constant ρ, such that if $p \in (0, c_0)$, $\tau \leq \frac{2}{3}\log_{1/(1-p)} sn$, and $q \leq c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn\sqrt{1-p}}$, then, with high probability, the shortest-path metric $d_{\tilde{G}_\tau}$ is a 3-approximation of the shortest-path metric d_{G^*} of G^*.*

However, if the deletion probability $p = 0$, then we have w.h.p. that $d_{\tilde{G}_\tau}$ is a 3-approximation of d_{G^*} as long as $\tau < \frac{sn}{4}$, and $q \leq \min\left\{c_1, c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn}\right\}$.

Remark. Consider the insertion-only case. If we choose $\tau = \ln n$ and assume that $sn > 4\tau$, then w.h.p. we can recover the shortest-path metric within a factor of 3 as long as $q \leq c\frac{\ln n}{sn}$ for some constant $c > 0$. If $sn = \Theta(\ln n)$ (but $sn > 4\tau = 4\ln n$), then q is only required to be smaller than a (sufficiently small) constant. If $sn = \ln^a n$ for some $a > 1$, then we require that $q \leq \frac{c}{\ln^{a-1} n}$. In constrast, the work of [16] requires that $q = o(s)$, which is $q = o(\frac{\ln^c n}{n})$ if $sn = \ln^a n$ with $a \geq 1$. The gap (ratio) between these two bounds is nearly a factor of n.

References

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016.

[2] Philippe Blanchard and Dimitri Volchenkov. *Mathematical analysis of urban spatial networks*. Springer Science & Business Media, 2008.

[3] Béla Bollobás and Paul Erdös. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 80, pages 419–427. Cambridge University Press, 1976.

[4] Béla Bollobás and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.

[5] L Booth, J Bruck, M Cook, and M Franceschetti. Ad hoc wireless networks with noisy links. In *Information Theory, 2003. Proceedings. IEEE International Symposium on*, pages 386–386. IEEE.

[6] Martin R Bridson and André Haefliger. *Metric spaces of non-positive curvature*, volume 319. Springer Science & Business Media, 2011.

[7] Don Coppersmith, David Gamarnik, and Maxim Sviridenko. *The Diameter of a Long-Range Percolation Graph*, pages 147–159. Birkhäuser Basel, Basel, 2002.

[8] Martin Doležal, Jan Hladkỳ, and András Máthé. Cliques in dense inhomogeneous random graphs. *Random Structures & Algorithms*, 2017.

[9] Herbert Federer. *Geometric measure theory*. Springer, 2014.

[10] Edward N Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961.

[11] Piyush Gupta and Panganamala R Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic analysis, control, optimization and applications*, pages 547–566. Springer, 1999.

[12] Antti Kaenmaki, Tapio Rajala, and Ville Suomala. Local homogeneity and dimensions of measures. *ANNALI DELLA SCUOLA NORMALE SUPERIORE DI PISA-CLASSE DI SCIENZE*, 16(4):1315–1351, 2016.

[13] Colin McDiarmid and Tobias Müller. On the chromatic number of random geometric graphs. *Combinatorica*, 31(4):423–488, 2011.

[14] Ronald Meester and Rahul Roy. *Continuum percolation*, volume 119. Cambridge University Press, 1996.

[15] Mark EJ Newman. Random graphs as models of networks. *Handbook of Graphs and Networks: From the Genome to the Internet*, pages 35–68.

[16] Srinivasan Parthasarathy, David Sivakoff, Minghao Tian, and Yusu Wang. A quest to unravel the metric structure behind perturbed networks. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 53:1–53:16, 2017.

[17] Mathew Penrose. *Random geometric graphs*. Number 5. Oxford University Press, 2003.

[18] Mathew D. Penrose. The longest edge of the random minimal spanning tree. *Ann. Appl. Probab.*, 7(2):340–361, 05 1997.

[19] Sriganesh Srihari, Chern Han Yong, and Limsoon Wong. *Computational Prediction of Protein Complexes from Protein Interaction Networks*. Morgan & Claypool, 2017.

[20] Xian Yuan Wu. Mixing time of random walk on poisson geometry small world. *Internet Mathematics*, 2017.

Lower Bounds for Indexing the k-Nearest Neighbor Problem

Paul Cesaretti
Queens College, CUNY
Flushing, New York
paul.cesaretti@qc.cuny.edu

Mayank Goswami
Queens College, CUNY
Flushing, New York
mayank.goswami@qc.cuny.edu

ABSTRACT

Similarity search for high dimensional data has been a major research problem for the last three decades. However, most of the techniques for dealing with similarity search in both its approximate and exact setting do not scale well to external memory. We consider the problem of constructing efficient indexing methods for the k-nearest neighbor problem (k-NN). Using the theoretical framework for space-query tradeoff of an indexing scheme for external memory developed by Hellerstein et al. [19], [20], we show that the k-NN problem is isomorphic to a general family of set workloads, namely the λ-set workload, implying that k-NN shares the same lower bounds. The theory of indexability puts these lower bounds in terms of *storage redundancy* r, the number of times each item in the data set is stored, and *access overhead* A, the number of blocks that a query retrieves. Our results imply that if $k = \Theta(1)$, $N = |I|$ is the size of the instance of the indexing scheme, and B is the size of the block, then in order to get access overhead less than k (meaning for k-NN queries to take fewer than k I/Os in the worst case), r would have to be $\Omega(N/B)$, implying that the space would have to be $\Omega((N/B)^2)$. We prove this result for both Euclidean and Hamming metric spaces. Furthermore, though lower bounds have been found for the nearest neighbor problem ($k = 1$), and despite the results of Chakrabarti et al. [13] and Barkol and Rabani [7] who found lower bounds in the cell probe model, these do not consider I/O nor do they apply directly to the k-NN problem.

1 INTRODUCTION

The relational database model developed by E.F. Codd in 1970 [14], which has served as the standard for most commercial database management systems owes much of its ubiquity to the development of an effective access method, the B-tree [15]. However, despite its extended use and success, the B-tree is inadequate for many novel database schemes, which has led much research in the database community around the development of new indexing methods to solve domain specific problems. For example, a survey by Gaede and Günther [17] cites over 50 different multidimensional access methods for spatial databases, implying that no one, general method is capable of accommodating the wide range of applications and requirements for contemporary information systems. Moreover, analytic research on these structures have focused on their average-case performance under various data sets and query distributions. This highlights the need to develop methods that can be used to analyze the viability of particular indexing scheme in terms of its performance and limitations, a priori.

Hellerstein et al. 1995 proposed and implemented a "generalized indexing" scheme [21] allowing new data types to be indexed in a manner that makes queries natural for its type. Called the Generalized Search Tree (GiST), it attempts to provide extensibility for new data types as well as provide a common code base for indexing structures used for disparate application domains; as such, implementations of the GiST exist that are able to function as either a B-tree, R-tree or RD-tree. The main theoretical question is whether a general framework exists to access the practicality of indexing arbitrary domains, both standard and non-standard.

The authors followed this work with a *theory of indexability* [19], which characterizes the hardness of indexing data sets on block memory devices. The theory singles out two measures of efficiency for an indexing scheme on a workload: *storage redundancy* and *access overhead*, where storage redundancy describes the number of times an item in the data set is stored in the indexing scheme, and access overhead describes how many more blocks than necessary are needed to answer the query. Both are expressed solely in terms of a parameter B, the block size. The workload then serves the same purpose as that of recursive languages in computability theory: as the unit on which a particular indexing scheme can be characterized. By defining a workload in such sparse terms, the approach neglects many important concerns in constructing any indexing scheme, namely the algorithm needed to determine which blocks within the index cover the query as well as the storage and retrieval costs of the algorithm or any auxiliary data structures. Furthermore, the model also ignores the dynamic aspect of the indexing problem, so the costs of insertion or deletion are not considered. These omissions are justified since the authors are primarily concerned with a theory that is capable of finding lower bounds that characterize common workloads. The theory is applied to examine the trade offs between storage redundancy and access overhead between two common family of workloads: two-dimensional range queries and set inclusion queries.

An important workload that we would like to consider is nearest neighbor search. That is, given a set P of points in high-dimensional space, construct a data structure, which, given any query point q, finds the point $p \in P$ closest to q. The problem is important to many areas of computer science, for example networking[12], file systems [9], data mining [11], machine learning and vision [28], pattern recognition [8], analysis [3],[4],[8], and optimization [2], [16], [22]. The basic problem is to perform similarity search for a query, where these objects consist of many features which represent the dimensions of the space being considered; for example an image of 1000 x 1000 pixels can be represented as a 1,000,000 dimensional vector [4]. Similarity indices for high dimensional data have been a major research problem for the last three decades [23], [3], [25], [27], [24], [10]. The major difficulty in constructing efficient algorithms for nearest neighbor search is that their space and time requirements grow as exponentially in the dimension [18], referred to as the *curse of dimensionality*. The most effective approach developed by Indyk and Motwani in 1998 loosens the requirements on finding the closest point $p \in P$ to the query to

finding a point whose distance to the query q is an $(1+\epsilon)$ factor away from q to closet point p [23]. However, the techniques for dealing with similarity search in both its approximate and exact setting work in main memory; none scale well to external memory. As data set become increasingly large, knowing what limits exist for constructing indexing schemes for external memory are becoming paramount.

1.1 Our Results

We work in the indexability model of Hellerstein et al. [19] and derive lower bounds on the Euclidean $\mathbb{R}^d$ and the Hamming $\{-1/\sqrt{d}, 1/\sqrt{d}\}^d$ versions of the k-NN problem. Our main result (Theorem 3.1 in Section 3) translated from indexability model language is the following.

Theorem 1.1 (Main result (informal)). *Let $d \in \omega(k^2 \log n)$. Consider the Euclidean-$\mathbb{R}^d$ or Hamming-$\{-1/\sqrt{d}, 1/\sqrt{d}\}^d$ versions of the k-NN problem for $k \leq B$. Any indexing scheme that has worst case query time at most $k-1$ I/Os must use $\Omega((N/B)^2)$ blocks of space on disk.*

Remarks

(1) Notice that the nearest neighbor problem ($k = 1$) gives trivial lower bounds in the indexability model since the block containing the answer to the query is given for free; the search aspect is completely removed from consideration, and the algorithm would return the block in one I/O. There are several approximate data structures for the nearest neighbor problem that use subquadratic space [23],[5]. In the latest paper, Andoni et al. [5] build upon the recent work on data-dependent hashing, and find a tradeoff between the space and query which is optimal in the sense that no list-of-points data structure can do better (note that this captures all hashing-based approaches). And although Chakrabarti et al. [13] and Barkol and Rabani [7] have applied the cell probe model [29] to the nearest neighbor problem, these lower bounds do not apply directly to the k-NN problem nor do they give any I/O or indexability lower bounds.
(2) If N is large and super-quadratic space is prohibitive, our result means that the queries will be slow in the worst case. However, our "bad dataset" is randomly drawn. We believe (as a result of our lower bound) that structured data will likely not suffer from this lower bound, and we leave exploiting the structure in a dataset to reduce the query time for future work.
(3) We leave the task of exploring the space-query tradeoff in other metric spaces (such as Hamming $\{0,1\}^d$) to future work.

The rest of the paper is organized as follows: In section 2 we define the external memory and indexability models, describe the concepts of an indexing scheme and the metrics that are used to analyze its performance, and define the set workload and the k-nearest neighbor problem. In section 3, we state our main result and prove that k-nearest neighbor problem, in both the Euclidean $\mathbb{R}^d$ and Hamming metric, is an instance of the set workload, described in section 2. In section 4, we state the implications of our results.

2 PRELIMINARIES

In this section we briefly describe the external memory model (due to Aggarwal and Vitter [1]), the indexability model (due to Hellerstein et al. [19], [20]), the set workload, and our problem.

Definition 2.1 (External Memory Model). In the external memory, or DAM (disk access machine) model, there is main memory of size M, and an infinite external memory, both realized as arrays. The data is stored in external memory, and is transferred to/from main memory for computation in I/Os or block transfers. If the block size is B, one I/O loads (writes) B consecutive elements in the array from (to) external memory to (from) main memory. The cost of an algorithm in the I/O model is described solely by the number of I/Os performed, and computation once data is inside the main memory is considered free.

All of the following material in this section is taken from Hellerstein et al. [19], [20].

Definition 2.2 (Workload). A workload W is a tuple $W = (D, I, Q)$, where D is a non-empty set (the *domain*), $I \subseteq D$ is a nonempty finite set (the *instance*, whose size we denote by N), and Q is a set of subsets of the instance I (the *query set*).

Definition 2.3 (Indexing Scheme). An indexing scheme is a pair $S = (W, \beta)$, where $W = (D, I, Q)$ is a workload and β is a set of B-subsets of I such that B is some positive integer and β covers I.

Basically, an indexing scheme is a way to lay out the data on the disk in blocks so as to answer queries efficiently. In the indexability model, once the query $q \in Q$ arrives, there is an oracle that tells the algorithm exactly which blocks on disk contain the elements in q for free (here we use q both for the query and for its "answers", as we don't have to search for them). Ideally, we want indexing schemes that are space-efficient and do not perform too many I/Os to bring in these $|q|$ elements from memory. This is captured by the following two performance parameters.

Definition 2.4 (Redundancy). The redundancy $r(x)$ of a data item $x \in I$ is defined as the number of blocks b in the block set β that contain x:

$$r(x) = |\{b \in \beta : x \in b\}|.$$

The redundancy r of the indexing scheme S is defined as the average of $r(x)$ over all objects x, i.e., $r = \frac{1}{N}\sum_{x \in I} r(x)$. If the space used by the indexing scheme is S blocks, it turns out that $S = rN/B$.

The redundancy ranges from 1 (with $S = N/B$ blocks, the minimum required to store the data), to $\binom{N}{B}/(N/B)$ (corresponding to $S = \binom{N}{B}$, where every B-combination of the input data is written out to a block).

Definition 2.5 (Cover set). A set of blocks $U \subseteq \beta$ *covers* a query $q \in Q$ iff $q \subseteq \bigcup_{b \in \beta} b$.Furthermore, a cover set, $C_q \subseteq \beta$ for a query $q \in Q$ is a minimum-size set of the blocks that cover Q.

Definition 2.6. The access overhead $A(q)$ of a query $q \in Q$ is defined as:

$$A(q) = \frac{C_q}{\lceil |q|/B \rceil}.$$

Definition 2.7 (Access Overhead). The access overhead A for an indexing scheme S is defined as $A = \max_{q \in Q} A(Q)$.

Given a query q, any indexing scheme must make at least $\lceil |q|/B \rceil$ I/Os to answer q. Intuitively, if the access overhead of an indexing scheme is A, then for some query q, the scheme requires $A\lceil |q|/B \rceil$ I/Os to answer q.

2.1 Set Workload

Definition 2.8. The λ-set workload $K_{n,\lambda}$ is a workload whose instance is $\{1, \cdots, n\}$,and whose query set is the set of all λ-subsets of the instance.

Clearly, if $\lambda \leq B$, the best indexing scheme will have access overhead 1, whereas the worst indexing scheme requires λ I/Os, giving an access overhead of $\lambda/\lceil \lambda/B \rceil = B$

Theorem 2.9. *[19, Theorem 7.2] For λ-set workload $K_{n,\lambda}(I, Q)$, $B \geq \lambda$, any indexing scheme with redundancy*

$$r < \frac{n - \lambda + 1}{(\lambda - 1)(B - 1)}$$

has worst possible access overhead $A = \lambda$.

The above theorem says that if an indexing scheme is to avoid the worst possible access overhead, it must use space at least quadratic in N/B.

2.2 k-Nearest Neighbor Problem

Definition 2.10. **k-Nearest Neighbor Problem**: Given a set $P = \{p_1, \cdots, p_n\}$ of n points, where p_i belongs to a metric space $M = (X, d)$, and a query q, return a set $K = \{t_1, \cdots, t_k\}$ of k points from P such that for any $p \in P \setminus K$ then $d(q, t_i) < d(q, p)$, for all $1 \leq i \leq k$

3 OUR RESULT: THE LOWER BOUND

In our main result we show that the k-nearest neighbor problem is an instance of the λ-set workload. We can thus use Theorem 2.9 to prove Theorem 3.1. To do so we construct a point set P containing N such that given any λ and any one of λ-subsets of P, we show that there exists a query q whose k-nearest neighbors are precisely those λ points. This would demonstrate that a query could potentially require the indexing scheme to bring in any λ subset of P, which means that the $k-NN$ problem is the same as the $K_{n,\lambda}$ set workload problem. This is stated formally as:

Theorem 3.1. *Let $d > 16(\lambda + 1)^2 \log n$. Then, a) there exists a set $P \subset \mathbb{R}^d$, $|P| = N$, such that for any subset $L \subset P$, $|L| = \lambda$, there exists a point $q \in \mathbb{R}^d$ such that $\forall \ell \in L$ and $\forall p \in P \setminus L$, $d(q, \ell) < d(q, p)$, where $d()$ denotes the Euclidean distance, and*

b) there exists a set $P \subset \{-1/\sqrt{d}, 1/\sqrt{d}\}^d$, $|P| = N$, such that for any subset $L \subset P$, $|L| = \lambda$, there exists a point $q \in \{-1/\sqrt{d}, 1/\sqrt{d}\}^d$ such that $\forall \ell \in L$ and $\forall p \in P \setminus L$, $d(q, \ell) < d(q, p)$, where $d()$ denotes the Hamming distance.

The rest of this section is devoted to the proof. We state the following lemma, which says that as long as the dimension d is large, there are vectors that are almost orthogonal with high probability. Here orthogonality is with respect to the usual inner product for vectors. This result seems to be be folklore but we provide a proof here for the sake of completeness.

Lemma 3.2. *a) Let P be a set of N vectors drawn uniformly from the surface of the d dimensional sphere S^{d-1}. Then*

$$Pr[\forall i, j, |\langle p_i, p_j \rangle| \leq \epsilon] \geq 1 - N^2 e^{-\epsilon^2 d/2}.$$

b) Let $P = \{p_1, \cdots, p_n\} \subset \{-1/\sqrt{d}, 1/\sqrt{d}\}^d$, where p_i is chosen randomly in such a way that each coordinate of p_i is either $-1/\sqrt{d}$ or $1/\sqrt{d}$ with equal probability. Then

$$Pr[\forall i, j, |\langle p_i, p_j \rangle| \leq \epsilon] \geq 1 - N^2 e^{-\epsilon^2 d/2}.$$

Proof. First we notice that the distribution of $\langle p_i, p_j \rangle$ where p_i and p_j are drawn uniformly (from either of the two metric spaces) is the same as that of $\langle v, p_i \rangle$, where v is a fixed vector.

For part a), define $C(v, \epsilon) = \{u \in S^{d-1} : \langle u, v \rangle \geq \epsilon\}$. Lemma 2.2 in [6] states that the normalized volume of $C(v, \epsilon)$ is at most $e^{-d\epsilon^2/2}$. This directly proves a).

To prove b), we apply Hoeffding's inequality [26, p. 77]. We abbreviate $p = p_i$ for now. Then $\langle v, p_i \rangle = \langle v, p \rangle = \sum_{i=1}^{d} v_i p_i$, where for all $1 \leq i \leq d$, p_i is randomly chosen between $-1/\sqrt{d}$ and $1/\sqrt{d}$. Thus we have that $\mathbb{E}\langle v, p \rangle = \sum_{i=1}^{d} v_i . \mathbb{E}[p_i]$, which equals zero since $\mathbb{E}[p_i] = 0$.

To apply Hoeffding's inequality, set $X_i = v_i p_i$. Note that $-1/d \leq X_i \leq 1/d$. Clearly, $X = \langle v, p \rangle = \sum_i X_i$. Thus we have that

$$\Pr\left[|\sum_i X_i| > \epsilon \right] < \exp\left(\frac{-2\epsilon^2}{2\sum_{i=1}^{d} (2/d)^2} \right) = e^{-\epsilon^2 d/2},$$

which proves part b) of the Lemma. □

We now show how to complete the proof of Theorem 3.1. Lemma 3.2 states that if vectors are chosen randomly, they are almost orthogonal with at least a certain probability. Set $\epsilon = 1/(2(\lambda + 1))$. Then one can check that if $d > 16(\lambda + 1)^2 \log n$ (which is a hypothesis of Theorem 1.1), then $1 - N^2 e^{-\epsilon^2 d/2} > 0$. Since this probability is strictly positive, this means there must exist a point set P^* (in both metric spaces) such that for any $p_i, p_j \in P^*$, $|\langle p_i, p_j \rangle| \leq 1/(2(\lambda + 1))$. This P^* is the instance of our workload for the λ-NN problem.

Now assume we are given a subset $L \subset P^*$ of size $|L| = \lambda$. We will prove part a) of Theorem 3.1, as the proof of part b) is essentially the same.

Given $L = \{\ell_1, \cdots, \ell_\lambda\}$, let $q = (1/\lambda) \sum_{i=1}^{\lambda} \ell_i$. We claim that for any $p \in P \setminus L$ and any $\ell_i \in L$, $d(q, p) > d(q, \ell_i)$. This would prove that the ℓ_is are indeed the λ-nearest neighbors of q in P, and since the choice of L was arbitrary, establishes that this is essentially the same as $K_{N,\lambda}$-set workload.

To prove $d(q, p) > d(q, \ell_i)$, notice that it suffices to prove that $\langle q, p \rangle < \langle q, \ell_i \rangle$, because that implies the following chain of inequalities (recall that $p, \ell_i \in S^{d-1}$)

$$\begin{aligned}
-2\langle q, p \rangle < -2\langle q, \ell_i \rangle \quad &\Rightarrow \quad \|q\|^2 + 1 - 2\langle q, p \rangle < \|q\|^2 + 1 - 2\langle q, \ell_i \rangle \\
&\Rightarrow \quad \|q\|^2 + \|p\|^2 - 2\langle q, p \rangle < \|q\|^2 + \|\ell_i\|^2 - 2\langle q, \ell_i \rangle \\
&\Rightarrow \quad \langle q - p, q - p \rangle < \langle q - \ell_i, q - \ell_i \rangle \\
&\Rightarrow \quad d(q, p) < d(q, \ell_i)
\end{aligned}$$

Lemma 3.3. *Let P^*, q, p, L and ℓ_i be as above. Then $\langle q, p \rangle \leq 1/2(\lambda + 1)$, and $\langle q, \ell_i \rangle > 1/\lambda - (\lambda - 1)/(\lambda(2(\lambda + 1)))$.*

Proof. $\langle q, p\rangle = \langle \sum_j \ell_j/\lambda, p\rangle = (1/\lambda)\sum_j \langle \ell_j, p\rangle$. Now because $p, \ell_j \in P^*$, and vectors in P^* have (absolute value of) pairwise inner product at most $1/2(\lambda+1)$, we get that $(1/\lambda)\sum_j \langle \ell_j, p\rangle \leq 1/2(\lambda+1)$.

Similarly $\langle q, \ell_i\rangle = (1/\lambda)\sum_j \langle \ell_j, \ell_i\rangle$. Now for any $1 \leq j \neq i \leq \lambda$, $\langle \ell_j, \ell_i\rangle > -1/2(\lambda+1)$. Hence we have that $(1/\lambda)\sum_j \langle \ell_j, \ell_i\rangle > (1/\lambda)(1-\sum_{j\neq i}\langle \ell_j, \ell_i\rangle)$, which is greater than $(1/\lambda) - (\lambda-1)/(\lambda(2(\lambda+1)))$. □

One can now check that $\langle q, \ell_i\rangle > 1/\lambda - (\lambda-1)/(\lambda(2(\lambda+1))) > 1/\lambda - 1/(2(\lambda+1)) > 1/\lambda - 1/2\lambda > 1/2(\lambda+1) > \langle q, p\rangle$, and this completes the proof of Theorem 1.1.

Combining with Theorem 2.9 about the redundancy-access overhead tradeoff for indexing schemes for set-workload, we get a redundancy-access overhead for indexing schemes for the λ-NN problem:

Theorem 3.4. *Consider any indexing scheme for the k-NN problem, where $k \leq B$. If the redundancy $r < (N-k+1)/((k-1)(B-1))$, then the indexing scheme has worst case access overhead $A = k$.*

If $k = \Theta(1)$, then the result says that in order to get access overhead less than k (meaning for k-NN queries to take fewer than k I/Os in the worst case), r would have to be $\Omega(N/B)$, implying that the space would have to be $\Omega((N/B)^2)$.

4 CONCLUSION

A viable and informative theory of indexability can be used to judge the efficacy of future external memory indices without the need of extensive experimentation on various data sets and query distributions. The work of Hellerstein et al. [19], [20] developed a framework that, though it omits consideration of algorithms needed to determine which blocks within the index cover a query as well as the storage and retrieval cost of the algorithm and auxiliary data structures, focuses on two measures, *storage redundancy* and *access overhead*, which are described exclusively in terms of a parameter, B, the block size. We apply the theory to an important problem,the k-nearest neighbor search, which is member of a larger class of proximity search problems, and show that we can get an instance of k-NN that is isomorphic to the λ-set workload. While our result shows that efficient indexing is in general not possible, it still leave open the hope that for structured data one may be able to get better indexing schemes. Another future work is to extend our results to other metric spaces.

5 ACKNOWLEDGEMENTS

The authors acknowledge support from NSF (Award ID 1755791). The authors would also like to thank Rasmus Pagh from IT University of Copenhagen for all his thought provoking discussions.

KEYWORDS

indexing, nearest neighbours search

REFERENCES

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, pages 1116–1127.
[2] Z. Allen Zhu, Y. Yuan, and K. Sridharan. Exploiting the structure: Stochastic gradient methods using raw clusters. *CoRR*, abs/1602.02151, 2016.
[3] A. Andoni. *Nearest neighbor search: the old, the new, and the impossible.* PhD thesis, Massachusetts Institute of Technology, 2009.
[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 459–468, Washington, DC, USA, 2006. IEEE Computer Society.
[5] A. Andoni, T. Laarhoven, I. P. Razenshteyn, and E. Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 47–66, 2017.
[6] K. Ball et al. An elementary introduction to modern convex geometry. *Flavors of geometry*, 31:1–58, 1997.
[7] O. Barkol and Y. Rabani. Tighter lower bounds for nearest neighbor search and related problems in the cell probe model. *J. Comput. Syst. Sci.*, 64(4):873–896, June 2002.
[8] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):266–278, Feb. 2011.
[9] M. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok. Don't trash: How to cache your hash on flash. In *The Proceedings of the VLDB Endowment (PVLDB)*, pages 1627–1637, 2012.
[10] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, 1997.
[11] P. Berkhin. A survey of clustering data mining techniques. In *Grouping Multidimensional Data - Recent Advances in Clustering*, pages 25–71. 2006.
[12] A. Broder, M. Mitzenmacher, and A. B. I. M. Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.
[13] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 473–482, Washington, DC, USA, 2004. IEEE Computer Society.
[14] E. F. Codd. A relational model of data for large shared data banks. In *Communications of ACM*, volume 13, pages 377–387. ACM, 2000.
[15] D. Cromer. Ubiquitous b-tree. *ACM Computing Surveys*, 11:121–137, 1979.
[16] I. S. Dhillon, P. Ravikumar, and A. Tewari. Nearest neighbor based greedy coordinate descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2160–2168, USA, 2011. Curran Associates Inc.
[17] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30, 1998.
[18] J. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry.* CRC Press, 2 edition, 2004.
[19] J. Hellerstein, E. Koutsoupias, D. Miranker, C. Papadimitriou, and S. Vasilis. On a model of indexability and its bounds for range queries. *Journal of the ACM*, 49(1):35–55, 2002.
[20] J. Hellerstein, E. Koutsoupias, and C. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 249–256. ACM, 1997.
[21] J. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search tree for database systems. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 562–573. ACM, 1995.
[22] T. Hofmann, A. Lucchi, and B. McWilliams. Neighborhood watch: Stochastic gradient descent with neighbors. *CoRR*, abs/1506.03662, 2015.
[23] P. Indyk and R. Motwani. Approximate near neighbors: Towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.
[24] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30:364–397, 2005.
[25] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. ACM, 2007.
[26] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.
[27] T. Seidl and H.-P. Krieger. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD international*, pages 154–165. ACM, 1998.
[28] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing).* The MIT Press, 2006.
[29] A. C.-C. Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, July 1981.

A Greedy Linear-time Algorithm for the Altitude Terrain Guarding Problem

Ovidiu Daescu, Hemant Malik
Department of Computer Science, University of Texas at Dallas
{daescu, malik}@utdallas.edu

Christiane Schmidt
Communications and Transport Systems, ITN, Linköping University, Norrköping, Sweden
christiane.schmidt@liu.se

Abstract

We present a simple linear-time greedy algorithm for the problem of guarding an x-monotone terrain with the minimum number of guards from an altitude line above the terrain.

Keywords and phrases Altitude Terrain Guarding Problem

1 Introduction

A *terrain* T is an x-monotone chain of line segments in $\mathbb{R}^2$ defined by its *vertices* $V(T) = \{v_1, \ldots, v_n\}$ and *edges* $E(T) = \{e_1, \ldots, e_{n-1}\}$, with $e_i = \overline{v_i v_{i+1}}$.

In the 1.5D Terrain Guarding Problem (TGP), we are given a terrain T, on which we have to place a minimum number of point-shaped guards, such that they cover T, i.e., each point on T is visible by at least one of the guards. King and Krohn [1] proved the problem to be NP-hard. Friedrichs et al. [2] considered the problem of guarding a terrain with guards placed on a horizontal line $\mathcal{A}$ above the terrain, the ATGP$(T, \mathcal{A})$: given a terrain T and an altitude line $\mathcal{A} = ab$, find an optimal guard set $G \subset \mathcal{A}$ w.r.t. ATGP$(T, \mathcal{A})$. Let a' and b' be the leftmost and rightmost vertices of T, as shown in Figure 1. Joining a with a' and b with b' results in a simple, uni-monotone polygon P (an x-monotone polygon with a single horizontal segment as one of its two chains). Let $\mathcal{V}_P(p)$ be the *visibility polygon* (VP) of a point p in P, with $\mathcal{V}_P(p) := \{q \in P \mid p \text{ sees } q\}$. For $G \subset P$ we abbreviate $\mathcal{V}_P(G) := \bigcup_{g \in G} \mathcal{V}_P(g)$. A guard set $G \subset \mathcal{A}$ is optimal, if G is feasible (that is, $T \subseteq \mathcal{V}_T(G)$) and $|G| = \text{OPT}(T, \mathcal{A}) := \min\{|C| \mid C \subseteq \mathcal{A} \text{ is feasible w.r.t. ATGP}(T, \mathcal{A})\}$. Friedrichs et al. [2] proved that uni-monotone polygons are perfect, showed that the Altitude Terrain Guarding Problem (ATGP) and the problem of guarding uni-monotone polygons are equivalent, and presented an $O(n^2 \log n)$ time algorithm for both. In his PhD Dissertation [3] Bengt Nilsson presented a linear time algorithm to compute an optimal set of vision points on a watchman route in a *walkable polygon*, a special type of simple polygon that encompasses spiral and monotone polygons. Being developed for a more general type of polygon, rather than a uni-modal polygon, Bengt's algorithm is non-trivial and its proof of correctness and optimality is complex. In contrast, the algorithm in [2] is simple and elegant, however it suffers from a suboptimal running time. In this paper we make some observations on the visibility characterizations in [2] that allow us to obtain a simple, greedy, linear-time algorithm.

2 Linear-time Algorithm

For a point v on T, we define the right intercept, r_v, and the left intercept, l_v, as the rightmost and leftmost point on $\mathcal{A}$ in $\mathcal{V}_P(v)$, respectively. Equivalently, for a line segment s on T, we define the right intercept, r_s, and the left intercept, l_s, as the right and left intersection of

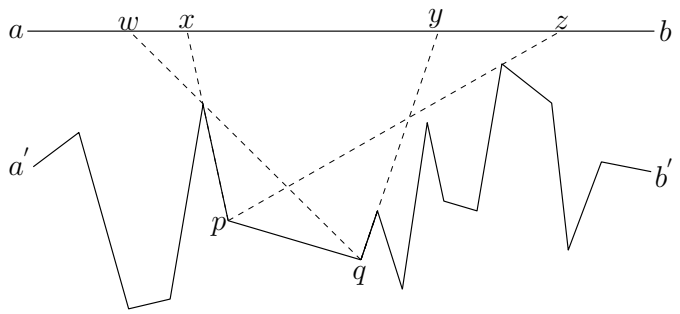


Figure 1 x-monotone chain from $a^{'}$ to $b^{'}$ and line segment ab at height h of Terrain T. Left and right intercept of points p, q and line segment pq

the strong visibility polygon of s and $\mathcal{A}$, respectively. For an example, consider Figure 1: x and z are the left and right intercept of point p, resp., and w and y are the left and right intercept of point q, resp. For the edge pq, x and y are the left and right intercept, resp. If we move along $\mathcal{A}$, from a to b, pq becomes partially visible at w while z is the last point from which pq is partially visible. The segment is completely visible for any point on $\mathcal{A}$ between x and y. Notice that $l_{pq} = l_p$ and $r_{pq} = r_q$. For $t, t^{'} \in \mathcal{A}$, we write $t \leq t^{'}$ $(t < t^{'})$ if t is (strictly) left of $t^{'}$, i.e., has a (strictly) smaller x-coordinate; we say that t lies before $t^{'}$, and $t^{'}$ lies after t on $\mathcal{A}$.

We first compute the shortest paths from each of a and b to the vertices of T, which can be done in $O(n)$ time [4]. For a point $v \in T$, let $P_{v,a}$ and $P_{v,b}$ be the shortest paths from v to a and v to b, respectively. Note that these shortest paths consist of convex chains of total complexity $O(n)$. To find the right intercept of point v, we can extend the first segment of $P_{v,b}$ and find its intersection with $\mathcal{A}$. To find the left intercept of point v, we can extend the first segment of $P_{v,a}$ and find its intersection with $\mathcal{A}$ (see Figure 1 and Figure 2). Similarly, we can find the left and right intercept of a line segment $s \in T$.

Our algorithm proceeds in a greedy fashion, placing guards on $\mathcal{A}$ in order, from a to b. Let $g_1, g_2, \ldots, g_i$ be the guards placed so far. As discussed in [2], all edges that lie to the left of the last placed guard, g_i, and the edge vertically below g_i, are visible by the guards placed so far. Thus, after placing g_i, we need be concerned with the edges to the right of g_i.

Let $e = pq$ be an edge of T that lies to the right of g_i. Then pq is either (a) visible from g_i, (b) not visible from g_i (no point of pq is visible from g_i) or (c) partially visible from g_i, in which case g_i sees a sub-segment $p'q$ of pq. An easy observation from [4, 2] is that none of the guards preceding g_i on $\mathcal{A}$ can see any point of pp'; that portion of pp' can only be seen by a guard placed to the right of g_i.

Another observation from [2] is that the guards forming the optimal set must be placed at well defined points on $\mathcal{A}$, each of which corresponds to a right intercept, r_v, where v is either a vertex of T or otherwise it corresponds to some point on a partially visible edge, as described earlier. This implies that, starting from g_i, the next guard will be placed at the leftmost right intercept r^l on $\mathcal{A}$, among those generated by the edges to the right of g_i. Once we reach an edge vertically below r^l we place g_{i+1} at r^l and repeat the process.

Note that to achieve linear time we cannot afford to keep the right intercepts in sorted order (see [5]). Instead, it is enough to keep track of the leftmost right intercept corresponding to the edges of T, including those generated by partially visible edges, following g_i.

▶ **Observation 1.** *After placing g_{i+1} all edges of T between g_i and g_{i+1} are visible by the guards $g_1, g_2, \ldots, g_{i+1}$.*

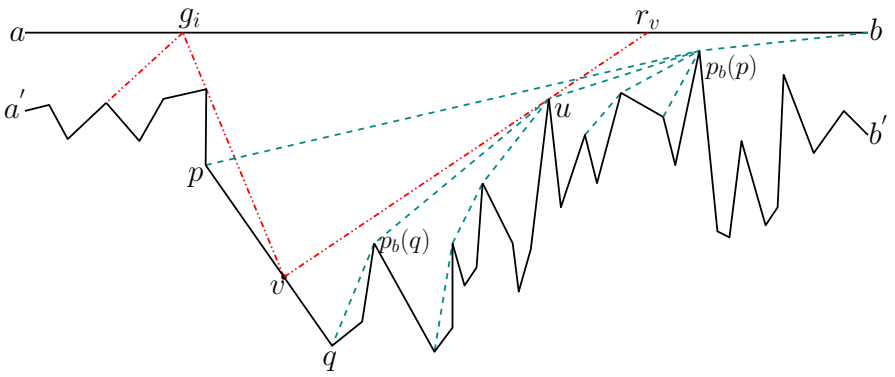


Figure 2 Line segment pq is partially seen by guard g_i. A portion of the shortest path tree originating from b is shown with dashed lines (cyan).

It follows from Observation 1 that after placing g_{i+1} we don't need to be concerned with the right intercepts of the edges of T between g_i and g_{i+1}.

For a segment s of T, we define x_s^l as the x-coordinate of the leftmost endpoint of s and x_s^r as the x-coordinate of the rightmost endpoint of s.

We now describe our algorithm in more details. Observe that all edges to the left of the first guard g_1 must be fully seen by g_1. To place g_1, we traverse the edges of T in order, starting with e_1. For each edge visited, we mark it as visible, compute its right intercept on $\mathcal{A}$, and keep track only of the leftmost such intercept, r^l. Once we reach an edge $e \in T$ such that $x_e^l \leq r^l < x_e^r$ we stop, mark e as visible, and place g_1 at r^l. We then repeat the following inductive process. Assume guard g_i has been placed. We start with the first edge of T to the right of g_i and check if the edge is visible, not visible, or partially visible from g_i. Let e_k be the current edge. If e_k is visible then we mark it as such. If e_k is not visible then we compute its right intercept on $\mathcal{A}$ while keeping track of the leftmost right intercept, r^l, following g_i on $\mathcal{A}$. If e_k is partially visible, let e'_k be the segment of e_k not visible from g_i and let q' be the right endpoint of e'_k; we compute the right intercept of q' on $\mathcal{A}$ while keeping track of r^l. Once we reach and edge $e \in T$ such that $x_e^l \leq r^l < x_e^r$ we stop, mark e as visible, and place g_{i+1} at r^l. The proof that this greedy placement results in an optimal set of guards has been given in [2].

▶ **Lemma 1.** *Given an edge $e = pq$ of T and a point $v \in e$, the right intercept r_v of v can be found in $O(1)$ amortized time. A similar claim holds for the left intercept of v.*

Proof. We make the proof for the right intercept (for the left one it is similar).

The shortest path from a and b to each vertex in T can be found in $O(n)$ time [2] and is available in the resulting shortest path tree. These shortest paths consist of convex chains. Let T_a and T_b be the shortest path trees originating from a and b, respectively. Both T_a and T_b have $O(n)$ vertices and edges. Let T_b^u be the subtree of T_b rooted at vertex u.

Let $p_b(u)$ denote the parent of vertex u in T_b. Obviously, if v is an end vertex of e the right intercept of v is available in constant time from T_b, as the intersection of the line through p and $p_b(v)$ with $\mathcal{A}$. Assume v is interior to e.

To find the right intercept of v, we need to find the **first vertex** u of T_b on the shortest path, $P_{v,b}$, from v to b; the intersection of the extension of vu and $\mathcal{A}$ corresponds to r_v. Note that vu is tangent to a convex chain of T_b at point u, specifically the chain capturing the shortest path from q to b in T_b. Hence, we can find r_v by finding the tangent from v to that

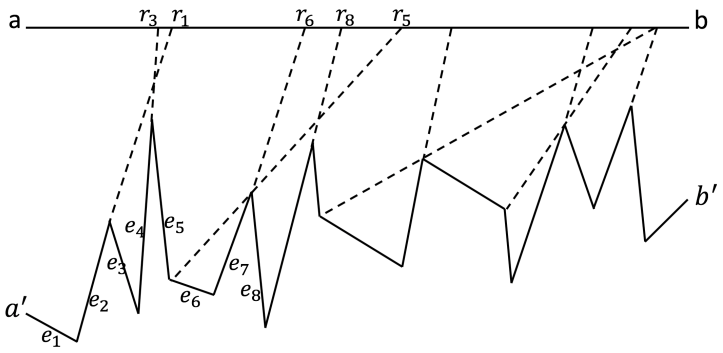


Figure 3 Terrain T with line segments $e_1, \ldots e_n$ and right intercept of each line segment

convex chain, by traversing the chain starting at q. Moreover, the vertex u is located on the portion of the chain from q to $p_b(p)$. Due to the structure of the shortest paths, it is an easy observation that this subchain of T_b will not be revisited while treating an edge of T to the right of e (see Figure 2). Since the total complexity of the convex chains is $O(n)$ it follows that over all edges of T we find r_v in amortized $O(1)$ time. ◀

For an example, see Figure 3. We start with e_1 and store r_1 (right intercept of e_1) as $\mathrm{r}^{'}$. We move to the next line segment, e_2, and $r_{e_1} = r_{e_2}$. Because $r_{e_3} < r_{e_1}$, we update $\mathrm{r}^{'} = r_{e_3}$. We move to the e_4 and $r_{e_3} = r_{e_4}$. For e_5, $r_5 > \mathrm{r}^{'}$, hence, no update is necessary. Moreover, $x^l_{e_5} \leq \mathrm{r}^{'} \leq x^r_{e_5}$. Hence, we place the first guard at r_{e_3} and set $\mathrm{r}^{'} = r_{e_6}$.

The algorithm visits each edge e of T only once, and the total time spent while visiting a line segment can be split into the following steps:

1. The time taken to decide the visibility of e by the last placed guard.
2. The time to find the partially visible segment of e, if needed.
3. The time to find the right intercept of a point v on edge e.
4. The time to compare r_e or r_v with $\mathrm{r}^{'}$.

Since we know the location of the last guard on $\mathcal{A}$ the first step takes constant time. The second step and the third step take $O(1)$ amortized time (see proof of Lemma 1). The last step takes constant time. Hence, the total running time of the algorithm is $O(n)$.

▶ **Theorem 2.** *The algorithm presented solves the* ATGP$(T, \mathcal{A})$ *problem in* $O(n)$ *time.*

References

1 J. King, E. Krohn, Terrain guarding is NP-hard, SIAM Journal on Computing 40 (5) (2011) 1316–1339.
2 S. Friedrichs, V. Polishchuk, C. Schmidt, Altitude terrain guarding and guarding unimonotone polygons, in: Proceedings of the 34th European Workshop on Computational Geometry, 2018.
3 B. Nilsson, Guarding art galleries; methods for mobile guards, Ph. D. thesis, Lund University.
4 D. Avis, G. T. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, IEEE Trans. Computers 30 (12) (1981) 910–914.
5 D. Z. Chen, O. Daescu, Maintaining visibility of a polygon with a moving point of view, Inf. Process. Lett. 65 (5) (1998) 269–275.

An Improved Lower Bound for Non-Trivial Reach

Hugo A. Akitaya* David Stalfa† Csaba D. Tóth*‡

1 Problem Statement

Let $S = \{s_1, \ldots, s_n\}$ be a set of points, or *anchors*, in the unit square $U = [0,1]^2$. We say that a square q is empty if $q \subset U$ and no point in S is interior to q. For $i = 1, \ldots, n$, let q_i^1 be the maximal, axis-aligned empty square whose lower-left corner is s_i. Define q_i^2, q_i^3, and q_i^4 similarly with s_i in the upper-left, upper-right, and lower-right corners, respectively We call such a square q_i^j an *anchored square*.

Definition 1. *For a given set $S = \{s_1, \ldots, s_n\}$, the* reach *of S is $R(S) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{4} q_i^j$.*

In [1], we showed that $\text{area}(R(S)) \geq \frac{1}{2}$ for all S. This lower bound has applications in approximating the maximal area of an anchored square packing [2], where the reach is the union of all possible anchored square packings. In this note, we improve the lower bound on $\text{area}(R(S))$ for some of the most interesting instances with a closer analysis of the techniques developed in [1].

2 Previous Results

In [1], we distinguish between *trivial* and *non-trivial* instances of a point set S. A *trivial* instance is a point set whose reach is disjoint from some side of U. For trivial instances, the lower bound given in [1] is tight: there is a trivial instance S for which $\text{area}(R(S)) = \frac{1}{2}$ (see Fig. 1(a)). For non-trivial instances, where the reach touches all four sides of U, the problem of finding a tight lower bound remains open. Figure 1(b) and (c) show non-trivial instances of the smallest area we have found. Both yield a reach of area $\frac{4}{7}$. In the remainder of this note, we assume that S is a non-trivial instance.

In establishing the lower bound in [1], we assume that no two anchors lie on an axis parallel line, and that no anchor lies on ∂U. This assumption is justified by the following lemma.

Lemma 2. *([1, Lemma 5]). If $\text{area}(R(S)) \geq \frac{1}{2}$ for every finite point set $S \subset U$ such that $S \subset \text{int}(U)$ and no two points in S have the same x- or y-coordinates, then $\text{area}(R(S)) \geq \frac{1}{2}$ for every finite $S \subset U$.*

We also make use of the notion of a gap, or a region disjoint from $R(S)$.

Definition 3. *A* gap *is a maximal connected region C such that $C \subset U \setminus R(S)$.*

In [1, Lemma 7], we give a complete classification of the gaps (in non-trivial instances) into five types: Every gap is either a rectangle adjacent to the side of the square U, or the union of two such rectangles. Using this classification, we define a charging scheme to pick out a *charged region* R_C around each gap C. We show that for every gap C: (i) $R_C \subseteq R(S)$, (ii) $\text{area}(R_C) \geq \text{area}(C)$, and (iii) for any two gaps $C' \neq C$, R_C and $R_{C'}$ are interior disjoint. These properties of the charged regions yield the first main result of [1].

Theorem 4. *([1, Theorem 12]). For every finite set $S \subset U$, $\text{area}(R(S)) \geq \frac{1}{2}$.*

Corollary 5. *Suppose $R(S)$ is non-trivial. If there is some region W such that $W \subseteq R(S)$ and, for all C, W is interior disjoint from R_C, then $\text{area}(R(S)) \geq \frac{1}{2} + \text{area}(W)$.*

*Tufts University, Medford, MA, USA. Email: hugo.alves_akitaya@tufts.edu
†Northeastern University, Boston, MA, USA. Email: stalfa@ccis.neu.edu
‡California State University Northridge, Los Angeles, CA, USA. Email: csaba.toth@csun.edu

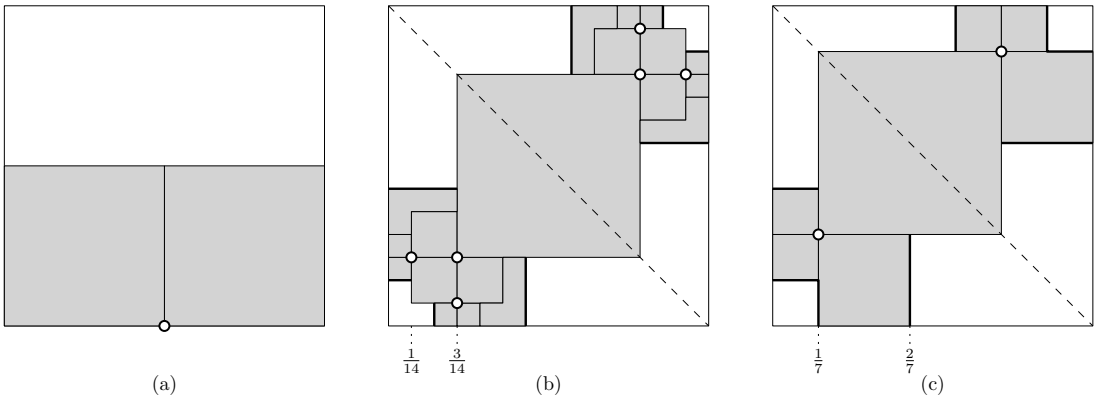


Figure 1: (a) A trivial reach with area $\frac{1}{2}$. (b) A non-trivial reach with area $\frac{4}{7}$. The anchors in the lower left have coordinates $(\frac{1}{14}, \frac{3}{14})$, $(\frac{3}{14}, \frac{1}{14})$, and $(\frac{3}{14}+\epsilon, \frac{3}{14}+\epsilon)$. (c) A non-trivial reach with area $\frac{4}{7}$.

3 Result: Improved Lower Bound for Non-Trivial Instances

Following Corollary 5, we define a region $W \subseteq R(S)$ that is disjoint from all charged regions. We prove a lower bound on the area of W, and so improve the lower bound on area$(R(S))$ for non-trivial instances.

Definition 6. *For a gap C, a line segment ℓ is a* leg *of C if $\ell \subseteq \partial C$, $\ell \cap \partial U$ is a single point, and $\ell \subset \partial q$ for some anchored square q. See the bold edges in Figure 1(b,c) for examples.*

Lemma 7. *Let C be a gap with leg ℓ. There are regions N_ℓ and $W_\ell \subseteq N_\ell$ with the following properties:*

1. *$N_\ell \subseteq R(S)$ and is an empty sqaure,*
2. *for all ℓ_1, ℓ_2 distinct from ℓ and each other, N_ℓ, N_{ℓ_1}, and N_{ℓ_2} are three-way interior disjoint,*
3. *for all ℓ' intersecting a side of U different than ℓ, W_ℓ and $W_{\ell'}$ are interior disjoint, and*
4. *for all C, W_ℓ and R_C are interior disjoint.*

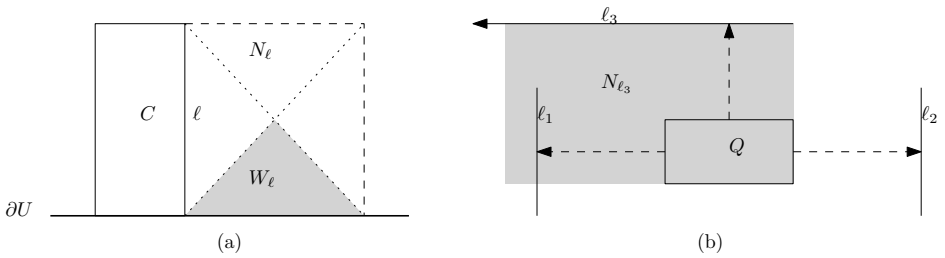


Figure 2: (a) Definitions of N_ℓ and W_ℓ. (b) If N_{ℓ_1}, N_{ℓ_2}, and N_{ℓ_3} intersect, then one of ℓ_1, ℓ_2, or ℓ_3 intersects $R(S)$.

Proof. Define N_ℓ as the square interior disjoint from C whose side is ℓ. Define W_ℓ as the triangle with one vertex at the center of N_ℓ and one side $N_\ell \cap \partial U$ (see Fig. 2(a)). We proceed to show that N_ℓ and W_ℓ have the properties listed in the lemma.

1. By characterization of the gaps in [1], $\ell \subseteq \partial q$ for some empty $q \subseteq R(S)$ anchored at some $s_i \in S$.

2. Suppose ℓ_1, ℓ_2, and ℓ_3 are legs such that N_{ℓ_1}, N_{ℓ_2}, and N_{ℓ_3} share some interior point. Let the intersection of the three squares be the rectangle Q. Three sides Q_1, Q_2, Q_3 of Q must have the following property: if r is a ray orthogonal to Q_i and disjoint from the interior of Q, with r intersecting Q_i at a single, interior point, then r intersects and is orthogonal to exactly one of ℓ_1, ℓ_2, or ℓ_3. In this case, we say that Q_i *projects* to the ℓ_j it intersects.

Let Q_1 project to ℓ_1, Q_2 to ℓ_2, and Q_3 to ℓ_3. Without loss of generality, let Q_1 and Q_2 be parallel, and let them be orthogonal to the x-axis with Q_1 to the left. The remaining leg, ℓ_3, must extend to either the left or right edge of U. Suppose ℓ_3 extends to the left. Then the intersection of ℓ_3 and ∂U is left of ℓ_1. But in this case, N_{ℓ_3} intersects with ℓ_1 (see Fig. 2(b)). This contradicts the assumption that ℓ_1 is a leg.

3. This follows from the fact that the sides of W_ℓ interior to U are defined along lines with slope ± 1.

4. Suppose, for some C, R_C and W_ℓ share an interior point. By definition of the charging scheme in [1] (see Fig. 3) and W_ℓ, W_ℓ cannot be a subset of R_C. So W_ℓ must intersect some edge of R_C. If W_ℓ intersects $R_C \cap C$, W_ℓ intersects C. So W_ℓ intersects some edge of R_C that shares at most one point with C. Because these edges of R_C and the edges of W_ℓ interior to U have slope ± 1, W_ℓ contains some vertex $v \notin \partial C$.

   Let an edge e of R_C be a *lead edge* if e intersects ∂U and is a subset of some line segment $d \subseteq R(S)$ such that one endpoint of d lies at $C \cap \partial U$, the other is at some anchor, and d has slope ± 1. By definition, all edges of R_C that intersect ∂U, and are not co-linear with some edge of C, are leading edges. Let a vertex $v \in R_C$ be a *lead* if v is an endpoint of some leading edge and $v \notin \partial U$. If W_ℓ contains a lead in its interior, then W_ℓ either intersects C or contains some anchor.

   Suppose W_ℓ contains some non-lead vertex $v \notin \partial C$ of R_C (and contains no lead and does not intersect C). Then, by definition of the charging scheme and the characterization of the gaps in [1], C is adjacent to the top edge of U, and v is the lowest point of R_C. Let T be the largest isosceles triangle in R_C that contains v and has a vertical axis of symmetry. By the characterization of the gaps in [1], T lies in the top quadrant of some square $q \subseteq R(S)$. So either C' intersects N_ℓ or C intersects $R_{C'}$.

□

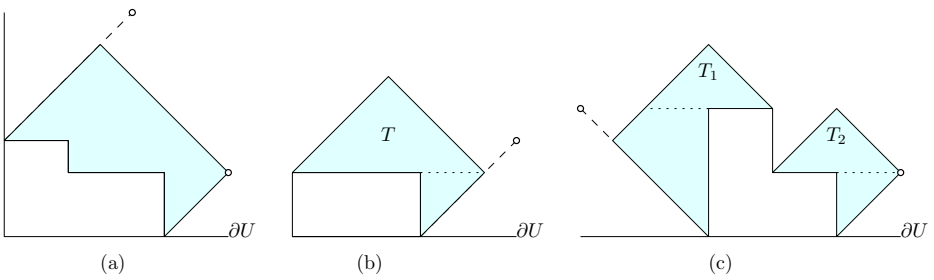


Figure 3: (a) A charged region for an L-shaped gap in a corner. Both vertices of R_C not in ∂C are leads. (b) A charged region for a rectangular gap (on a side or in a corner). Only the right-most vertex of R_C is a lead. (c) A charged region for an L-shaped gap on a side. The left- and right-most vertices of R_C are both leads.

Lemma 8. *Let p be the lower left corner of U. If $p \in R(S)$ for some finite point set $S \subset U$, then there is a set S' such that $S \subset S'$ and* $\text{area}(R(S')) < \text{area}(R(S))$.

Proof. Suppose $p \in R(S)$. Then p lies on the boundary of some anchored square q anchored at some s_i. By Lemma 2, we can assume that $s_i = (a, a)$ lies on the diagonal of q from p (see Fig. 4(a)). Let $S' = S \cup \{(\frac{a}{2}, \frac{a}{2}), (\frac{a}{3}, \frac{a}{6})\}$ (see Fig. 4(b)). Then $S \subset S'$ and $\text{area}(R(S')) < \text{area}(R(S))$, as claimed. □

Corollary 9. *There are at least four gaps in the minimal reach.*

Proof. By Lemma 3, all four corners of U are outside $R(S)$. By assumption, the reach is non-trivial, and so must touch all four sides. Therefore, there are at least four gaps. □

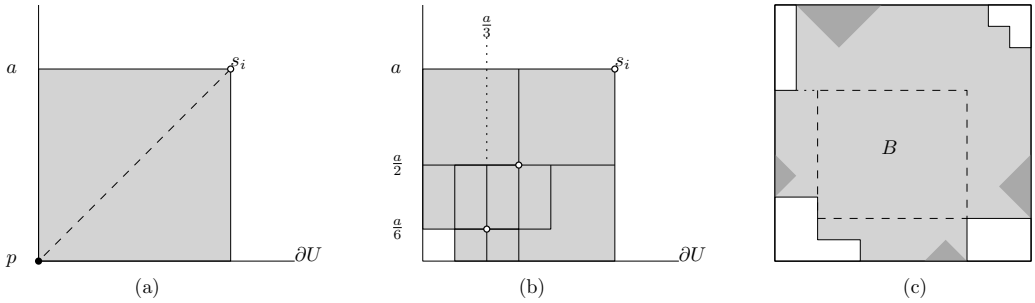


Figure 4: (a) If a corner p of U is in $R(S)$, then there is some anchor s_i diagonal from p. (b) By inserting anchors, $R(S)$ decreases; consequently, if $R(S)$ has minimal area, the corner of U is not in $R(S)$. (c) A central rectangle B with four W_ℓ's shown in dark gray.

Assume that $R(S)$ is non-trivial for a finite point set $S \subset U$. By Corollary 9, there is at least one leg intersecting each side. Let $\ell_1, \ldots, \ell_4$, resp., be a longest leg in $R(S)$ intersecting the bottom, top, right, and left edge of U, respectively. Let $\lambda_i = |\ell_i|$ and $\lambda = \sum_{i=1}^4 \lambda_i$.

Lemma 10. *If $R(S)$ is nontrivial, then* $\text{area}(R(S)) \geq \max\left(1 - \lambda + \frac{1}{4}\lambda^2, \frac{1}{2} + \frac{1}{16}\lambda^2\right)$.

Proof. Since $\ell_1, \ldots, \ell_4$ are the longest legs on each side, we have that a rectangle B defined by sides of length $(1 - \lambda_1 - \lambda_2)$ and $(1 - \lambda_3 - \lambda_4)$ is a subset of $R(S)$. By Lemma 7, $R(S)$ also contains squares $N_{\ell_1}, \ldots, N_{\ell_4}$ with respective side lengths $\lambda_1, \ldots, \lambda_4$ such that each N_{ℓ_i} is interior disjoint from B, and no point is in more than two such N_{ℓ_i} (see Fig. 4(c)). So $\text{area}(R(S))$ is at least

$$\text{area}(B) + \frac{1}{2}\sum_{i=1}^4 \text{area}(N_{\ell_i}) \geq (1 - \lambda_1 - \lambda_2)(1 - \lambda_3 - \lambda_4) + \frac{1}{2}\sum_{i=1}^4 \lambda_i^2 \geq 1 - \lambda + \frac{1}{4}\left(\sum_{i=1}^4 \lambda_i\right)^2 \geq 1 - \lambda + \frac{1}{4}\lambda^2$$

by Jensen's inequality.

Let $W = \bigcup W_{\ell_i}$ for $i = 1, \ldots, 4$. By Corollary 5 and Lemma 7, $\text{area}(R(S)) \geq \frac{1}{2} + \text{area}(W)$. By Lemma 7, the regions W_{ℓ_i}, $i = 1, 2, 3, 4$, are pairwise disjoint. So $\text{area}(R(S))$ is at least

$$\frac{1}{2} + \sum_{i=1}^4 \text{area}(W_{\ell_i}) \geq \frac{1}{2} + \frac{1}{4}\sum_{i=1}^4 \lambda_i^2 \geq \frac{1}{2} + \frac{1}{16}\left(\sum_{i=1}^4 \lambda_i\right)^2 \geq \frac{1}{2} + \frac{1}{16}\lambda^2$$

by Jensen's inequality. □

Theorem 11. *Let $R(S)$ be a non-trivial reach. Then* $\text{area}(R(S)) \geq \frac{1}{9}(11 - 2\sqrt{10}) \approx 0.51949\ldots$.

Proof. By Lemma 10, we have

$$\text{area}(R(S)) \geq \min_{\lambda > 0} \max\left(1 - \lambda + \frac{1}{4}\lambda^2, \frac{1}{2} + \frac{1}{16}\lambda^2\right).$$

By solving $1 - \lambda + \frac{1}{4}\lambda^2 = \frac{1}{2} + \frac{1}{16}\lambda^2$, the minimum is attained at $\lambda = \frac{1}{3}(8 - 2\sqrt{10})$, and it is $\frac{1}{9}(11 - 2\sqrt{10})$. □

4 Directions for Future Work

This result improves the lower bound proved in [1] for non-trivial instances. The problem of finding a tight lower bound on $\text{area}(R(S))$ for arbitrary non-trivial instances remains open. One direction for improvement is to find some S that yields a non-trivial reach with area less than $4/7$. However, based on our construction of two instances with area $4/7$, we conjecture that $4/7$ is tight. One strategy for proving a tight lower bound might be to show that some constant k anchors is sufficient to construct a reach of minimal area, and then find the optimal arrangement of k anchors.

References

[1] Hugo A. Akitaya, Matthew D. Jones, David Stalfa, and Csaba D. Tóth. Maximum Area Axis-Aligned Square Packings. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 77:1–77:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2018.77.

[2] Kevin Balas, Adrian Dumitrescu, and Csaba D. Tóth. Anchored rectangle and square packings. *Discrete Optimization*, 26:131–162, 2017. doi:10.1016/j.disopt.2017.08.003.

On the Hardness of Some Geometric Optimization Problems with Rectangles

Joseph S B Mitchell[1], Supantha Pandit[2]

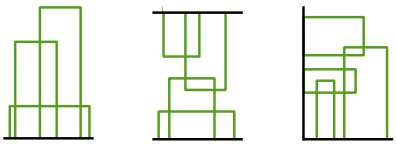
Stony Brook University, Stony Brook, New York, USA
joseph.mitchell@stonybrook.edu, pantha.pandit@gmail.com

Abstract

We study Set Cover, Hitting Set, Piercing Set, Independent Set, Dominating Set problems, and discrete versions (Discrete Independent Set and Discrete Dominating Set) for geometric instances in the plane. We focus on certain restricted classes of geometric objects, including axis-parallel lines, strips, and rectangles. For rectangles, we consider the cases in which the rectangles are (i) anchored on a horizontal line, (ii) anchored on two lines (either two parallel lines or one vertical and one horizontal line), and (iii) stabbed by a horizontal line. Some versions of these problems have been studied previously; we focus here on the open cases, for which no complexity results were known. In particular, we show that the Discrete Dominating Set and Discrete Independent Set problems are NP-hard for rectangles anchored on two parallel lines and rectangles stabbed by a horizontal line.

1. Introduction

We study several special cases of various optimization problems in the plane, including the Set Cover (SC), Hitting Set (HS), Piercing Set (PS), Independent Set (IS), and Dominating Set (DS) problems. In addition, we consider ***discrete*** versions of the IS and DS problems, the Discrete Independent Set (DIS) and Discrete Dominating Set (DDS) problems. The inputs of these two problems are a set of objects O and a set of points P. In the DIS problem the objective is to select minimum cardinality subset $O' \subseteq O$ of objects such that any two objects in O' do not share a point in P. On the other hand, in the DDS problem the objective is to select a minimum collection $O' \subseteq O$ of objects such that the intersection of any object in $O \setminus O'$ and an object in O' contains a point in P.



(a) (b) (c)

Figure 1: (a) Rectangles anchored on a horizontal line. (b) Rectangles anchored on two lines. (c) Rectangles anchored on two orthogonal lines.

In this paper we study various optimization problems on various types of geometric objects as follows (see Figure 1).
LINE: Axis parallel lines.
STRIP: Axis-parallel strips.
R-AHL: Rectangles anchored on a horizontal line.
R-ATL: Rectangles anchored on two lines.
R-ATOL: Rectangles anchored on two orthogonal lines.
R-SHL: Rectangles stabbing a horizontal line.

[1]Support from the National Science Foundation (CCF-1526406) and the US-Israel Binational Science Foundation (project 2016116).
[2]Partially supported by the Indo-US Science & Technology Forum (IUSSTF) under the SERB Indo-US Postdoctoral Fellowship scheme with grant number 2017/94, Department of Science and Technology, Government of India.

1.1. Our Contributions

We list our contributions as follows.

Problems	IS	SC	HS	PS	DS	DDS	DIS
LINE	P	P	P[1]	P	P	***H***	***P***
STRIP	P	H[2]	H[2]	P	P	***H***	H[2]
R-AHL	P	P[3, 2]	P[3, 2]	P	P	??	P[2]
R-ATL	P [4, 5, 6, 7]	H[8]	H[8]	??	??	***H*** Th 1	***H*** Th 3
R-ATOL	P [4, 5, 6, 7]	H	H	??	??	***H***	H
R-SHL	P	***H***	***H***	P	P	***H*** Th 2	***H*** Th 4

Table 1: Our contributions are shown in colored text (P-> polynomial time. H-> NP-hard). In this extended abstract, we only present a partial set of the results. The results in non-colored text for which no references are given are either trivial to show or can be derived from the other problems easily.

1.2. Prerequisites

The **rectilinear planar 3-SAT** (**R-P-3-SAT**) problem can be defined as follows. For each variable or clause we take a horizontal line ***segment***. The variable segments are placed on a horizontal line and clause segments are connected to these variable segments either from above or below by vertical line segments called ***connections*** such that none of these line segments and connections intersect. The goal is to decide whether there a truth assignment to the variables such that ϕ is satisfiable. Figure 2 shows an instance of the R-P-3-SAT problem. Knuth and Raghunathan [9] proved that this problem is NP-complete. Note that we can order the variable segments in increasing x direction. Let $C_t = (x_i \vee \overline{x}_j \vee x_k)$ be a clause that connects the variables from above where x_i, x_j, x_k. Then we say that, x_i is a ***left***, x_j is a ***middle***, x_k is a ***right*** variable.

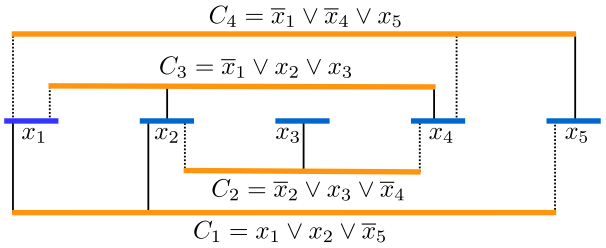


Figure 2: An instance of R-P-3-SAT problem. Solid (resp. dotted) clause vertical segments represent that the variable is positively (resp. negatively) present in the corresponding clauses. For clause C_1, x_1 is a left, x_2 is a middle, and x_5 is a right variable.

2. Discrete Dominating Set

2.1. Rectangles anchored on two lines

We prove that the DDS-R-ATL problem is NP-hard by a reduction from the R-P-3-SAT problem (see prerequisites).

Reduction: To represent a variable gadget of the DDS-R-ATL problem, we consider the graph G given in Figure 3. The following result on G can be proved easily.

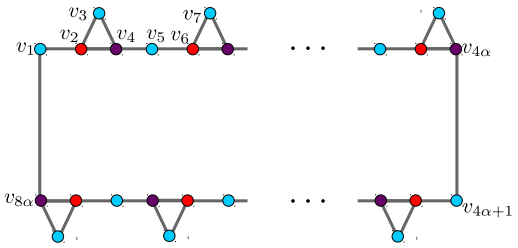


Figure 3: Structure of the graph G.

Lemma 1. *There are exactly two optimal dominating sets, $D_0 = \{v_4, v_8, \ldots, v_{8\alpha}\}$ and $D_1 = \{v_2, v_6, \ldots, v_{8\alpha-2}\}$ of vertices each with cost exactly 2α for graph G.*

We choose α to be the maximum number of clause vertical connections connecting from clause segments to a single variable segment either from above or from below. We encode the graph in Figure 3 as a variable gadget of the DDS-R-ATL problem. For each vertex, we take a rectangle and for each edge, we take a point which is contained in exactly the rectangle corresponding to the two vertices that form the edge. The gadget for variable x_i is shown in Figure 4. We take 8α rectangles R_i and 10α points P_i in two sides of a horizontal line L. The 4α rectangles $\{s^i_1, s^i_2, \ldots, s^i_{4\alpha}\}$ and 5α points $\{p^i_1, p^i_2, \ldots, p^i_{5\alpha}\}$ are one side of L and the 4α rectangles $\{s^i_{4\alpha+1}, s^i_{4\alpha+2}, \ldots, s^i_{8\alpha}\}$ and 5α points $\{p^i_{5\alpha+1}, p^i_{5\alpha+2}, \ldots, p^i_{10\alpha}\}$ are another side L. Therefore, by Lemma 1 we conclude that for each variable gadget there are exactly two optimal dominating set of rectangles $S^0_i = \{s^i_4, s^i_8, \ldots, s^i_{8\alpha}\}$ and $S^1_i = \{s^i_2, s^i_6, \ldots, s^i_{8\alpha-2}\}$ each size exactly 2α. This represents the truth value of the variable x_i.

The construction of the clause gadgets in the above and below are independent, and hence we describe the clause gadgets only for the above. For a clause C_t which contains three variables x_i, x_j and x_k in this order from left to right, we take a rectangle r^t and three points $p^{t_i}, p^{t_j}, p^{t_k}$. The bottom boundary of r^t, say b^t, are on the horizontal segment of C_t. In Figure 5, we give a schematic diagram of the clause rectangles and positions

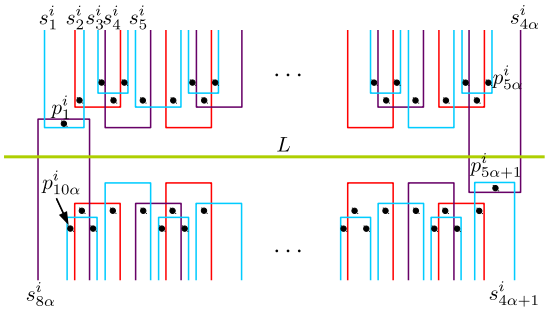


Figure 4: Structure of a variable gadget.

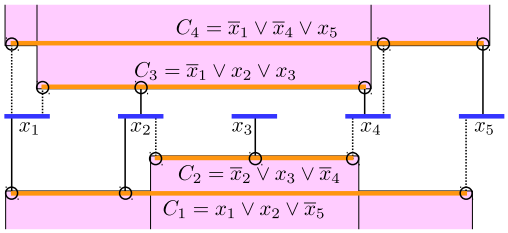


Figure 5: Schematic diagram of the clause rectangles and position of the points (circles) and their interaction with the variable gadget.

of the points corresponding to the clauses. We now describe how $r^t, p^{t_i}, p^{t_j}, p^{t_k}$ interact with the variable gadgets.

For each variable x_i, $1 \le i \le n$, sort the vertical connections from left to right which connect to x_i from clauses connecting from above. Let the clause C_t connects to x_i via l-th connection, then we say that C_t is the l-th clause for the variable x_i.

Let C_t be a clause containing the three variables x_i, x_j and x_k in this order from left to right. Here x_i is a left variable in the clause C_t and let C_t be the l_1-th clause for x_i. If x_i occurs as a positive literal in C_t, then we place the point p_{t_i} on b^t and inside the rectangle $s^i_{4l_1+4}$ only. Otherwise, we place the point p_{t_i} on b^t and inside the rectangle $s^i_{4l_1+2}$ only. The interaction is similar for x_j (middle variable) and x_k (right variable) by replacing l_1 with l_2 and l_3 respectively. See Figure 6 for the above construction. Clearly, the above construction can be done in polynomial time. We now prove the correctness of the above construction.

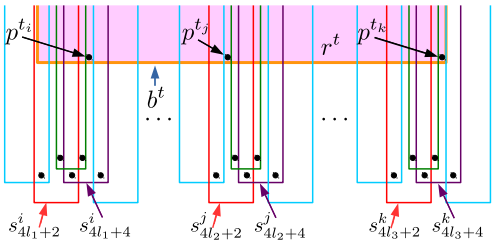


Figure 6: Structure of a clause gadget and its interaction with variable gadgets.

Lemma 2. *The formula ϕ is satisfiable iff there exists a solution to $\mathscr{D}$, an instance of the* DDS-R-ATL *problem constructed from ϕ, with cost at most $2\alpha n$.*

Proof. Let us assume that the formula ϕ is satisfiable and let $A : \{x_1, x_2, \ldots, x_n\} \to \{true, false\}$ be a satisfying assignment. For the i-th variable gadget, take the solution S^0_i if $A(x_i) = true$.

Otherwise take S_i^1 if $A(x_i) = false$. Clearly, we choose a total of $2\alpha n$ rectangles and these rectangles dominates all the variable and clause rectangles.

On the other hand, Suppose that there is a solution to $\mathscr{D}$ with cost at most $2\alpha n$. To dominate all the half-strips in a variable gadget requires at least 2α rectangles (see Claim 1). Note that all the variable gadgets are disjoint. Therefore, from each variable gadget we must choose exactly 2α rectangles (either set S_i^0 or set S_i^1). Set a variable to *true* if S_i^0 is chosen in its variable gadget, otherwise set it to *false*. Note that there are three points in a clause rectangle. Since the clause rectangle is dominated, at least one of these three points is covered by the solution. Such a point is either in solution S_i^0 or in solution S_i^1 of the corresponding variable gadget based on whether the variable is positively or negatively present in the clause. Hence, the above assignment is a satisfying assignment. □

Theorem 1. *The* DDS-R-ATL *problem is* NP*-hard.*

2.2. Rectangles stabbed by a horizontal line

We prove that the DDS-R-SHL problem is NP-hard. The reduction is similar to the reduction that is shown in Section 2.1. Here also we encode the graph G in Figure 3 as a variable gadget (see Figure 7(a)). Note that in Figure 7(a), all the rectangles are anchored on L except $s^i_{4\alpha}$ and $s^i_{8\alpha}$ that is stabbing in the middle by L. Clearly, using Lemma 1, we say that for each variable gadget there are exactly two optimal dominating sets of rectangles $S_i^0 = \{s^i_4, s^i_8, \ldots, s^i_{8\alpha}\}$ and $S_i^1 = \{s^i_2, s^i_6, \ldots, s^i_{8\alpha-2}\}$ each with size exactly α. These two sets represent the truth value of x_i.

The clause gadgets are exactly the same as the clause gadgets in Section 2.1. However here the interaction between the clause gadgets and the variable gadgets is different. We first reverse the connection of the clauses in the R-P-3-SAT problem instance i.e., the clauses those connect the variables from above (resp below) are now connect the variables from below (resp above). Now observe that the description of the variable clause connection for the clauses that connects to the variable from above in Section 2.1 are true here for the variable clause connection for the clauses that connects to the variable from below. In Figure 7(b), we give a schematic diagram of the clause rectangle and position of the points corresponding to the clauses.

A similar proof of Theorem 1 leads to the following theorem.

Theorem 2. *The* DDS-R-SHL *problem is* NP*-hard.*

3. Discrete Independent Set

3.1. Rectangles anchored on two lines

We prove that the DIS-R-ATL problem is NP-hard. We give a reduction from the R-P-3-SAT problem (see prerequisites).

Reduction: Note that, we choose α to be the maximum number of clause vertical connections connecting from clause segments to a single variable segment either from above or from below. The gadget for the variable x_i is shown in Figure 8(a). We take 16α rectangles and 16α points in two sides of a horizontal line L. The 8α rectangles $\{s^i_1, s^i_2, \ldots, s^i_{8\alpha}\}$ and 8α points $\{p^i_1, p^i_2, \ldots, p^i_{8\alpha}\}$ are one side of L and the 8α rectangles $\{s^i_{8\alpha+1}, s^i_{8\alpha+2}, \ldots, s^i_{16\alpha}\}$ and 8α points $\{p^i_{8\alpha+1}, p^i_{8\alpha+2}, \ldots, p^i_{16\alpha}\}$

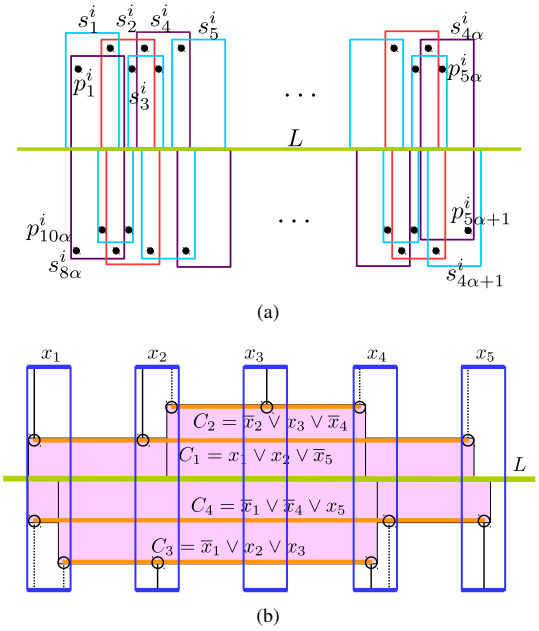
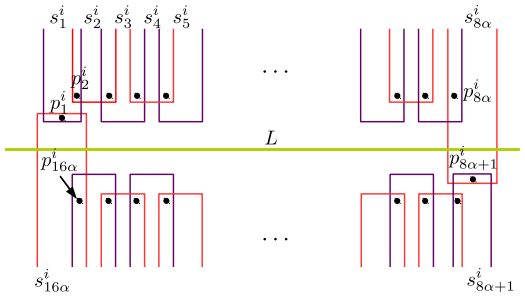


Figure 7: (a) A variable gadget. (b) Schematic diagram of the variable and clause gadgets and their interaction. Blue rectangles are schematically represent the variable gadgets.

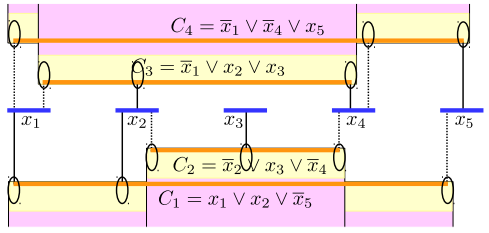
are another side of L. Each pair of consecutive rectangles have a point in common. Since these rectangles forms a cycle graph, where rectangles corresponding to vertices and two rectangles share a point if and only if there is an edge between the corresponding vertices of these two rectangles.

Observation 1. *For each variable gadget there are exactly two optimal independent sets of rectangles* $H_i^0 = \{s^i_2, s^i_4, \ldots, s^i_{16\alpha}\}$ *and* $H_i^1 = \{s^i_1, s^i_3, \ldots, s^i_{16\alpha-1}\}$ *each with size exactly* 8α.

The construction of the clause gadgets in above and below are independent, and hence we describe the clause gadgets only for above. Let C_t be a clause that contains variables x_i, x_j and x_k in this order from left to right. For C_t, we take 5 rectangles $\{s^t_1, s^t_2, s^t_3, s^t_4, s^t_5\}$ and 6 points; 1 point p^{t_i} corresponding to x_i, 4 points $p^{t_j}_1, p^{t_j}_2, p^{t_j}_3, p^{t_j}_4$ corresponding to x_j, and 1 point p^{t_k} corresponding to x_k. The rectangle s^t_1 covers the points $\{p^{t_i}, p^{t_j}_1, p^{t_j}_4\}$, s^t_2 covers $\{p^{t_i}, p^{t_j}_1, p^{t_j}_2\}$, s^t_3 covers $\{p^{t_j}_2, p^{t_j}_3\}$, s^t_4 covers $\{p^{t_j}_1, p^{t_j}_2, p^{t_j}_4, p^{t_k}\}$, and s^t_5 covers $\{p^{t_j}_1, p^{t_j}_4, p^{t_k}\}$. See Figure 9 for this construction. We now describe the placement of the points and rectangles with respect to the variable gadget. We take a rectangle r^t. The bottom boundary of r^t, say b^t, are on the horizontal segment of C_t and it can extends to the infinity (actually we can take horizontal line in a far enough distance such that the top boundaries of all such rectangles touch it) in the upward direction. We take a horizontal thin rectangular ***region*** along the top edge of r^t. We place points corresponding to the clause C_t inside this region. The rectangles corresponding to C_t are exclusively in r^t and their top boundaries are inside the region. In Figure 8(b), we give a schematic diagram of the clause rectangles, regions, and positions of the points corresponding to the clauses. We now describe how the rectangles and points corresponding to the clauses interact with the rectangles and points corresponding to variables.



(a)



(b)

Figure 8: (a) Structure of a variable gadget. (b) Schematic diagram of the variable and clause gadgets and their interaction.

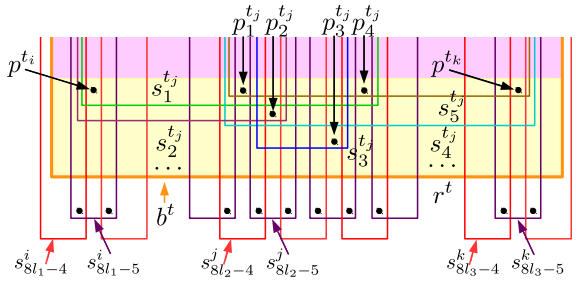


Figure 9: Position of the rectangles and points (empty circles) corresponding to the clauses.

For each variable x_i, $1 \le i \le n$, sort the vertical connections from left to right that connect to x_i from clauses connecting from above. Let clause C_t connect to x_i via l-th connection, then we say that C_t is the l-th clause for the variable x_i. Assume that the clause C_t contains three variables x_i, x_j, and x_k in this order from left to right. We now have the following cases.

- Here x_i is a left variable in the clause C_t and let C_t be the l_1-th clause for x_i. If x_i occurs as a positive literal in C_t, then we place the point p^{t_i} inside the rectangle $s^i_{8l_1-5}$. Otherwise, we place the point p^{t_i} inside the rectangle $s^i_{8l_1-4}$.
- Here x_j is a middle variable in the clause C_t and let C_t be the l_2-th clause for x_j. If x_j occurs as a positive literal in C_t, then we place the point $p^{t_j}_1$, $p^{t_j}_2$, $p^{t_j}_3$, and $p^{t_j}_4$ inside the rectangle $s^j_{8l_2-6}$, $s^j_{8l_2-5}$, $s^j_{8l_2-3}$, and $s^j_{8l_2-2}$ respectively. Otherwise, we shift all the points one rectangle to the right.
- Here x_k is a right variable in the clause C_t and let C_t be the l_3-th clause for x_k. If x_k occurs as a positive literal in C_t, then we place the point p^{t_k} inside the rectangle $s^k_{8l_3-5}$. Otherwise, we place the point p^{t_k} inside the rectangle $s^k_{8l_3-4}$.

See Figure 9 for the above construction. Clearly, the construction described above can be done in polynomial time.

Theorem 3. *The* DIS-R-ATL *problem is* NP*-hard.*

Proof. We prove that formula ϕ is satisfiable if and only if there exists a solution to the DIS-R-ATL problem instance $\mathscr{D}$ with cost $8\alpha n + m$. Assume that ϕ has a satisfying assignment. From the gadget of x_i, select the set H^1_i if the x_i is true. Otherwise select the set H^0_i. Hence we select a total of $8\alpha n$ rectangles from the variable gadget. Observe that the way we construct the clause gadget, if the clause is satisfied then exactly one of the rectangles corresponding to each clause is selected in an independent set. Hence we get a solution of $8\alpha n + m$ rectangles.

On the other hand, assume that $\mathscr{D}$ has a solution with $8\alpha n + m$ rectangles. From the gadget of x_i we select 8α rectangles either H^0_i or H^1_i. We set x_i to be true if H^1_i is selected otherwise set x_i to be false if H^0_i is selected. We now argue that this is a satisfying assignment of ϕ i.e., every clause is satisfied. Consider a clause $C_t = (x_i \vee x_j \vee x_k)$ (a similar argument can be applied for other clauses as well). If C_t is not satisfied, then we select the sets H^0_i, H^0_j, and H^0_k from the corresponding variable gadget. These rectangles prevent in selecting any rectangle from the set of rectangles corresponding to C_t. This contradicts the fact that the size of the solution is $8\alpha n + m$. However if one of H^1_i, H^1_j, and H^1_k is selected then from the set of rectangles of C_t, exactly one rectangle is selected in a solution. Therefore, the above assignment is a satisfying assignment. □

3.2. Rectangles stabbed by a horizontal line

In this section we prove that the DIS-R-SHL problem is NP-hard. The reduction is from the R-P-3-SAT problem and is a composition of the two reductions in Sections 3.1 and 2.2.

Theorem 4. *The* DIS-R-SHL *problem is* NP*-hard.*

References

[1] R. Hassin, N. Megiddo, Approximation algorithms for hitting objects with straight lines, Discrete Applied Mathematics 30 (1) (1991) 29 – 42.

[2] T. M. Chan, E. Grant, Exact algorithms and APX-hardness results for geometric packing and covering problems, Computational Geometry 47 (2, Part A) (2014) 112 – 124.

[3] M. J. Katz, J. S. B. Mitchell, Y. Nir, Orthogonal segment stabbing, Comput. Geom. Theory Appl. 30 (2) (2005) 197–205.

[4] J. M. Keil, J. S. Mitchell, D. Pradhan, M. Vatshelle, An algorithm for the maximum weight independent set problem on outerstring graphs, Comput. Geom. Theory Appl. 60 (C) (2017) 19–25.

[5] H. Kong, Q. Ma, T. Yan, M. D. F. Wong, An optimal algorithm for finding disjoint rectangles and its application to PCB routing, in: Design Automation Conference, 2010, pp. 212–217.

[6] A. Ahmadinejad, H. Zarrabi-Zadeh, Finding maximum disjoint set of boundary rectangles with application to PCB routing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36 (3) (2017) 412–420.

[7] A. Ahmadinejad, S. Assadi, E. Emamjomeh-Zadeh, S. Yazdanbod, H. Zarrabi-Zadeh, On the rectangle escape problem, Theoretical Computer Science 689 (2017) 126 – 136.

[8] A. Mudgal, S. Pandit, Geometric hitting set, set cover and generalized class cover problems with half-strips in opposite directions, Discrete Applied Mathematics 211 (2016) 143–162.

[9] D. E. Knuth, A. Raghunathan, The problem of compatible representatives, SIAM Journal on Discrete Mathematics 5 (3) (1992) 422–427.

Min–# Curve Simplification, Revisited

Mees van de Kerkhof[1], Irina Kostitsyna[2], Maarten Löffler[1], Majid Mirzanezhad[3], and Carola Wenk[3]

[1] Utrecht University, Netherlands, {m.a.vandekerkhof,m.loffler}@uu.nl
[2] Eindhoven University of Technology, Netherlands, i.kostitsyna@tue.nl
[3] Tulane University, USA, {mmirzane,cwenk}@tulane.edu

Abstract. In this paper we consider the classical min–# curve simplification problem. Given a polygonal curve P with n vertices in $\mathbb{R}^d$ and $\delta > 0$, we aim to simplify P by another polygonal curve P' with the minimum number of vertices such that the Fréchet distance between them is at most δ. We present an $O(n^4)$ time dynamic programming algorithm for the min–# problem that uses $O(n^3)$ space when the vertices in P' are selected from a subsequence of vertices in P. Our result is an improvement of the one recently proposed by van Kreveld et al [10].

1 Introduction

Approximating a polygonal curve by another curve is a long-standing problem in computational geometry. One of the most well-known settings that has received considerable attention is the *min–# simplification problem*: given a polygonal curve $P = \langle p_1, p_2, \cdots, p_n\rangle$ in $\mathbb{R}^d$, a distance measure $\mathsf{D}(\cdot,\cdot)$ between two curves and a real value $\delta > 0$, find a polygonal curve $P' = \langle p'_1, p'_2, \cdots, p'_k\rangle$ with the minimum number of vertices such that $\mathsf{D}(P, P') \le \delta$. We call the edges in P' *links*.

There are several variants of the min–# problem: (1) *vertex-restricted*, where vertices of P' have to be a subsequence of vertices of P, (2) *curve-restricted*, where vertices of P' can lie anywhere on P, and (3) *non-restricted*, when vertices of P' can be anywhere in the ambient space. Given a distance measure $\mathsf{D}(\cdot,\cdot)$ between two curves, such as Hausdorff δ_H, directed Hausdorff $\overrightarrow{\delta_H}$, or Fréchet distance δ_F, one can apply this distance in a *global* or *local* way to the min–# problem as follows: First, one can simply measure the distance $\mathsf{D}(P, P')$ between the two curves; we denote this as the *global* distance D^g. In the local distance D^ℓ, for the vertex- and curve-restricted cases, one measures the distance between each link in P' and its corresponding subcurve in P whose endpoints are identical, and returns the maximum of these distances. In this paper, we focus on global distance measures in the vertex-restricted case.

There have been numerous results on different variants of the min–# problem mostly for the vertex-restricted version under local distance measures. The classical algorithm proposed by Imai and Iri [8] solves the vertex-restricted min–# problem under $\overrightarrow{\delta_H^\ell}$ from P' to P. While their algorithm runs in $O(n^2 \log n)$ time, Chan and Chin [6] improved the running time to $O(n^2)$. Guibas et al. [7] proposed an algorithm that computes the non-restricted δ_F^g in the plane in $O(n^2 \log^2 n)$ time using $O(n)$ space. Agarwal et al. [1] gave a near linear time approximation algorithm for the vertex-restricted version under δ_F^ℓ for any L_p metric. Recently van Kreveld et al. [10] considered global distance measures and proved that the vertex-restricted min–# problem under $\delta_H^g(P, P')$ is NP-hard, whereas they gave a polynomial time output-sensitive algorithm under $\delta_F^g(P, P')$. They also gave a polynomial time algorithm for the min–# simplification under $\overrightarrow{\delta_H^g}(P', P)$, however they showed that the problem under $\overrightarrow{\delta_H^g}(P, P')$ is NP-hard. See Table 1 for an overview of existing results and our results. In this paper we only present one of our results for the vertex-restricted case under the global Fréchet distance. Proofs of our lemmas are skipped due to the space constraint and can be found in [11].

2 Preliminaries

Let $P = \langle p_1, p_2, \cdots, p_n\rangle$ be a polygonal curve. We treat a *polygonal curve* as a continuous map $P : [1, n] \to \mathbb{R}^d$ where $P(i) = p_i$ for an integer i, and the i-th edge is linearly parametrized as $P(i + \lambda) = (1 - \lambda)p_i + \lambda p_{i+1}$, for integer i and $0 < \lambda < 1$. A *re-parametrization* $\sigma : [0, 1] \to [1, n]$ of P is any continuous, non-decreasing function such that $\sigma(0) = 1$ and $\sigma(1) = n$. We denote the subcurve between $P(s)$ and $P(t)$ by $P[s, t]$, where $1 \le s \le t \le n$. To compute the Fréchet distance between P and Q with n and m vertices, respectively, Alt and Godau [3] introduced the notion of *free-space diagram*. For any $\delta > 0$, we denote the free-space diagram between P and Q by $\mathsf{FSD}_\delta(P, Q)$. This diagram has the domain of $[1, n] \times [1, m]$ and it consists of $(n - 1) \times (m - 1)$ cells, where each point (s, t) in the diagram corresponds to two points $P(s)$ and $Q(t)$. A point (s, t) in $\mathsf{FSD}_\delta(P, Q)$ is called *free* if $\|P(s) - Q(t)\| \le \delta$, and *blocked* otherwise. The union of all free points is referred to as the *free space*. The intervals induced by free space in $\mathsf{FSD}_\delta(P(i), Q)$ for all $i = 1, \cdots, n$ is called *free space intervals* of the range $i \times [1, m]$. *Fréchet matching* between P and Q is a pair of re-parameterizations (σ, θ) corresponding to an xy-monotone path from $(1, 1)$ to (n, m) within the free space in $\mathsf{FSD}_\delta(P, Q)$. The *Fréchet distance* between two curves is defined as $\delta_F(P, Q) = \inf_{(\sigma,\theta)} \max_{0\le t\le 1} \|P(\sigma(t)) - Q(\theta(t))\|$. Let $a = (x_1, y_1)$ and $b = (x_2, y_2)$ be two points in $\mathsf{FSD}_\delta(P, Q)$ with $x_1 \le x_2$ and $y_1 \le y_2$. We say b is *reachable* from a if there exists a Fréchet matching from a to b within $\mathsf{FSD}_\delta(P, Q)$. A Fréchet matching in $\mathsf{FSD}_\delta(P, Q)$ from a to b is also called a *reachable path* between a and b denoted by $\mathcal{P}(a, b)$. If $a = (1, 1)$ we denote it by $\mathcal{P}(1, b)$. Alt and Godau [3] compute a reachable path

Distance type	Vertex-restricted	Curve-restricted	Non-restricted
$\overrightarrow{\delta_H^g}(P, P')$	NP-hard [10]	N/A	N/A
$\overrightarrow{\delta_H^g}(P', P)$	$O(n^4)$ [10] $O(n^2 \log n)$☆	NP-hard☆	poly(n) [9]
$\delta_H^g(P, P')$	NP-hard [10]	N/A	N/A
$\delta_F^g(P, P')$	$O(mn^5)$ [10] $O(n^4)$☆ (Section 3) $O(n^3)$☆ EREW PRAM	$O(n)$ in $\mathbb{R}^1$☆	$O(n^2 \log^2 n)$ in $\mathbb{R}^2$ [7] $O^*\left(n^2 \log n \log\log n\right)$☆
$\delta_{dF}^g(P, P')$	$O(n^2)$ [5]	N/A	$O(n \log n)$ [5]

Table 1. Known results for the min-# problem under global and local distance measures. Our results are marked with ☆. In this paper we only present the result for the vertex-restricted min–# problem under $\delta_F^g(P, P')$. Our other results marked with ☆ can be found in [11].

by propagating reachable points across free space cell boundaries in a dynamic programming manner, which requires the exploration of the entire $\mathsf{FSD}_\delta(P, Q)$ and takes $O(mn)$ time.

3 Vertex-Restricted Under $\delta_F^g(P, P')$ in $\mathbb{R}^d$

Let $P = \langle p_1, p_2, \cdots, p_n \rangle$ be a polygonal curve with n vertices, where $P : I \to \mathbb{R}^d$ with $I = [1, n]$. We construct a DAG $G = (V, E)$ such that $V = \{1, 2, \cdots, n\}$ and $E = \{(i, j) \mid 1 \le i < j \le n\}$. Here, we consider each vertex $v \in V$ to be embedded at p_v and each edge $(u, v) \in E$ to be embedded as the straight line segment shortcut $\overline{p_u p_v}$ between p_u and p_v, parameterized linearly by $\overline{p_u p_v}(j) = \frac{v-j}{v-u} p_u + \frac{j-u}{v-u} p_v$ for all $j \in [u, v]$. For any $\delta > 0$ we now define the free space surface for P and G; see also [2, 3] for the definitions of free space and free space surface.

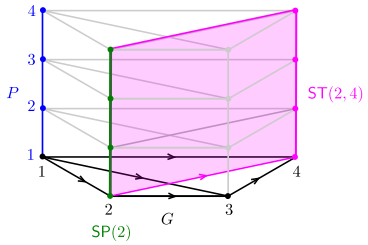


Fig. 1. Schematic example of a free space surface.

Definition 1 (Strips and Spines). *Let $(u, v) \in E$. The* δ-strip $\mathsf{ST}_\delta(u, v) = \{(i, j) \mid 1 \le i \le n, u \le j \le v, ||P(i) - \overline{p_u p_v}(j)|| \le \delta\}$ *is the free space between P and $\overline{p_u p_v}$. The* strip *is defined as $\mathsf{ST}(u, v) = I \times [u, v]$. We have $\mathsf{ST}_\delta(u, v) \subseteq \mathsf{ST}(u, v)$. The* δ-spine $\mathsf{SP}_\delta(v) = \{(v, i) \mid 1 \le i \le n, ||P(i) - p_v|| \le \delta\}$ *is the free space between P and p_v. The* spine *is defined as $\mathsf{SP}(v) = I \times v$. We have $\mathsf{SP}_\delta(v) \subseteq \mathsf{SP}(v)$.*

For any edge $(u, v) \in E$, both spines centered at the vertices of the edge are subsets of the strip: $\mathsf{SP}_\delta(u), \mathsf{SP}_\delta(v) \subseteq \mathsf{ST}_\delta(u, v)$ and $\mathsf{SP}_\delta(u)$ is a subset of all strips with respect to edges incident on u. The *free space surface* of P and G, denoted by $\mathsf{FSD}_\delta(P, G)$, is the collection of all strips (and spines) over all $v \in V$ and $(u, v) \in E$; see Fig. 1. Note that the edges in E are directed, and hence any reachable path has to visit a sequence of spines that corresponds to an increasing sequence of vertices in V.

The goal of our algorithm is to compute a reachable path in the free space surface from $(1, 1)$ to (n, n) that uses the minimum number of strips. The main idea is to use dynamic programming to propagate a reachable path with a minimum number of strips from spine to spine. Each spine $\mathsf{SP}(v) = I \times v$ contains a sequence of free space intervals of $\mathsf{SP}_\delta(v)$. Let $S \subseteq I$ be the union of all interval endpoints of free space intervals of $\mathsf{SP}_\delta(v)$ for all $v \in V$, projected onto I. The set S induces a partition of I into intervals. For each $v \in V$ let $\mathsf{L}_\delta(v)$ be the ordered list of free space intervals obtained by subdividing the free space intervals of $\mathsf{SP}_\delta(v)$ with all points in S. We call the intervals in $\mathsf{L}_\delta(v)$, whose ending points are excluded, *elementary intervals*; see Fig. 2.

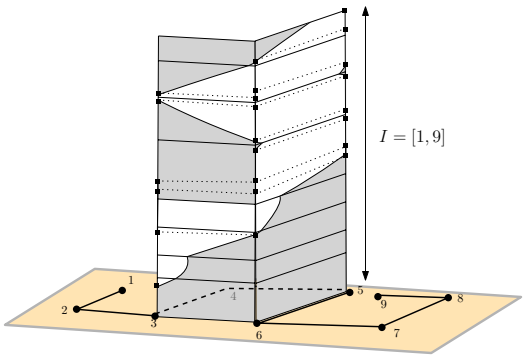


Fig. 2. Elementary intervals in each spine are created by overlaying all free space intervals onto each spine.

We assume that elementary intervals in $\mathsf{L}_\delta(v)$ are ordered in increasing order of their starting point. When clear from the context we may identify elementary intervals with their projections onto I, and use $<$ and $=$ to compare intervals in the resulting total ordering of all elementary intervals along I.

We now extend the definition of a reachable path $\mathcal{P}(a,b)$ whose starting and ending points are defined with respect to two points a and b in $\mathsf{FSD}_\delta(P,Q)$ to the one that is defined with respect to two elementary intervals $r \in \mathsf{L}_\delta(u)$ and $e \in \mathsf{L}_\delta(v)$ with $r \leq e$ and $u \leq v$. We denote a *elementary reachable path* from an elementary interval r to elementary interval e by $\overline{\mathcal{P}}(r,e)$. $\mathbf{L}(\overline{\mathcal{P}}(r,e))$ denotes the length of a reachable path that is the number of strips visited by $\overline{\mathcal{P}}(r,e)$. If an elementary reachable path starts at $(1,1)$ and ends at $e \in \mathsf{L}_\delta(v)$ we denote it by $\overline{\mathcal{P}}(1,e)$ for more simplicity in our notation.

We define the cost function $\phi : V \times I \to \mathbb{N}$ for any point $z = (v,x) \in \mathsf{SP}_\delta(v)$ with $v \in V$ as $\phi(v,z) = \min_{\mathcal{P}(1,z)} \mathbf{L}(\mathcal{P}(1,z))$, where the minimum ranges over all reachable paths $\mathcal{P}(1,z)$ in the free space surface. If no such path exists then $\phi(v,z) = \infty$. Lemma 1 below shows some properties of elementary intervals and their sufficiency to propagate ϕ values across $\mathsf{FSD}_\delta(P,Q)$.

Lemma 1 (Elementary Intervals Properties). *Let $v \in V$, $a,b \in [1,n]$. The following statements are true:*

1. *For any two points $x = (v,a)$ and $y = (v,b)$ with $x,y \in e$ and $e \in \mathsf{L}_\delta(v)$, $\phi(v,x) = \phi(v,y)$*
2. *$|\mathsf{L}_\delta(v)| \leq 2n^2 + n$,*

Since by Property 1 of Lemma 1, ϕ is fixed on each elementary interval, we will write $\phi(v,e)$ for each elementary interval $e \in \mathsf{L}_\delta(v)$ for all $v \in V$. A subset of $\mathsf{L}_\delta(v)$ that consists of all points that are reachable from an elementary interval $r \in \mathsf{L}_\delta(u)$ with $u \leq v$ is called a *reachable interval* in $\mathsf{L}_\delta(v)$ from r denoted by $\overline{\mathcal{I}}(v,r)$. Note that not only elementary intervals but also reachable intervals can be passed as the second argument of the ϕ function since the domain is defined as $V \times I$. Lemma 2 below shows a recursive formula for ϕ, which we will be used in our dynamic programming algorithm.

Lemma 2. *For all elementary intervals $e \in \mathsf{L}_\delta(1)$: If e is reachable from $(1,1)$ then $\phi(1,e) = 0$, otherwise $\phi(1,e) = \infty$. For all $v \in V \setminus \{1\}$ and all elementary intervals $e \in \mathsf{L}_\delta(v)$: $\phi(v,e) = \min_{u \leq v} \min_{r \leq e} \phi(u,r) + 1$ such that there is a reachable path from $r \in \mathsf{L}_\delta(u)$ to e within $\mathsf{ST}_\delta(u,v)$.*

Algorithm 1 shows our dynamic programming algorithm that computes ϕ using the recursive formula in Lemma 2. Our algorithm consists of two main steps: (1) *Initialization*, (2) *Propagation*.

We provide a more detailed description of the pseudocode in the following:

1. **Initialization (lines 1-5):** We first compute the free space surface induced by G and P by incorporating strips (and spines) respecting the adjacency of strips in G. We compute all the elementary interval using the ElemInterval procedure and set the cost of all of the elementary intervals to infinity. For those elementary intervals that are reachable from $(1,1)$ we set their ϕ value to zero (line 5).
2. **Propagation (lines 6-15):** We process spines in increasing order according to their indices in V (except for the first one). For a $\mathsf{SP}_\delta(v)$ we compute the reachable intervals $\overline{\mathcal{I}}(v,r)$ originating from any elementary interval r in previous spines $\mathsf{SP}_\delta(u)$ for all, $u = 1, \cdots, v$ (line 10) by means of a procedure called ReachInterval. The arguments of this procedure are an elementary interval and a spine. We give more details about this procedure below. Next, we assign reachable interval $\overline{\mathcal{I}}(v,r)$ the cost of r incremented by one (line 11). We take the union of all reachable intervals and store them in a set U_v (line 12). Now, we call Subdivide whose arguments are the set U_v and the current spine $\mathsf{SP}_\delta(v)$. This procedure computes a subdivision of reachable intervals in U_v in terms of their ϕ-values in which each subdivided interval acquires the minimum ϕ value among all ϕ values of reachable intervals in U_v that contain the subdivided interval. We finally assign the cost of each subdivided interval to each elementary interval in the current spine by linearly traversing both $\mathcal{L}$ and $\mathsf{L}_\delta(v)$ (see Fig. 3).

Algorithm 1: Compute Vertex-Restricted Min–# Simplification Under $\delta_F^q(P,P') \leq \delta$

```
1  Compute FSD_δ(P, G);
2  for each v ∈ V :
3      L_δ(v) ← ElemInterval(FSD_δ(P, G), SP_δ(v));
4      for each e ∈ L_δ(v) : φ(v, e) = ∞;
5  for each e ∈ L_δ(1) that is reachable from (1, 1) :
       φ(1, e) = 0;
6  for each v ∈ V − {1} :
7      U_v = ∅;
8      for each u ∈ {1, ··· , v} :
9          for each r ∈ L_δ(u) :
10             Ī(v, r) ← ReachInterval(r, SP_δ(v));
11             φ(v, Ī(v, r)) = φ(u, r) + 1;
12             U_v = Ī(v, r) ∪ U_v;
13 L ← Subdivide(U_v, L_δ(v));
14 for each e ∈ L_δ(v) :
15     φ(v, e) = min_{ℓ∈L} φ(v, ℓ), where e ∈ ℓ.
16 Return vertices of P′ by tracing back on φ values.
```

We now provide further details for the two procedures ReachInterval and Subdivide.

Elementary Reachable Interval Procedure. The two arguments of this procedure are an elementary interval $r \in \mathsf{L}_\delta(u)$ and a spine $\mathsf{SP}_\delta(v)$, $u \leq v$. This procedure computes an interval in $\mathsf{L}_\delta(v)$ that contains all points in $\mathsf{SP}_\delta(v)$ that are reachable

from r within $\mathsf{ST}_\delta(u,v)$. Note that this interval may contain some non-reachable points as well. All we need is to find start and end pointers of such a reachable interval in $\mathsf{SP}_\delta(v)$. This can be done by using the algorithm proposed in Lemma 3 of [2]. We have the following lemma:

Lemma 3. *Let u and v be two integers such that $1 \leq u < v \leq n$. One can compute all $\cup_{r\in\mathsf{L}_\delta(u)}\overline{\mathcal{I}}(v,r)$ in $O(|\mathsf{L}_\delta(u)|)$ time.*

Subdivide Procedure. We are given a set of reachable intervals U_v with their ϕ values. We need to efficiently compute a subdivision of reachable intervals in U_v where each subdivided interval takes the minimum value of ϕ from all subintervals in U_v which overlap the respective subdivided interval. To achieve this, we overlay all intervals in U_v vertically onto the XY-coordinate plane in such a way that their start and end points are (increasingly) encountered along the Y-axis and their ϕ values are (increasingly) set along the X-axis. Due to the monotonicity of reachable intervals their end and start points are sorted along $\mathsf{SP}_\delta(v)$ and correspondingly along the Y-axis as well. Now all we need is to compute the lower envelope (in terms of ϕ values) over all intervals in U_v (see Fig. 3). The algorithm described in Section 5 of [4] allows us to achieve this efficiently for the case where the ordering in which the start and end points of intervals are being visited along the Y-axis is given. We have the following lemma:

Lemma 4. *Let $S = \{S_1, \cdots, S_N\}$ be a set of segments in the plane and the left-to-right order of endpoints of segments in S is given. One can compute a lower envelope of size $O(N)$ induced by segments in S in $O(N)$ time.*

Applying the lemma to U_v yields a subdivision of intervals into some subdivided subintervals each with their ϕ value (see Fig 3). We immediately have the following lemma:

Lemma 5. *For any $v \in V \setminus \{1\}$, $\textsc{Subdivide}\ (U_v, \mathsf{L}_\delta(v))$ computes a list $\mathcal{L}$ as a subdivision of U_v in $O(|U_v|)$ time, where $|U_v| = v \cdot \left(\max_{u=1}^{v} |\mathsf{L}_\delta(u)|\right)$ and $|\mathcal{L}| = O(|U_v|)$.*

Theorem 1. *Algorithm 1 runs in time $O(n^4)$ using $O(n^3)$ space.*

4 Discussion

In this paper we provided a dynamic programming algorithm for the vertex-restricted case under the global Fréchet distance that significantly improves the running time of the algorithm in [10]. It would be interesting to give a faster algorithm.

Acknowledgment. Carola Wenk and Majid Mirzanezhad acknowledge the support of the National Science Foundation under grant CCF-1637576.

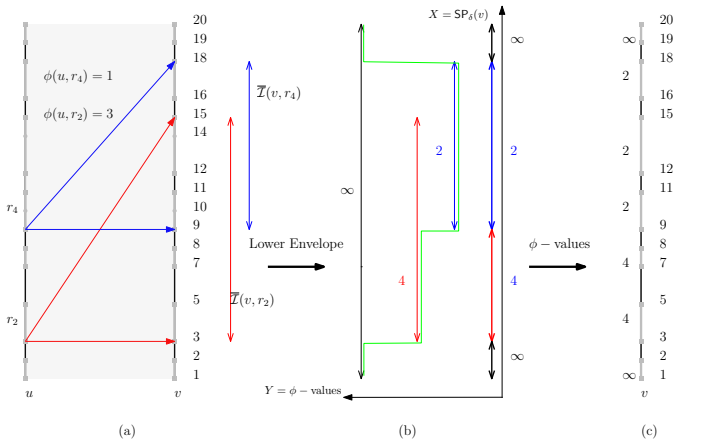


Fig. 3. In this example let $\mathsf{L}_\delta(v)$ contain elementary intervals distributed within the interval $I=[1,20]$. In (a) there are two reachable intervals $\overline{\mathcal{I}}(v,r_2)$ and $\overline{\mathcal{I}}(v,r_4)$ of ϕ values 4 and 2, respectively. In (b) the lower envelope technique results in having a subdivision of min ϕ values on $\mathsf{SP}_\delta(v)$. Here the green curve indicates the lower envelope of the intervals in terms of the ϕ values. In (c) each elementary interval in $\mathsf{L}_\delta(v)$ can take the min ϕ value obtained from (b).

References

1. P.K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near linear time approximation algorithm for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005.
2. H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262—283, 2003.
3. H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(1–2):75–91, 1995.
4. A. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1–4):49–63, 1985.
5. S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3d polygonal chains under the discrete Fréchet distance. volume 4957, pages 630–641. LATIN, 2008.
6. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Intl. J. Computational Geoemtry and Applications*, 6(1):59–77, 1996.
7. L. Guibas, J. Hershberger, J. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *Intl. J. Computational Geometry and Applications*, 3(4):383–415, 1993.
8. H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.
9. I. Kostitsyna, M. Löffler, V. Polishchuk, and F. Staals. On the complexity of minimum-link path problems. *Journal of Computational Geometry*, 8(2):80–108, 2017.
10. M. Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. In *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 56, pages 1–14, 2018.
11. M. van Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. On optimal min–# curve simplification problem. `http://arxiv.org/abs/1809.10269`, 2018.

Threshold-Based Graph Reconstruction Using Discrete Morse Theory*

Brittany Terese Fasy[1,2], Sushovan Majhi[3], and Carola Wenk[4]

[1]School of Computing, Montana State University
[2]Department of Mathematical Sciences, Montana State University
[3]Department of Mathematics, Tulane University
[4]Department of Computer Science, Tulane University

Abstract

Discrete Morse theory has recently been applied in metric graph reconstruction from a given density function concentrated around an (unknown) underlying embedded graph. We propose a new noise model for the density function to reconstruct a connected graph both topologically and geometrically.

1 Introduction

Graph-like or filamentary structures are very common in science and engineering. Examples include road networks, sensor networks, and earthquake trails. With the advent of modern sampling techniques, very large amounts of data, sampled around such (often hidden) structures, are becoming widely available to data analysts.

Problem Statement. Given a set of points sampled around an unknown metric graph G embedded in $\mathbb{R}^2$, output a metric graph $\widehat{G}$ that has the same homotopy type as G and has a small Hausdorff distance to G.

Background. Graph reconstruction from noisy samples has been studied extensively in the last decade; see e.g., [1, 2, 4, 5]. One can typically classify noise models for reconstruction problems into two categories: Hausdorff noise and non-Hausdorff noise. A sample S may not lie exactly on G, however S is sampled from a very small offset of G. In this case, the Hausdorff distance between the sample and ground truth is assumed to be very small. We call such a noise *Hausdorff* noise. The situation becomes different in the presence of outliers in S. If outliers in S are far away from G, they contribute to an uncontrollably large Hausdorff distance. In this paper, we aim at geometric reconstruction of Euclidean graphs under *non-Hausdorff* noise model.

Several non-Hausdorff based graph reconstruction approaches use the density of the sample points in the ambient space. In this density-based reconstruction regime, a density function over a rectangular grid of pixels in the plane is computed from the raw sample S. There are several ways one can define density on the planar grid. A histogram computation or a kernel density estimate are usually very popular and easy to implement in practice. Then, an appropriate threshold is chosen to get a thickened graph as the super-level set of the density at the threshold. Some algorithms work by choosing this threshold empirically, whereas some others, e.g., [1], use systematic topological techniques like persistent homology to choose a set of thresholds just big enough to capture the desired topologi-

*The authors would like to acknowledge the generous support of the National Science Foundation under grants CCF-1618469 and CCF-1618605.

cal changes in the sub-level set filtration dictated by the density function. While most of the previous approaches gained success in practice, not much has been proved theoretically to guarantee the desired topological or geometric correctness. Also, the output is usually a region around the underlying graph. Finally, one prunes the region to extract a graph like structure from it using some heuristic thinning algorithm.

Related Work. Our work is inspired by the recent work by Dey et al. [2]. The authors use a topological technique called discrete Morse theory to extract the cycles of the underlying graph G from a density function. They show that if the density function satisfies a noise model, that the authors call an (ω, β, ν)-approximation, then the output $\widehat{G}$ of their algorithm has the same homotopy type as G. However, the noise model is too simplistic to capture degree one vertices of G. For this reason, the leaves or the "hairs" of G cannot be reconstructed, resulting in a large (undirected) Hausdorff distance between $\widehat{G}$ and G.

Our Contribution. In order to overcome the above mentioned limitations of the algorithm developed in [2], we propose a two-threshold based noise model for the density function that is more practical and that can localize all vertices of G. Using different thresholds for the graph vertices and the graph edges, we develop an algorithm (Algorithm 1) that can output a reconstruction that is also geometrically close to G. We prove in Theorem 5.1 that the output of our algorithm successfully captures both the topology and geometry of the underlying graph G.

2 Discrete Morse Theory

Let K be a finite simplicial complex. A *discrete vector field* V on K is a collection of pairs $(\tau^{(p)}, \sigma^{(p+1)})$ of simplices of K such that $\tau^{(p)} < \sigma^{(p+1)}$ and each simplex of K appears in at most one of such pair. Here, the symbol '$<$' denotes the face relation and the superscript denotes the dimension of the simplex. A simplex $\sigma \in K$ is called *critical* if σ does not take part in any pair. We define a V-path as a sequence of simplices

$$\tau_0^{(p)}, \sigma_0^{(p+1)}, \tau_1^{(p)}, \sigma_1^{(p+1)}, \ldots, \sigma_r^{(p+1)}, \tau_{r+1}^{(p)},$$

where $r > 0$, $(\tau_i^{(p)}, \sigma_i^{(p+1)}) \in V$ and $\tau_{i+1}^{(p)} < \sigma_i^{(p+1)}$ for all $i \in \{0, \ldots, r\}$. The *Morse cancellation* of a pair of critical simplices τ, σ takes place when there is a unique V-path from a co-dimension one face of τ to σ. This process of cancellation reverses the vectors along that V-path to obtain another vector field on K. For more details see [3]. Finally, for a critical simplex σ we define its *stable manifold* to be the union of the V-paths that end at σ. Similarly, we define its *unstable manifold* to be the union of the V-paths that start at σ. For definitions and more details see [2, 3].

3 Double Threshold

For ease of presentation, we define the noise model in the smooth set-up. Let Ω be a planar rectangle, and let G be a finite planar graph embedded inside Ω. Let ω be a small positive number such that G^ω, the ω-offset of G, is contained in Ω and has a deformation retraction onto G. Also, for each vertex v of G, we call the ω-ball centered at v the *vertex region* of v. Now, let $\mathcal{V}^\omega$ be the union of all vertex regions of G. We call a density function f on Ω an $(\omega, \beta_1, \beta_2, \nu)$-approximation of G if

$$f(x) \in \begin{cases} [\beta_1, \beta_1 + \nu], & \text{if } x \in \mathcal{V}^\omega \\ [\beta_2, \beta_2 + \nu], & \text{if } x \in G^\omega - \mathcal{V}^\omega \\ [0, \nu], & \text{otherwise} \end{cases}$$

where $\beta_1 > \beta_2 + 2\nu, \beta_2 > 2\nu$. In this case, we call β_1 and β_2 the thresholds for f. Throughout this paper, we assume our density function is an $(\omega, \beta_1, \beta_2, \nu)$-approximation. In practice, these four parameters are unknown. However, in our algorithm, we use a cut-off δ such that $\nu < \delta < \min(\beta_2 - \nu, \beta_1 - \beta_2 - \nu)$ and which is assumed to be known to us.

In order to reconstruct G, the density is expected to assume very large values inside G^ω relative to the outside region. Here, a small noise

Data: The discretized domain K, the density function f, the threshold δ.
Result: The reconstructed graph $\widehat{G}$.
Initialize V as the trivial vector field on K.
Initialize $\widehat{G} = \emptyset$
Run persistence on the super-level set filtration of f to get the persistence pairs $P(K)$.
for $(\sigma, \tau) \in P(K)$ *with* $Pers(\sigma, \tau) < \delta$ **do**
    Try to perform a Morse cancellation for the pair.
    Update V.
end
for $(v, e) \in P(K)$ *and* $(e, t) \in P(K)$ *with persistence* $\geq \delta$ **do**
    $\widehat{G} = \widehat{G} \cup \{$ stable manifold of $e\}$.
end
output $\widehat{G}$

Algorithm 1: Graph Reconstruction Algorithm

or perturbation ν has been assumed. The above mentioned two thresholds make this noise model close to real-world applications involving the extraction of road-networks from GPS trajectory data. Because points along trajectories make the density higher near the intersections than than the edges, this noise model enables us to correctly reconstruct not only the topology but also the geometry of G as shown in Theorem 5.1.

4 Algorithm

We devise our reconstruction algorithm, Algorithm 1, by using discrete Morse cancellation guided by persistence pairs.

Analysis of Algorithm We start with a discretization K of the planar rectangle Ω. For example, K can be a planaer two-dimensional cubical complex. Let the density function $f : K \to \mathbb{R}$ be an $(\omega, \beta_1, \beta_2, \nu)$-approximation and let cutoff $\nu < \delta < \min(\beta_2 - \nu, \beta_1 - \beta_2 - \nu)$. Our goal is to construct a discrete vector field V on K that is associated to a discrete Morse function that is much simpler than f. This way, we clean the density function from the noise administered by ν. We initialize V with the initial vector field K in which all simplices are critical. In order to remove non-genuine critical simplices, we run persistence on the super-level set filtration of K defined by f. Then, for each persistence pair (σ, τ) with persistence smaller than δ, we try to perform Morse cancellation of the Morse pair (σ, τ) to update V. After the cancellations are done, we get V which is a cleaner discrete gradient field on K. We can show that the resulting V only contains genuine critical points, i.e., for each graph vertex we have a critical vertex v of K in its vertex region and for each edge e of G we have a critical edge in V. All these critical vertices and edges will be contained in G^{ω}. Moreover, these critical vertices and edges are characterized by their persistence being larger than δ. Therefore, to extract the edges of G we consider each edge of K with persistence $> \delta$ and compute their stable manifolds. The union of their stable manifolds is the reconstruction $\widehat{G}$.

5 Reconstruction Guarantees

The two thresholds help us to localize the critical vertices of the discrete gradient field inside the vertex regions. The output $\widehat{G}$ has the same homotopy type as G as shown in the following theorem.

Theorem 5.1 (Graph Reconstruction)**.** *If G is a connected, embedded planar graph in a cubical complex K and f is an $(\omega, \beta_1, \beta_2, \nu)$-approximation of G then the output $\widehat{G}$ of Algorithm 1 has the same homotopy type as G. Moreover, $d_H(G, \widehat{G}) < \omega$.*

Sketch of proof. We prove the homotopy type by showing that G and $\widehat{G}$ have the same first Betti numbers, as the homotopy type of a connected graph is completely characterized by its first Betti number.

After the termination of Algorithm 1, by the assumption on the density function, for each graph

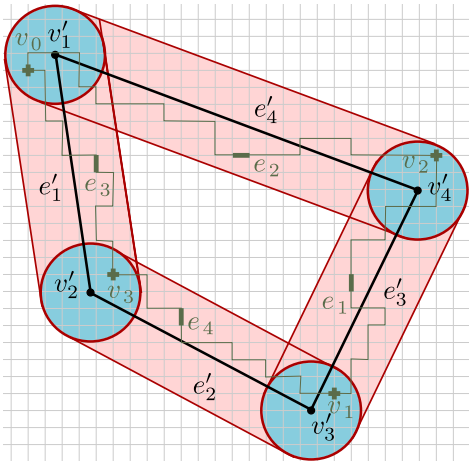


Figure 1: A graph G with vertex and edge regions. Critical edges and their stable manifolds are shown in green.

vertex v' of G we will have exactly one critical vertex v of K inside the vertex region of v'. This vertex is the local maximum of f inside the vertex region of v'. For the persistence pairings in $P(K)$ with persistence larger than δ, a vertex v of K has to be paired with either $+\infty$ or with a critical edge e of K from the edge region of a graph edge e' of G. And, e' will be incident to v', as illustrated in Figure 5. Now, for each critical edge e of K, e must lie inside one of the edge regions of G. Moreover, for each each e' of G we have exactly one critical edge e of K. For the pairings of $P(K)$ with persistence larger than δ, each edge e is either paired with a vertex v from the vertex region of an incident edge or a triangle t from the complement of G^ω.

The one-to-one correspondence of the edges of G and the critical edges of K and the vertices of G and the critical vertices of K in V, shows that the stable manifold of a critical edge e of K that lies in the edge region of a graph edge e' of G will be a path in G^ω joining the critical vertices of the vertex regions of the end-points of e'. This concludes that $\widehat{G}$ and G will have the same first Betti numbers. Also, since the critical vertices and edges are localized inside the corresponding regions we conclude that $d_H(G, \widehat{G}) < \omega$. □

6 Discussion

The nature of our project is ongoing. The noise model discussed in the paper is only a rough approximation of realistic noise models. We are still in the process of finding a better noise model. We also hope to find a condition on the density that enables us to guarantee a small Fréchet distance between the edges of G and the reconstruction.

Acknowledgments The authors acknowledge the generous support of the National Science Foundation under grants CCF-1618469 and CCF-1618605. The authors also thank Yusu Wang for her feedback.

References

[1] Ahmed, M., Fasy, B. T., Gibson, M., and Wenk, C. Choosing thresholds for density-based map construction algorithms. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2015), SIGSPATIAL '15, ACM, pp. 24:1–24:10.

[2] Dey, T. K., Wang, J., and Wang, Y. Graph reconstruction by discrete Morse theory. In *34th International Symposium on Computational Geometry* (2018), pp. 31:1–31:15.

[3] Forman, R. A user's guide to discrete Morse theory. *Séminaire Lotharingien de Combinatiore*, 48 (2002), Art. B48c.

[4] Ge, X., Safa, I., Belkin, M., and Wang, Y. Data skeletonization via reeb graphs. In *Proceedings of the 24th International Conference on Neural Information Processing Systems* (USA, 2011), NIPS'11, Curran Associates Inc., pp. 837–845.

[5] Wang, S., Wang, Y., and Li, Y. Efficient map reconstruction and augmentation via topological methods. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '15* (2015), ACM Press, pp. 1–10.

Characterizing Topological Discrepancies in Additive Manufacturing

MORAD BEHANDISH and SAIGOPAL NELATURI, PARC

Additive manufacturing (AM) enables enormous freedom for design of complex structures. However, the process-dependent limitations that result in discrepancies between as-designed and as-manufactured shapes are not fully understood. The tradeoffs between infinitely many different ways to approximate design by a manufacturable replica are even harder to characterize.

To support design for AM (DfAM), we quantify local discrepancies introduced by AM processes, identify the detrimental deviations (if any) to the original design intent, and prescribe modifications to the design and/or process parameters to countervail their effects. Our focus in this work will be on *topological analysis*. There is ample evidence in many applications that preserving local topology (e.g., connectivity of beams in a lattice) is important even when slight geometric deviations can be tolerated. We first present a generic method to characterize local topological discrepancies due to material under- and over-deposition in AM, and show how it captures various types of defects in the as-manufactured structures. We use this information to systematically modify the as-manufactured outcome within the limitations of the 3D printer, which often comes at the expense of introducing more geometric deviations (e.g., thickening a beam to avoid disconnection). To reveal the full extent of intrinsic tradeoffs between geometric precision and topological integrity, we use methods from persistent homology on a parameterized family of as-manufactured alternatives based on the interplay between machine resolution and allowance for over-deposition.

1 BACKGROUND

AM continues to reduce the limitations imposed on design by traditional fabrication. However, the notion of 'features' that are in correspondence with traditional manufacturing (as in machining holes, slots, grooves, etc.) is less relevant in AM, opening up an entirely new set of challenges for DfAM. AM parts are often characterized by complex structures to achieve improved performance such as high stiffness per weight for structural support, high surface area per volume for heat transfer, and so on. The as-manufactured structures will differ from the as-designed in ways that are difficult to characterize, quantify, and correct. These deviations will depend on machine and process parameters, as well as the choices hard-coded into the algorithms applied to the as-designed model by each printer's software kit to slice, tessellate, path plan, etc. To prevent unpredictable outcomes/failures, it is desirable to model an as-manufactured structure with controlled deviations from the as-designed in accordance with functional requirements. Among the infinitely many manufacturable alternatives that closely approximate a non-manufacturable design, it is desirable to find the one(s) that minimize(s) deviations that matter the most from a functional standpoint when it is not possible to preserve all of them.

In earlier studies, we have demonstrated methods to identify, visualize, and correct non-manufacturable features in 3D printed parts [Nelaturi et al. 2015; Nelaturi and Shapiro 2015]. Although these methods provide visual and metrological information about the deviations from intended design, they do not offer insight on their topological consequences. Especially with many AM parts (e.g., infill lattices and foams) the topological integrity of the structure has substantial functional significance—commonly even more important than geometric precision. Sometime "small" deviations (from a metric point of view) form the as-designed geometry may lead to changes in topology that lead to compromised function such as broken beams in load-bearing lattice microstructures, covered tunnels in heat exchanger microchannels, filled cavities in porous meta-materials, and so on. On the other hand, if the shape of lattice beams are slightly deformed due to the stair-stepping effect of layered fabrication, it may not matter as much as preserving the connectivity. Similarly, addition/removal of tunnels and cavities can impact performance (e.g., stress concentration under loads) or post-processing (e.g., powder removal after laser sintering).

2 FROM AS-DESIGNED TO AS-MANUFACTURED

Practical limitations on the AM resolution and wall thickness introduce geometric and topological discrepancies between the as-designed target and as-manufactured outcome. Attempting to fabricate designs that have smaller features than the printer's minimum manufacturable feature size will result in disconnected beams, filled holes/tunnels, or hard-to-predict combinations (Fig. 2). The as-manufactured part may eventually look nothing like the as-designed. See, for example, Xometry's results for the "wicked-small cylinder test" using a number of AM processes.

In earlier work [Nelaturi et al. 2015; Nelaturi and Shapiro 2015], we have developed methods to model as-manufactured structures from a knowledge of as-design shape and AM parameters such as manufacturing resolution and wall thickness. We have recently used them to generate AM primitives in hybrid (i.e., combined additive and subtractive) manufacturing processes [Behandish et al. 2018]. The basic model of an as-manufactured shape is obtained by sweeping a minimum manufacturable neighborhood (MMN) along an arbitrary motion that is allowed by the machine's degrees of freedom (DOF). Most 3D printers operate by 3D translations of a printer head over the workpiece as it deposits a blob of material that is modeled by the MMN (e.g., an ellipsoid or a cylindroid) whose radii/height are determined by the printer resolution along the slices and the build direction. Unless the as-designed shape is perfectly sweepable by the MMN via an allowable motion, the as-manufactured shape will differ. The challenge is to find the "best" motion of the head whose sweep of MMN results in a shape as close as possible to the as-designed target. The answer is not unique, as it depends on the notion of closeness; i.e., the criteria based on which the discrepancies are measured.

We use a measure-theoretic approach to define and compute as-manufactured shapes. At every point in the 3D space inside the printer workspace—which represents a hypothetical translational configuration of the printer head—we obtain the overlap measure between the stationary as-designed shape and a MMN instance translated to the said point as the volume of the intersection region between them. This measure can change from zero (no overlap) to the total volume of the MMN (full overlap). We can think of this measure as a real-valued field defined over the configuration space (C−space) of relative translational motions between the two shapes.

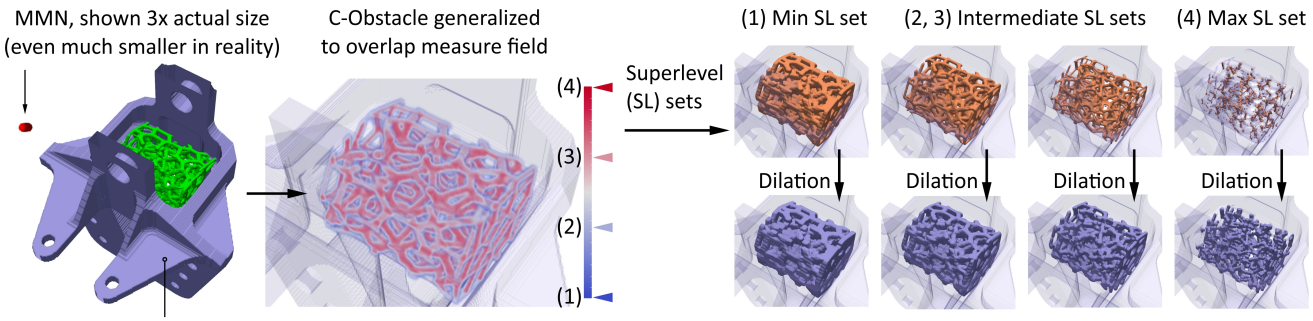


Fig. 1. AM primitives obtained by varying the OMR between a moving MMN and the as-designed shape. The field of overlap values shown in (b) is computed as a convolution of indicator functions of as-designed shape and MMN, whose superlevel-sets are used for to sweep the MMN.

Most commercial 3D printers operate with translational DOFs by printing flat layers on top of each other. In such cases, the printability analysis can be performed in at least two different ways:

(1) **Bulk spatial analysis:** A 3D field of overlap measures is obtained between the 3D as-designed model and a 3D model of the MMN, e.g., a blob of material that is representative of a deposition unit. The layer thickness and build orientation may or may not be encoded into the shape of the MMN. The measure is the volume of intersections between 3D shapes.
(2) **Layer-by-layer analysis:** The as-designed model is sliced along the build orientation into many layers that are a constant distance apart, e.g., equal to printer's known layer thickness. For each 2D as-designed slice, a 2D field of overlap measures is constructed by using a 2D model of the MMN, e.g., nozzle or laser beam cross-section. The measure is the surface area of intersections between 2D shapes.

The former provides a rapid approximation to the latter and allows a high-level analysis in the absence of slicing parameters. The latter provides a more precise analysis, still at a relatively high-level when 1D tool path and parametrization (e.g., G-code) or machine digitization details are unknown or left out to simplify the analysis.

It has been shown that the overlap measure field can be computed cumulatively for all translational motions by a cross-correlation of indicator (i.e., characteristic) functions of the as-designed shape and MMN in 3D (case (1) above) or their slices/cross-sections in 2D (case (2) above). The cross-correlation is computed as a convolution of the first field—indicator function of the as-designed shape—with a reflection of the second field—indicator function of the MMN. The convolution, in turn, can be computed rapidly on high-resolution sampling/voxelization of the part using fast Fourier transforms (FFT) [Kavraki 1995] which are highly optimized for GPU-accelerated implementation. We have shown that this approach extends to group convolutions over the Lie group of rigid motions (combined rotations and translations) [Lysenko et al. 2010]. This is useful with robotic 3D printers with higher-DOF—e.g., to enable support-free 3D printing on adaptively reoriented platforms [Dai et al. 2018]—as well as "multi-tasking" machines for hybrid manufacturing [Yamazaki 2016] that are becoming increasingly popular.

The different superlevel-sets of the cross-correlation field give a family of totally-ordered sets (in terms of containment) in the C−space (i.e., relative translations and/or rotations) $T_\lambda \subseteq \mathrm{C}$:

$$T_\lambda = \left\{\tau \in \mathrm{C} \mid \mu^{\mathrm{d}}[\Omega_{\mathrm{D}} \cap \tau B] \geq \lambda \mu^{\mathrm{d}}[B]\right\}, \quad \text{for some } 0 < \lambda < 1. \quad (1)$$

The members of the family of motions $\{T_\lambda\}_{0<\lambda<1}$ are distinguished by the Lebesgue d−measure—volume for $\mathrm{d} = 3$ (case (1) above) and area for $\mathrm{d} = 2$ (case (2) above)—of the intersection between the as-designed shape and the displaced MMN $\tau B = \{\tau \mathbf{b} \mid \mathbf{b} \in B\}$ that changes between zero and total measure of the MMN. At every choice of the overlap measure ratio (OMR) $\lambda \in (0, 1)$, all configurations of the MMN that lead to at least that OMR are included in the set. For translational motion $\mathrm{C} \cong \mathbb{R}^{\mathrm{d}}$, the extreme superlevel-sets corresponding to $\lambda \to 1^-, 0^+$ are the same as C−space obstacle and its complement (i.e., free space), and can be computed by Minkowski sum/difference, respectively, of the as-designed shape with the MMN's reflection [Nelaturi et al. 2015]:

$$\lim_{\epsilon \to 0^+} T_{(1-\epsilon)} = (\Omega_{\mathrm{D}} \oplus B^{-1}), \quad \text{and} \quad \lim_{\epsilon \to 0^+} T_{(0+\epsilon)} = (\Omega_{\mathrm{D}} \ominus B^{-1}). \quad (2)$$

For other values in between, $(\Omega_{\mathrm{D}} \oplus B^{-1}) \subseteq T_\lambda \subseteq (\Omega_{\mathrm{D}} \ominus B^{-1})$, which is computed as the superlevel-set of the convolution field $(\mathbf{1}_{\Omega_{\mathrm{D}}} * \mathbf{1}_{B^{-1}}) = \mu^{\mathrm{d}}[\Omega_{\mathrm{D}} \cap \tau B]$ for a threshold of $\lambda \|\mathbf{1}_B\|_1 = \lambda \mu^{\mathrm{d}}[B]$, where $\mathbf{1}_{\Omega_{\mathrm{D}}}, \mathbf{1}_B : \mathbb{R}^{\mathrm{d}} \to \{0, 1\}$ are indicator functions of the as-designed shape and MMN, respectively. For general motions including rotational DOFs, these notions generalize to Minkowski products/quotients and group convolutions composed with lifting/projection maps between the Euclidean 3−space and the Lie group $\mathrm{C} = \mathrm{SE(d)} \cong \mathrm{SO(d)} \rtimes \mathbb{R}^{\mathrm{d}}$ [Lysenko et al. 2010].

For each configuration set (i.e., unparameterized motions), the as-manufactured shape is obtained by sweeping the MMN along the set, which is characterized as a morphological dilation $\Omega_{\mathrm{M}} = \mathrm{dil}(T_\lambda^{-1}, B)$ and can also be computed as a Minkowski sum (translation only) or Minkowski product (general rigid motion) and streamlined using convolutions in either C−space. The one-parametric family of as-manufactured alternatives form a totally ordered set (in terms of containment) bounded by the two extremes; namely,

(1) strict under-deposition (UD) ($\lambda \to 1^-$) in which the resulting as-manufactured shape is the unique maximal (in terms of

containment) manufacturable shape strictly contained within the as-designed shape; and

(2) strict over-deposition (OD) ($\lambda \to 0^+$) in which the resulting as-manufactured shape is a generalized offset of the as-designed shape with the MMN, and contains the MMN with a conservative margin.

There is a spectrum of possibilities for the as-manufactured shape when the allowance is relaxed by choosing an arbitrary $\lambda \in (0, 1)$. Every decrease in the OMR grows the as-manufactured shape by a non-uniform offset that depends on the local geometry of both the as-designed shape and MMN. For translational motions, the UD shape is a morphological opening (i.e., dilation of erosion) while the OD shape is a double-offset (i.e., dilation of dilation). The continuous family of as-manufactured shapes in between, parameterized by the OMR, will have small geometric deviations from the as-designed for small MMN. However, they can have dramatically different topological properties (Fig. 1) which is the focus of next section.

3 TOPOLOGICAL ANALYSIS OF DEVIATIONS

We present a novel method to characterize the differences in basic topological properties of an arbitrary as-designed shape Ω_D and an as-manufactured shape Ω_M—including but not necessarily computed by the methods described above—both of which are r-sets [Requicha 1980]. We quantify the discrepancies in terms of the Euler characteristic (EC) and Betti numbers (BN)—denoted and related by $E = b_0 - b_1 + b_2$ in 3D, respectively—of the different components of the symmetric set difference of the two shapes. In 3D, the BN correspond to the number of connected components, tunnels (i.e., through-holes), and voids/cavities, respectively.

It is worthwhile noting that comparing *global* EC/BN between the as-designed and as-manufactured shapes does not provide much insight on local topological discrepancies and what features are responsible for them. For example, beams thinner than the MMN diameter could break when using a under-deposition policy. However, the structure may remain globally connected through other links across (Fig. 2 (a)). Tunnels or cavities may be covered if they are smaller than the MMN diameter when using an OD policy, but they may be part of a larger hole or tunnel that remains topologically intact (Fig. 2 (b)). In more complicated scenarios, multiple holes can merge in strange ways while new holes appear, keeping the total number of holes the same (Fig. 2 (c)). In general, the BN may remain the same and cannot detect local topological discrepancies. Even when they do change, their values provide no insight into what features might have caused those changes and how to fix them by either changing the design or the MMN.

To capture local effects, we investigate the connected components of UD/OD sets (regularized set differences of Ω_D and Ω_M). Each connected component, paired with the portion of its boundary that connects the component to the common region ($\Omega_D \cap \Omega_M$)—called UD/OD 'cut boundaries' (Fig. 3)—is called a 'UD/OD deviation feature' (or 'feature' for short):

$$\bigcup_{1 \le i \le n_U} U_i = (\Omega_D -^* \Omega_M), \quad (U_i \cap U_j) = \emptyset \text{ if } i \neq j, \tag{3}$$

$$\bigcup_{1 \le i \le n_O} O_i = (\Omega_M -^* \Omega_D), \quad (O_i \cap O_j) = \emptyset \text{ if } i \neq j, \tag{4}$$

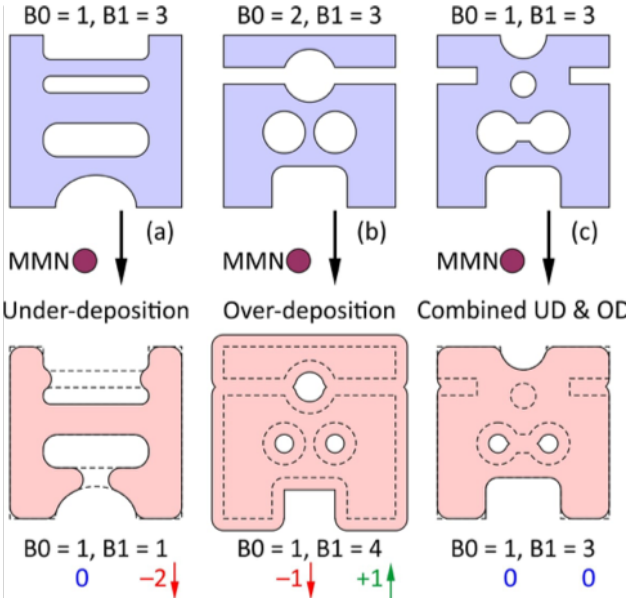


Fig. 2. The global BN (only b_0 and b_1) are shown for a few simple 2D examples of as-designed vs. as-manufactured shapes with UD/OD policies.

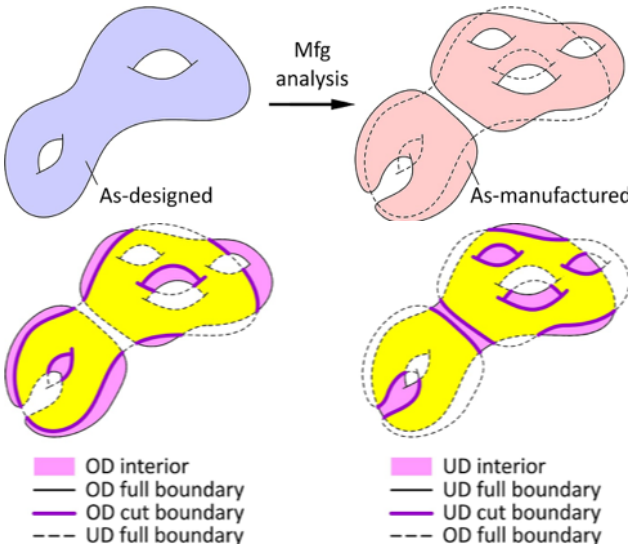


Fig. 3. For arbitrary changes in the global topology of the disturbed shape, we identify the deviations in terms of contributions of local UD/OD features.

where the asterisk implies "regularized" set difference [Tilove and Requicha 1980]. We prove that the total difference in ECs of Ω_D and Ω_M can be obtained from a sum of contributions of individual UD/OD features:

$$E[\Omega_M] - E[\Omega_D] = + \sum_{1 \le i \le n_O} \Big(E[O_i] - E[O_i \cap \partial\Omega_D] \Big) \tag{5}$$

$$- \sum_{1 \le i \le n_U} \Big(E[U_i] - E[U_i \cap \partial\Omega_M] \Big). \tag{6}$$

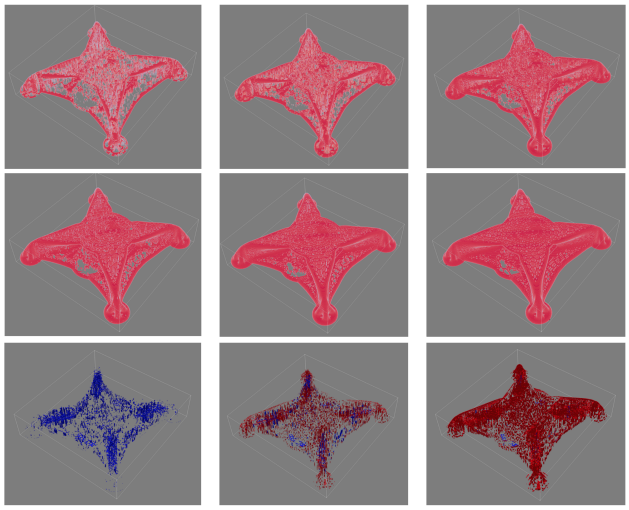


Fig. 4. The OMR superlevel-sets for $\lambda = 1.0, 0.9, 0.8$ (top), corresponding as-manufactured sweep (middle) and their deviation from as-designed (bottom): UD (blue) and OD (red) features; for a topologically optimized quadcopter.

Each feature's contribution (i.e., the terms on the right-hand side of (6)) is computed by subtracting the EC of its cut boundary from the EC of its solid part The proof will be presented in the full paper. A UD/OD feature is called 'simple' if its contribution is zero. If a feature is both simple and simply-connected, the deviation due to its under-/over-deposition bears no topological significance. The features that are simple but not simply-connected may still contribute nonzero BNs (in spite of zero sum into EC) hence must be partitioned further into simply-connected pieces, which makes (6) slightly more complicated. It will be discussed in the full paper but not here due to limited space.

Figure 4 illustrates the manufacturability analysis for a topologically optimized quadcopter frame for different OMR values. A full report of local topological discrepancies is generated (not shown) using (6). We also developed an algorithm for scalable computation (i.e., embarrassing parallelization) of EC for high-resolution sparse voxelization on OpenVDB [Museth 2013]. The details of the algorithm will be given in the full paper.

Once the problematic features are identified (forward problem), they can be repaired by either changing the design or AM process parameters (e.g., localizing OMR) by geometric planning/optimization (inverse problem). Details related to the iterative solution of inverse problem are beyond the scope of this abstract.

4 PERSISTENT MANUFACTURING FEATURES

The above analysis detects *spatial* distribution of topological discrepancies. It also provides useful information to make local changes to the design and/or MMN for eliminating the issues one-at-a-time. However, this analysis cannot detect how important each of different feature contributions are relative to one another. It considers a single as-manufactured outcome; hence provides little insight on their persistence across the spectrum of as-manufactured variants that one can obtain by changing the 3D printer specs or deposition policies—e.g., by changing the shape/size of MMN, threshold on OMR, etc. If we think of changes in a one-parametric family of as-manufactured shapes as a continuous evolution along time-axis, what we need in addition is a *temporal* analysis.

We use persistent homology [Edelsbrunner and Morozov 2013] with two different filtrations:

(1) For a fixed deposition policy (e.g., constant OMR in cross-correlation superlevel-set formulation), a filtration is provided by changing the size of the MMN. This can be done by applying uniform scaling on a fixed-shape MMN, but any other shape parameterization that corresponds to a realistic family of growing MMNs for one or more 3D printers can be used.
(2) For a fixed MMN shape/size (e.g., fixed 3D printer specs), a filtration is provided by changing the deposition policy. This can be done by using the OMR (between 0 and 1) as the filtration parameter, but any other filtering that produces a total ordering can be used.

More recent results in persistent homology with multi-variate filtrations [Cerri et al. 2013] can be applied to analyze the simultaneous changes in both (1) MMN shape/size and (2) deposition policy. For now, we focus on a single-parametric filtration that changes one parameter while keeping the other fixed. To speed up the relatively expensive computation, we run persistence analysis only on local UD/OD features that were identified problematic in the previous section. Details and results will be presented in the full paper.

REFERENCES

Behandish, M., S. Nelaturi, and J. de Kleer
2018. Automated process planning for hybrid manufacturing. *Computer-Aided Design*, 102:115–127.

Cerri, A., B. D. Fabio, M. Ferri, P. Frosini, and C. Landi
2013. Betti numbers in multidimensional persistent homology are stable functions. *Mathematical Methods in the Applied Sciences*, 36(12):1543–1557.

Dai, C., C. C. Wang, C. Wu, S. Lefebvre, G. Fang, and Y.-J. Liu
2018. Support-free volume printing by multi-axis motion. *ACM Transactions on Graphics (TOG)*, 37(4):134.

Edelsbrunner, H. and D. Morozov
2013. Persistent homology: Theory and practice.

Kavraki, L. E.
1995. Computation of configuration-space obstacles using the fast Fourier transform. *IEEE Transactions on Robotics and Automation*, 11(3):408–413.

Lysenko, M., S. Nelaturi, and V. Shapiro
2010. Group morphology with convolution algebras. In *Proceedings of the 14th ACM symposium on solid and physical modeling*, Pp. 11–22. ACM.

Museth, K.
2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics (TOG)*, 32(3):27.

Nelaturi, S., W. Kim, and T. Kurtoglu
2015. Manufacturability feedback and model correction for additive manufacturing. *Journal of Manufacturing Science and Engineering*, 137(2):021015.

Nelaturi, S. and V. Shapiro
2015. Representation and analysis of additively manufactured parts. *Computer-Aided Design*, 67:13–23.

Requicha, A. A. G.
1980. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):1–78.

Tilove, R. B. and A. A. G. Requicha
1980. Closure of Boolean operations on geometric entities. *Computer-Aided Design*, 12(5):219–220.

Yamazaki, T.
2016. Development of a hybrid multi-tasking machine tool: Integration of additive manufacturing technology with CNC machining. *Procedia CIRP*, 42:81–86.

Condensation for the Approximate Nearest Neighbor rule

Alejandro Flores V.* David M. Mount†

Abstract

The problem of Nearest Neighbor (NN) condensation aims to reduce the training set of a NN rule classifier while maintaining its classification accuracy. Although many condensation techniques have been proposed, few bounds on the amount of reduction achieved are known. Moreover, most techniques focus on preserving the classification accuracy on exact NN queries, ignoring the effect of condensation on approximate NN queries. In this paper, we present one of the first theoretical guarantees for condensation algorithms. We propose an algorithm called α-MSS, and provide upper-bounds on the size of the selected subset. Furthermore, we present sufficient conditions to correctly classify query points using approximate NN search on α-MSS.

1 Introduction

In non-parametric classification, a training set P is given, consisting of n points in a metric space $(\mathcal{X}, \mathsf{d})$. Each point $p \in \mathsf{P}$ is also given a label $l(p)$, indicating its membership within one of a set of discrete classes. Given an *unlabeled* query point $q \in \mathcal{X}$, the goal of a *classifier* is to predict q's label (i.e., to classify q) using the training set P. The *Nearest Neighbor* (NN) *rule* is one such classification technique, classifying a query point q with the label of its closest point in P according to the metric d.

While the NN rule exhibits good classification accuracy, both experimentally and theoretically [11, 4, 5], it is often criticized due to its high space and time complexities. Clearly, P must be stored to answer NN queries, and the time required for such queries depends, to a large degree, on the size and dimensionality of the data. These drawbacks motivate the following question: is it possible to replace the training set P with a significantly smaller subset without affecting the classification accuracy under the NN rule? This process is called *Nearest Neighbor* (NN) *condensation*.

Preliminaries Given any point $p \in \mathsf{P}$, define an *enemy* to be any point in P of different class as p. The *nearest enemy* (NE) of p, denoted $\mathsf{NE}(p)$, is the closest such point, and its distance from p, called the NE *distance*, is denoted $\mathsf{d}_{\mathsf{NE}}(p) = \mathsf{d}(p, \mathsf{NE}(p))$. Similarly, denote the NN distance as $\mathsf{d}_{\mathsf{NN}}(p) = \mathsf{d}(p, \mathsf{NN}(p))$.

When P is a point set in Euclidean space, consider the Delaunay triangulation of P. Any point $p \in \mathsf{P}$ with at least one enemy neighbor in this triangulation is called a *border point*, and otherwise is called an *internal point*.

A subset $\mathsf{R} \subseteq \mathsf{P}$ is said to be *consistent* if and only if $\forall\ p \in \mathsf{P}$ its nearest neighbor in R is strictly closer than its nearest enemy in R. Intuitively, R is consistent *iff* every point of P is correctly classified using under the NN rule over R. The NN *condensation problem* involves finding an (ideally small) *consistent* subset of P.

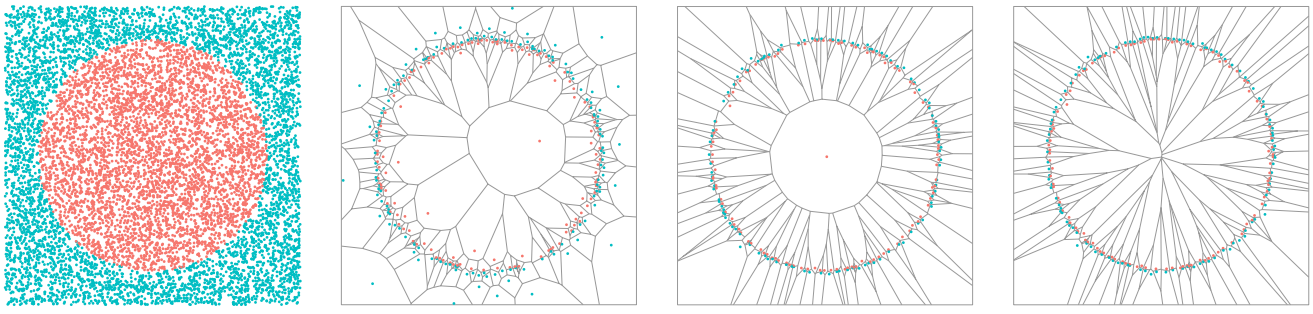
Related work There are other criteria for condensation. A subset $\mathsf{R} \subseteq \mathsf{P}$ is *selective* if and only if $\forall\ p \in \mathsf{P}$ its NN in R is closer to p than its NE in P. Clearly selectivity implies consistency, as the NE distance in R of any point is at least its NE distance in P.

In the Euclidean case, another natural criteria is known as *Voronoi condensation* [14]. It consists of selecting the subset of all border points of P, as these points completely characterize the boundaries between sets of points of different classes. Note that, while a consistent subset can only guarantee the correct classification of points of P, Voronoi condensation guarantees the correct classification of any query point. For the planar case, an output-sensitive algorithm was proposed [3] with $\mathcal{O}(n \log k)$ time complexity, where k is the number of border points of P. Unfortunately, it is not known how to generalize this algorithm to higher dimensions, and a straightforward algorithm for Voronoi condensation would be impractical in high-dimensional spaces.

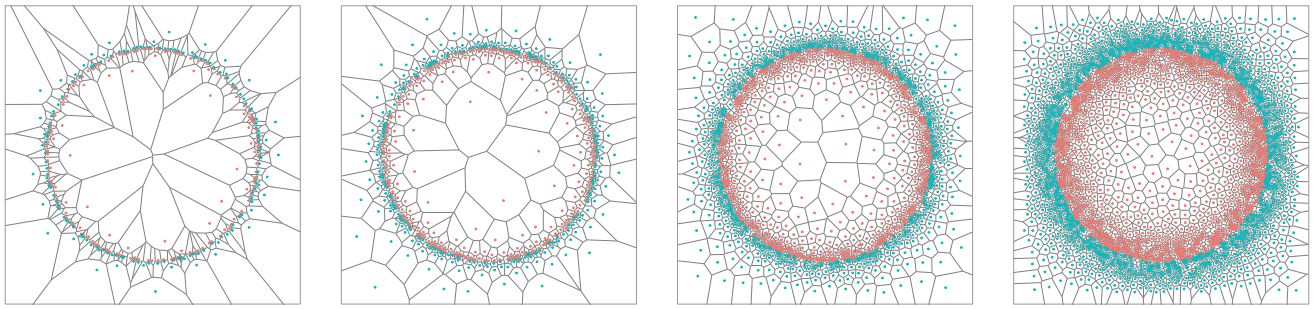
In general, it has been shown that the problems of computing consistent and selective subsets of minimum cardinality are both NP-complete [15, 16]. Thus, most research on the problem has focused on the performance of heuristics for finding subsets with these properties. For comprehensive surveys, see [12, 13, 9].

CNN (Condensed Nearest Neighbor) [7] was the first algorithm proposed for computing consistent subsets. Even though it has been widely used in the literature, CNN suffers from several drawbacks: its running time is cubic in the worst-case, and the resulting subset is *order-dependent*, meaning that the result is determined by the order in which points are considered by the algorithm. Alternatives include FCNN (Fast CNN) [1] and MSS (Modified Selective Subset) [2], which produce consistent and selective subsets respectively. Both

*University of Maryland, College Park, afloresv@cs.umd.edu
†University of Maryland, College Park, mount@cs.umd.edu



(a) Set P (10^4 pts) (b) CNN (281 pts) (c) FCNN (222 pts) (d) MSS (233 pts)



(e) 0.1-MSS (300 pts) (f) 0.5-MSS (540 pts) (g) 2-MSS (1395 pts) (h) 5-MSS (2894 pts)

Figure 1: An illustrative example of the subsets selected by CNN, FCNN, MSS, and α-MSS (with $\alpha = \{0.1, 0.5, 2, 5\}$), from an initial training set P of 10^4 points. While most algorithms focus on selecting border points, or points near the decision boundaries of P, α-MSS also selects internal points in a systematic way.

algorithms run in $\mathcal{O}(n^2)$ worst-case time, and are order-independent. These algorithms are considered the state-of-the-art in the NN condensation problem, subject to achieving these properties. Unfortunately, to the best of our knowledge, no bounds are known for the size of the subsets generated by any of these heuristics.

More recently, an approximation algorithm called NET [6] was proposed, along with almost matching hardness lower bounds. NET produces an γ-net of P, with γ equal to the minimum NE distance in P, which results in a consistent subset. In general, such scheme allows very little room for condensation, as only a few points can be covered with γ-balls, and therefore removed. Thus, while NET has good worst-case performance, the resulting subset can be too large to be of any practical value.

Drawbacks of NN condensation In general, NN condensation algorithms focus on selecting border points, or points close to the decision boundaries. Specially, as illustrated in Figure 1(c) and (d), this is the case with state-of-the-art algorithms like FCNN and MSS. By definition, these border points completely characterize the decision boundaries of P, and therefore, are key in maintaining the classification accuracy of the NN rule after condensation.

However, in practice, nearest neighbors are not computed exactly, but rather *approximately*. Given $\varepsilon > 0$, an ε-ANN query returns any point whose distance from the query point is within a factor of $(1+\varepsilon)$ times the true NN distance. If the condensation is performed without consideration of the approximation error in NN queries, points may be misclassified. This notion is formalized in [10] as the *chromatic density* of a query point, defined as $\delta_q = \frac{\mathsf{d_{NE}}(q) - \mathsf{d_{NN}}(q)}{\mathsf{d_{NN}}(q)}$. If $\delta_q > \varepsilon$, q will always be correctly classified by ε-ANN queries. Therefore, by removing internal points, these heuristics can significantly reduce the *chromatic density* of a query point, and thus, decrease the classification accuracy after condensation, when using ANN queries.

Contributions In this paper, we present theoretical guarantees on both new and existing condensation techniques. The following is a summary of our results.

- We propose α-MSS, a parameterized version of MSS designed to address the drawbacks of existing condensation techniques.
- We describe sufficient conditions to correctly classify query points using ANN queries on α-MSS.
- We provide an upper-bound on the size of α-MSS for doubling spaces. In Euclidean space, the upper-bound is further improved, reducing the dependency on the dimensionality for α-MSS and MSS.

2 Approximate NN Condensation

Let's consider a state-of-the-art heuristic algorithm for the problem of NN condensation, known as MSS or Modified Selective Subset (see Algorithm 1). This algorithm is rather simple; the points of P are examined in increasing order with respect to their NE distance, and any point that fails to satisfy the conditions required of a selective subset (defined above) is added to the resulting subset.

Algorithm 1: Modified Selective Subset

Input: Initial training set P
Output: Condensed training set MSS $\subseteq$ P

```
Let {p_i}_{i=1}^n be the points of P sorted in increasing order of NE distance d_NE(p_i)
MSS ← ∅
foreach p_i ∈ P, where i = 1...n do
    if ∀ r ∈ MSS, d(p_i, r) ≥ d_NE(p_i) then
        MSS ← MSS ∪ {p_i}
return MSS
```

This algorithm tends to select border points, or points close to the decision boundaries of P (see Figure 1(d)). As described before, excluding internal points from the selection, affects the classification accuracy when performing ANN queries after condensation. Therefore, we propose to adapt MSS to address these drawbacks.

2.1 The α-MSS algorithm

Our algorithm is a parameterized version of MSS, called α-MSS. Given a parameter $\alpha \geq 0$, the underlying idea is that every point removed from P will *always* be correctly classified using α-ANN queries on the selected subset (see Algorithm 2).

Algorithm 2: α-Modified Selective Subset

Input: Initial training set P, and value $\alpha \geq 0$
Output: Condensed training set α-MSS $\subseteq$ P

```
Let {p_i}_{i=1}^n be the points of P sorted in increasing order of NE distance d_NE(p_i)
α-MSS ← ∅
foreach p_i ∈ P, where i = 1...n do
    if ∀ r ∈ α-MSS, (1+α)·d(p_i, r) ≥ d_NE(p_i) then
        α-MSS ← α-MSS ∪ {p_i}
return α-MSS
```

From the algorithm's definition, clearly 0-MSS equals MSS. Likewise, when α goes to ∞, the only way to satisfy the given condition is to include all points of P. Therefore, ∞-MSS equals P.

Just like MSS, α-MSS meets some basic properties that make it comparable with other state-of-the-art algorithms for NN condensation.

Theorem 1 *α-MSS is a selective (therefore consistent) subset of P, can be computed in worst-case $\mathcal{O}(n^2)$ time, and it's order-independent.*

2.2 Guarantees on Classification Accuracy

By design, we know that for any point $p \in$ P, the NN of p in α-MSS is at least $(1+\alpha)$ times closer than its NE in P. Meaning the chromatic density of $p \in$ P is at least α. This observation is key in order to analyze the chromatic density of other points, after condensation.

Theorem 2 *Consider a query point $q \in \mathcal{X}$ of chromatic density δ_q with respect to P and chromatic density δ'_q with respect to α-MSS. Then,*

$$\delta'_q \geq \frac{\alpha\delta_q - 2}{\delta_q + \alpha + 3}$$

Corollary 1 *Let $q \in \mathcal{X}$ be a query point with chromatic density $\delta_q = \Omega(1/(\alpha - \varepsilon))$ with respect to P, for $\alpha > \varepsilon$ upper-bounded by some constant. Then q is correctly classified using by an ε-ANN query on α-MSS.*

This implies that, by fixing the values for α and ε, we know the sufficient conditions for a query point to be *always* correctly classified by ε-ANN queries over α-MSS. On the contrary, from the following result, we can fix the lower bounds on the chromatic densities of the query points we want to correctly classify (both before and after condensation), and calculate the value of α needed.

Corollary 2 *Let $q \in \mathcal{X}$ be a query point with chromatic density $\delta_q > \varepsilon$ with respect to P, for some value of ε upper-bounded by a constant. (By definition, q is correctly classified by an ε-ANN query on P.) Consider some value $\varepsilon' \leq \varepsilon$, and set $\alpha = \Omega(1/(\varepsilon - \varepsilon'))$. Then, the chromatic density of q with respect to α-MSS is $\delta'_q > \varepsilon'$, implying that q is also correctly classified by an ε-ANN query on α-MSS.*

3 Upper-bounds on condensation size

One of the most significant shortcomings in research on practical condensation techniques is the lack of theoretical results on the sizes of the selected subsets. Typically, the performance of these heuristics has been established experimentally.

We establish our bounds with respect to the size of a well-known and structured solution: the set of all NE points of P. Additionally, these bounds depend on the *spread* Δ of P, which is defined to be the ratio of the largest to smallest pairwise distances in P, and the *doubling dimension* $\mathsf{ddim}(\mathcal{X})$ of the metric space [8].

Theorem 3 (Size of α-MSS in doubling spaces) *Consider a point set* P *of spread* Δ *in a metric space* $(\mathcal{X}, \mathsf{d})$ *with bounded doubling dimension* $\mathsf{ddim}(\mathcal{X})$*. Let* κ *be the number of* NE *points of* P*. Then,*

$$|\alpha\text{-MSS}| \leq \kappa \left\lceil \frac{\log \Delta}{\log \frac{\alpha+2}{\alpha+1}} \right\rceil \mathcal{O}\left(\alpha^{\mathsf{ddim}(\mathcal{X})+1}\right)$$

Thus, for constant α and $\mathsf{ddim}(\mathcal{X})$, the size of α-MSS is $\mathcal{O}(\kappa \log \Delta)$. This bound follows from a charging argument on each NE point in P, where we consider the number of points in α-MSS whose NE is the same. By partitioning these points according to their NE distance, we show that points in α-MSS can't be to close. Finally, we use a known packing argument for doubling spaces [6], and the upper-bound follows.

The Euclidean case When the underlying space is Euclidean, the arguments used on doubling spaces can be further improved. These results rely on a packing argument dependent on the minimum angle between any two selected points who share the same NE.

Theorem 4 (Size of MSS in Euclidean space) *Let* $\mathsf{P} \subseteq \mathbb{R}^d$ *be a point set in Euclidean space. Let* κ *be the number of* NE *points of* P*. Then,*

$$|\text{MSS}| = \kappa\ \mathcal{O}((3/\pi)^{d-1})$$

For constant d, the size of MSS is $\mathcal{O}(\kappa)$. Moreover, we can show that the number of NE points of P is at most the number of border points of P (*i.e.*, $\kappa \leq k$). Recall that *Voronoi condensation* computes the set of all border points of P, which is of size k. Then, the size of MSS is $\mathcal{O}(k)$ for constant d.

Theorem 5 (Size of α-MSS in Euclidean space) *Let* $\mathsf{P} \subseteq \mathbb{R}^d$ *be a point set in Euclidean space of spread* Δ*. Let* κ *be the number of* NE *points of* P*. Then,*

$$|\alpha\text{-MSS}| \leq \kappa \left\lceil \frac{\log \Delta}{\log \frac{(\alpha+1)^2}{\alpha(\alpha+2)}} \right\rceil \mathcal{O}(\alpha^{d-1})$$

For constant α and d, the size of α-MSS is $\mathcal{O}(\kappa \log \Delta)$.

References

[1] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *Knowledge and Data Engineering, IEEE Transactions on*, 19(11):1450–1464, 2007.

[2] R. Barandela, F. J. Ferri, and J. S. Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.

[3] D. Bremner, E. Demaine, J. Erickson, J. Iacono, S. Langerman, P. Morin, and G. Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. In F. Dehne, J.-R. Sack, and M. Smid, editors, *Algorithms and Data Structures: 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003. Proceedings*, pages 451–461, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, Jan. 1967.

[5] L. Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):75–78, 1981.

[6] L.-A. Gottlieb, A. Kontorovich, and P. Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, pages 370–378, 2014.

[7] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, Sept. 1968.

[8] J. Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2012.

[9] N. Jankowski and M. Grochowski. Comparison of instances selection algorithms I. Algorithms survey. In *Artificial Intelligence and Soft Computing-ICAISC 2004*, pages 598–603. Springer, 2004.

[10] D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry*, 17(3):97 – 119, 2000.

[11] C. J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.

[12] G. Toussaint. Open problems in geometric methods for instance-based learning. In J. Akiyama and M. Kano, editors, *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2002.

[13] G. Toussaint. Proximity graphs for nearest neighbor decision rules: Recent progress. In *Progress, Proceedings of the 34th Symposium on the INTERFACE*, pages 17–20, 2002.

[14] G. T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen. The application of Voronoi diagrams to non-parametric decision rules. *Proc. 16th Symposium on Computer Science and Statistics: The Interface*, pages 97–108, 1984.

[15] G. Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224–233, New York, NY, USA, 1991. ACM.

[16] A. V. Zukhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recog. Image Anal.*, 20(4):484–494, Dec. 2010.

Exact fast parallel intersection of large 3-D triangular meshes *(extended abstract)*[*]

Salles V. G. Magalhães
Universidade Fed. de Viçosa
Viçosa, MG, Brazil
salles@ufv.br

W. Randolph Franklin
Rensselaer Polytechnic Institute
Troy, NY, USA
mail@wrfranklin.org

Marcus V.A. Andrade
Universidade Federal de Viçosa
Minas Gerais, Brasil
marcus.ufv@gmail.com

ABSTRACT

We present 3D-EPUG-Overlay, a fast, exact, parallel, memory-efficient, algorithm for computing the intersection between two large 3-D triangular meshes with geometric degeneracies. Applications include CAD/CAM, CFD, GIS, and additive manufacturing. 3D-EPUG-Overlay combines 5 separate techniques: multiple precision rational numbers to eliminate roundoff errors during the computations; Simulation of Simplicity to properly handle geometric degeneracies; simple data representations and only local topological information to simplify the correct processing of the data and make the algorithm more parallelizable; a uniform grid to efficiently index the data, and accelerate testing pairs of triangles for intersection or locating points in the mesh; and parallel programming to exploit current hardware. 3D-EPUG-Overlay is up to 101 times faster than LibiGL, and comparable to QuickCSG, a parallel inexact algorithm. 3D-EPUG-Overlay is also more memory efficient. In all test cases 3D-EPUG-Overlay's result matched the reference solution. It is freely available for nonprofit research and education at https://github.com/sallesviana/MeshIntersection . The full version of this paper was presented at the 27th International Meshing Roundtable, October 1–5, 2018, Albuquerque, NM, USA. It is currently online at https://project.inria.fr/imr27/files/2018/09/1016.pdf.

1 INTRODUCTION

The classic problem of intersecting two 3-D meshes has been a foundational component of CAD systems for some decades. However, as data sizes grow, and parallel execution becomes desirable, the classic algorithms and implementions now exhibit some problems.

1. *Roundoff errors.* Floating point numbers violate most of the axioms of an algebraic field, e.g., $(a + b) + c \neq a + (b + c)$. These arithmetic errors cause topological errors, such as causing a point to be seen to fall on the wrong side of a line. Those inconsistencies propagate, causing, e.g., nonwatertight models. Heuristics exist to ameliorate the problem, and they work, but only up to a point. Larger datasets mean a larger probability of the heuristics failing.
2. *Special cases (geometric degeneracies).* These include a vertex of one object incident on the face of another object. In principle, simple cases could be enumerated and handled. However, some widely available software fails.
3. Another problem is that current data structures are too complex for easy parallelization. Efficient parallelization prefers simple regular data structures, such as structures of arrays of plain old datatypes That disparages pointers, linked lists, and trees.

Some components of 3D-EPUG-Overlay have been presented earlier. PinMesh preprocesses a 3D mesh so that point locations can be performed quickly [24]. EPUG-Overlay overlays 2D meshes [23].

Background: Kettner et al [21] studied failures caused by roundoff errors in geometric problems. They also showed situations where epsilon-tweaking failed. Snap rounding arbitrary precision segments into fixed-precision numbers, Hobby [19], can also generate inconsistencies and deform the original topology. Variations attempting to get around these issues include de Berg et al [6], Hersberger [18], and Belussi et al [2]. Controlled Perturbation (CP), Melhorn [27], slightly perturbs the input to remove degeneracies such that the geometric predicates are correctly evaluated even using floating-point arithmetic. Adaptive Precision Floating-Point, Shewchuk [30], exactly evaluates predicates (e.g. orientation tests) using the minimum necessary precision.

Exact Geometric Computation (EGC), Li [22], represents mathematical objects using algebraic numbers to perform computations without errors. However this is slow.

One technique to accelerate algorithms based on exact arithmetic is to employ arithmetic filters and interval arithmetic, Pion et al [29], such as embodied in CGAL [4].

Current freely available implementations: One technique for overlaying 3-D polyhedra is to convert the data to a volumetric representation (voxelization), perhaps stored as an octree, Meagher [26], and then perform the overlay using the converted data. For exactly computing overlays, a common strategy is to use indexing to accelerate operations such as computing the triangle-triangle intersection. For example, Franklin [12] uses a uniform grid to intersect two polyhedra, Feito et al [11] and Mei et al [28] use octrees, and Yongbin et al [32] use Oriented Bounding Boxes trees (OBBs) to intersect triangulations.

Another algorithm that does not guarantee robustness is QuickCSG, Douze et al [9], which is designed to be extremely efficient. QuickCSG employs parallel programming and a *k-d*-tree index to accelerate the computation. However, it does not handle special cases (it assumes vertices are in general position), and does not handle the numerical non-robustness from floating-point arithmetic, Zhou et al [33]. To reduce errors caused by special cases, QuickCSG allows the user to apply random numerical perturbations to the input, but this has no guarantees.

Although small errors may sometimes be acceptable, they accumulate if several inexact operations are performed in sequence. This gets even worse in CAD and GIS where it is common to compose operations. For use when exactness is required, Hachenberger et al [17] presented an algorithm for computing the exact intersection of Nef polyhedra.

[*]

Bernstein et al [3] presented an algorithm that tries to achieve robustness in mesh intersection by representing the polyhedra using binary space partitioning (*BSP*) trees with fixed-precision coordinates. It can intersect two such polyhedra by only evaluating fixed-precision predicates. However, in 3D, the BSP representation often has superlinear size, because the partitioning planes intersect so many objects. Also, converting BSPs back to more widely used representations (such as triangular meshes) is slow and inexact.

Recently, Zhou [33] presented an exact and parallel algorithm for performing booleans on meshes. The key is to use the concept of winding numbers to disambiguate self-intersections on the mesh. That algorithm is freely available and distributed in the LibiGL package, Jacobson et al [20]. Its implementation employs CGAL's exact predicates. The triangle-triangle intersection computation is also accelerated using CGAL's bounding-box-based spatial index. LibiGL is not only exact, but also much faster than Nef Polyhedra. However, it is still slower than fast inexact algorithms such as QuickCSG.

2 OUR TECHNIQUES

Our solution to the above problems combines the following five techniques (aka engineering subsystems, tricks).

Big rational numbers: Representing a number as the quotient of two integers, each represented as an array of groups of digits, is a classic technique. The fundamental limitation is that the number of digits grows exponentially with the depth of the computation tree. Our relevant computation comprises comparing the intersection of two lines defined by their endpoints against a plane defined by three vertices. So, this growth in precision is quite tolerable.

The challenges in going from an academic theory to a workable implementation are harder. Many C++ implementations of new data structures automatically construct new objects on a global heap, and assume the construction cost to be negligible. That is false for parallel programs processing large datasets. Constructing and destroying heap objects has a superlinear cost in the number of objects on the heap. Parallel modifications to the heap must be serialized. Therefore we carefully construct our code to minimize the number of times that a rational variable needs to be constructed or enlarged. This includes minimizing the number of temporary variables needed to evaluate an expression. Furthermore, we use interval arithmetic as a filter to determine when evaluation with rationals is necessary.

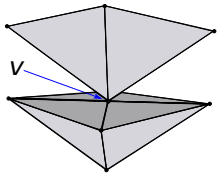


Figure 1: Difficult test case for 3-D point location.

Simulation of Simplicity: Simulation of Simplicity (SoS), Edelsbrunner et al [10], addresses the problem that, "sometimes, even careful attempts at capturing all degenerate cases leave hard-to-detect gaps", Yap [31]. Figure 1 is a challenging case. It consists of two pyramids with central vertices incident at a common vertex v. v is non-manifold and is on 8 faces, 4 from each pyramid. It is not easy to determine which of the 8 faces should intersect the ray that would be run up from v in order to locate v. In the subproblem of point location, RCT gets this point location case wrong; PinMesh is correct because of SoS, Magalhães et al [24]. SoS symbolically perturbs coordinates by adding infinitesimals of different orders. The result is that there are no longer any coincidences, e.g., three points are never collinear.

Minimal topology: A sufficient representation of a 3-D mesh comprises the following: (a) the array of vertices, (v_i), where each $v_i = (x_i, y_i, z_i)$. (b) the array of tetrahedra or other polyhedra, t_i, used solely to store properties such as density, and (c) the array of augmented oriented triangular faces (f_i), where $f_i = (v_{i1}, v_{i2}, v_{i3}, t_{i1}, t_{i2})$. The tetrahedron or polyhedron t_{i1} is on the positive side of the face $f_i = (v_{i1}, v_{i2}, v_{i3})$; t_{i2} on the negative. It is unnecessary to store any further relations, such as from face to adjacent face, from vertex to adjacent face, edge loops, or face shells.

Note that there are no pointers or lists; we need only several structures of arrays. If the tetrahedra have no properties, then the tetrahedron array does not need to exist, so long as the tetrahedra, which we are not storing explicitly, are consistently sequentially numbered. The point is to minimize what types of topology need to be stored.

Uniform grid: The uniform grid, Akman et al [1], Franklin et al [13–15] is used as an initial cull so that, when two objects are tested for possible intersection, then the probability of intersecting is bounded below by a positive number. Therefore, the number of pairs of objects tested for intersection that do not actually intersect is linear in the number that do intersect. Thus the expected execution time is linear in the output size.

A careful concrete implementation of this abstraction is critical. We tested several choices; details are in Magalhães [7]. We also tested an octree, but our uniform grid implementation is much faster. We also used a second level grid for some cells. This allowed us to use an approximation to determine which faces intersected each cell: enclosing oblique faces with a box and then marking all the cells intersecting that box, which is more cells than necessary.

3 3-D MESH INTERSECTION

3D-EPUG-Overlay exactly intersects 3-D meshes. Its input is two triangular meshes M_0 and M_1. Each mesh contains a set of 3-D triangles representing a set of polyhedra. The output is another mesh where each represented polyhedron is the intersection of a polyhedron from M_0 with another one from M_1. The key is the combination of five techniques described later. Extra details are in Magalhães et al [7, 8, 23–25].

Data representation: The input is a pair of triangular meshes in 3-D (E^3). Both meshes must be watertight and free from self-intersections. The polyhedra may have complex and nonmanifold topologies, with holes and disjoint components. The two meshes may be identical, which is an excellent stress test, because of all the degeneracies.

There are two types of output vertices: input vertices, and intersection vertices resulting from intersections between an edge of one mesh and a triangle of the other. Similarly, there are two types of output triangles: input triangles and triangles from retesselation. The first contains only input vertices while the second may contain vertices generated from intersections created during the retesselation of input triangles. An intersection vertex is represented by an

edge and the intersecting triangle. For speed, its coordinates are cached when first computed.

Retesselation of faces that were split was implemented with orientation predicates, Magalhães [7], which reduced to implementing 164 functions. A Wolfram Mathematica script was developed to create the code for all the predicates.

4 EXPERIMENTS

We conducted extensive experiments to measure 3D-EPUG-Overlay's performance on large datasets, to compare it to other implementations, to validate its output, and to try to make it fail on degenerate cases.

3D-EPUG-Overlay was implemented in C++ and compiled using g++ 5.4.1. For better parallel scalability, the gperftools Tcmalloc memory allocator [16], was employed. Parallel programming was provided by OpenMP 4.0, multiple precision rational numbers were provided by GNU GMPXX and arithmetic filters were implemented using the Interval_nt number type provided by CGAL for interval arithmetic. The experiments were performed on a workstation with 128 GiB of RAM and dual Intel Xeon E5-2687 processors, each with 8 physical cores and 16 hyper-threads, running Ubuntu Linux 16.04. We evaluated 3D-EPUG-Overlay, by comparing it against three state-of-the-art algorithms:

1. *LibiGL* [33], which is exact and parallel,
2. *Nef Polyhedra* [4], which is exact, and
3. *QuickCSG* [9], which is fast and parallel, but not exact, and does not handle special cases.

Our experiments showed that 3D-EPUG-Overlay is fast, parallel, exact, economical of memory, and handles special cases.

Experiments were performed with a variety of non self-intersecting and watertight meshes. The datasets and lengthy results are detailed in the full paper.

We compared 3D-EPUG-Overlay against other three algorithms. 3D-EPUG-Overlay was up to 101 times faster than LibiGL. The only test cases where the times spent by LibiGL were similar to the times spent by 3D-EPUG-Overlay were during the computation of the intersections of a mesh with itself (even in these test cases 3D-EPUG-Overlay was still faster than LibiGL). In this situation, the intersecting triangles from the two meshes are never in general position, and thus the computation has to frequently trigger the SoS version of the predicates, which we haven't not optimized yet. In the future, we intend to optimize this.

However, LibiGL also repairs meshes (by resolving self-intersections) during the intersection computation, which 3D-EPUG-Overlay does not attempt.

Because of the overhead of Nef Polyhedra and since it is a sequential algorithm, CGAL was always the slowest. When computing the intersections, 3D-EPUG-Overlay was up to 1,284 times faster than CGAL. The difference is much higher if the time CGAL spends converting the triangular mesh to Nef Polyhedra is taken into consideration: intersecting meshes with 3D-EPUG-Overlay was up to 4,241 times faster than using CGAL to convert and intersect the meshes.

While 3D-EPUG-Overlay was faster than QuickCSG in most of the test cases (mainly the largest ones), in others QuickCSG was up to 20% faster than 3D-EPUG-Overlay. The relatively small performance difference between 3D-EPUG-Overlay and an inexact method (that was specifically designed to be very fast) indicates that 3D-EPUG-Overlay presents good performance allied with exact results. Besides reporting errors during some experiments QuickCSG also failed in some situations where errors were not reported.

Finally, we also performed experiments with tetra-meshes. Each tetrahedron in these meshes is considered to be a different object and, thus, the output of 3D-EPUG-Overlay is a mesh where each object represents the intersection of two tetrahedra (from the two input meshes). These meshes are particularly hard to process because of their internal structure, which generates many triangle-triangle intersections. For example, during the intersection of the *Neptune* with the *Neptune translated* datasets (two meshes without internal structure), there are 78 thousand pairs of intersecting triangles and the resulting mesh contains 3 million triangles. On the other hand, in the intersection of *518092_tetra* (a mesh with 6 million triangles and 3 million tetrahedra) with *461112_tetra* (a mesh with 8 million triangles and 4 million tetrahedra) there are 5 million pairs of intersecting triangles and the output contains 23 million triangles.

To the best of our knowledge, LibiGL, CGAL and QuickCSG were not designed to handle meshes with multi-material and, thus, we couldn't compare the running time of 3D-EPUG-Overlay against them in these test cases.

We also evaluated the peak memory usage of each algorithm. 3D-EPUG-Overlay was: almost always smaller than LibiGL, with the difference increasing as the datasets became larger; smaller than QuickCSG in every case where QuickCSG returned the correct answer; and much smaller than CGAL. A typical result was the intersection of Neptune (4M triangles) with Ramesses (1.7M triangles): 3D-EPUG-Overlay used 2.6GB, LibiGL used 6.7GB, and CGAL 84GB. The largest example that 3D-EPUG-Overlay processed, 518092Tetra (6M triangles) with 461112Tetra (8.5M triangles) used 43GB. Magalhães [7] contains detailed results.

Correctness evaluation: 3D-EPUG-Overlay was developed on a solid foundation (i.e., all computation is exact and special cases are properly handled using Simulation of Simplicity) in order to ensure correctness. However, perhaps its implementation has errors? Therefore, we performed extensive experiments comparing it against LibiGL (as a reference solution). We employed the Metro tool, Cignoni et al [5], to compute the Hausdorff distances between the meshes being compared. Metro is widely employed, for example, to evaluate mesh simplification algorithms by comparing their results with the original meshes.

In every test, the difference between 3D-EPUG-Overlay and LibiGL was reported as 0. In some situations the difference between LibiGL and CGAL was a small number (maximum 0.0007% of the diagonal of the bounding-box). We guess this is because the exact results are stored using floating-point variables, and different strategies are used to round the vertices to floats and write them to the text file. QuickCSG, on the other hand, generated errors much larger than CGAL: in the worst case, the difference between QuickCSG output and LibiGL was 0.13% of the diagonal of the bounding-box). Magalhães [7] contains detailed results.

Visual inspection: We also visually inspected the results using MeshLab. Even though small changes in the coordinates of the vertices cannot be easily identified by visual inspection (and even the

program employed for displaying the meshes may have roundoff errors), topological errors (such as triangles with reversed orientation, self-intersections, etc) often stand out.

Even when QuickCSG did not report a failure, results were frequently inconsistent, with open meshes, spurious triangles or inconsistent orientations.

Rotation invariance: We also validated 3D-EPUG-Overlay by verifying that its result does not change when the input meshes are rotated. In all the experiments Metro reported that the resulting meshes were equal (i.e., the Hausdorff distance was 0.000000) to the corresponding ones obtained without rotation. In addition, we intersected several meshes with a rotated version of themselves. This is a notoriously difficult case for CAD systems because the large number of intersections and small triangles. In every experiment the Hausdorff distance between the two outputs was 0.000000. That is, we can quickly process cases that can crash CAD systems.

5 SUMMARY

3D-EPUG-Overlay is an algorithm and implementation to intersect a pair of 3D triangular meshes. It is simultaneously the fastest, free from roundoff errors, handles geometric degeneracies, parallelizes well, and is economical of memory. The source code, albeit research quality, is freely available for nonprofit research and education at https://github.com/sallesviana/MeshIntersection . We have extensively tested it for errors; we encourage others to test it. It is a suitable subroutine for larger systems such as 3D GIS or CAD systems. Computing other kinds of overlays, such as union, difference, and exclusive-or, would require modifying only the classification step. We expect that 3D-EPUG-Overlay could easily process datasets that are orders of magnitude larger, with hundreds of millions of triangles. Finally, 3D-EPUG-Overlay has not nearly been fully optimized, and could be made much faster.

REFERENCES

[1] Varol Akman, Wm Randolph Franklin, Mohan Kankanhalli, and Chandrasekhar Narayanaswami. 1989. Geometric Computing and the Uniform Grid Data Technique. *Computer Aided Design* 21, 7 (1989), 410–420.

[2] Alberto Belussi, Sara Migliorini, Mauro Negri, and Giuseppe Pelagatti. 2016. Snap Rounding with Restore: An Algorithm for Producing Robust Geometric Datasets. *ACM Trans. Spatial Algorithms and Syst.* 2, 1, Article 1 (March 2016), 36 pages. https://doi.org/10.1145/2811256

[3] Gilbert Bernstein and Don Fussell. 2009. Fast, exact, linear booleans. *Eurographics Symp. on Geom. Process.* 28, 5 (2009), 1269–1278. https://doi.org/10.1111/j.1467-8659.2009.01504.x

[4] CGAL. 2018. Computational Geometry Algorithms Library. (2018). Retrieved 2018-09-09 from https://www.cgal.org

[5] P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring Error on Simplified Surfaces. *Comput. Graph. Forum* 17, 2 (June 1998), 167–174. https://doi.org/10.1111/1467-8659.00236

[6] Mark de Berg, Dan Halperin, and Mark Overmars. 2007. An intersection-sensitive algorithm for snap rounding. *Computational Geometry* 36, 3 (Apr. 2007), 159–165.

[7] Salles Viana Gomes de Magalhães. 2017. *Exact and parallel intersection of 3D triangular meshes.* Ph.D. Dissertation. Rensselaer Polytechnic Institute.

[8] Salles V. G. de Magalhães, W. Randolph Franklin, Marcus V. A. Andrade, and Wenli Li. 2015. An efficient algorithm for computing the exact overlay of triangulations. In *25th Fall Workshop on Computational Geometry*. U. Buffalo, New York, USA. (extended abstract).

[9] Matthijs Douze, Jean-Sébastien Franco, and Bruno Raffin. 2015. *QuickCSG: Arbitrary and faster boolean combinations of n solids.* Ph.D. Dissertation. Inria-Research Centre, Grenoble–Rhône-Alpes, France.

[10] Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM TOG* 9, 1 (1990), 66–104.

[11] F.R. Feito, C.J. Ogayar, R.J. Segura, and M.L. Rivero. 2013. Fast and accurate evaluation of regularized Boolean operations on triangulated solids. *Computer-Aided Design* 45, 3 (2013), 705 – 716. https://doi.org/10.1016/j.cad.2012.11.004

[12] Wm Randolph Franklin. 1982. Efficient polyhedron intersection and union. In *Proc. Graphics Interface*. Toronto, 73–80.

[13] Wm Randolph Franklin. 1984. Adaptive Grids for geometric operations. *Cartographica* 21, 2–3 (Summer – Autumn 1984), 161–167. monograph 32–33.

[14] Wm Randolph Franklin, Narayanaswami Chandrasekhar., Mohan Kankanhalli, Manoj Seshan, and Varol Akman. 1988. Efficiency of uniform grids for intersection detection on serial and parallel machines. In *New Trends in Computer Graphics (Proc. Computer Graphics International'88)*, Nadia Magnenat-Thalmann and D. Thalmann (Eds.). Springer-Verlag, 288–297.

[15] Wm Randolph Franklin, Chandrasekhar Narayanaswami, Mohan Kankanhalli, David Sun, Meng-Chu Zhou, and Peter YF Wu. 1989. Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines. In *Proceedings of Auto Carto 9: Ninth International Symposium on Computer-Assisted Cartography*. Baltimore, Maryland, 100–109.

[16] Sanjay Ghemawat and Paul Menage. 2015. TCMalloc: Thread-Caching Malloc. http://goog-perftools.sourceforge.net/doc/tcmalloc.html (retrieved on 13 Nov 2016). (15 Nov 2015).

[17] Peter Hachenberger, Lutz Kettner, and Kurt Mehlhorn. 2007. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Comupt. Geom.* 38, 1 (Sept. 2007), 64–99.

[18] John Hershberger. 2013. Stable snap rounding. *Comput. Geom.* 46, 4 (May 2013), 403–416.

[19] John D. Hobby. 1999. Practical segment intersection with finite precision output. *Comput. Geom.* 13, 4 (1999), 199–214. http://dblp.uni-trier.de/db/journals/comgeo/comgeo13.html#Hobby99

[20] Alec Jacobson, Daniele Panozzo, et al. 2016. *libigl: A Simple C++ Geometry Processing Library.* http://libigl.github.io/libigl/ (Retrieved on 18 Oct 2017).

[21] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. 2008. Classroom Examples of Robustness Problems in Geometric Computations. *Comput. Geom. Theory Appl.* 40, 1 (May 2008), 61–78. https://doi.org/10.1016/j.comgeo.2007.06.003

[22] C. Li. 2001. *Exact geometric computation: theory and applications,*. Ph.D. Dissertation. Department of Computer Science, Courant Institute - New York University.

[23] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. 2015. Fast exact parallel map overlay using a two-level uniform grid. In *4th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial)*. Bellevue WA USA. https://doi.org/10.1145/2835185.2835188

[24] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. 2016. PinMesh – Fast and exact 3D point location queries using a uniform grid. *Computer & Graphics Journal, special issue on Shape Modeling International 2016* 58 (Aug. 2016), 1–11. https://doi.org/10.1016/j.cag.2016.05.017 (online 17 May). Awarded a reproducibility stamp, http://www.reproducibilitystamp.com/.

[25] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, Wenli Li, and Maurício Gouvêa Gruppi. 2016. Exact intersection of 3D geometric models. In *Geoinfo 2016, XVII Brazilian Symposium on GeoInformatics*. Campos do Jordão, SP, Brazil.

[26] Donald J. Meagher. 1982. Geometric Modelling Using Octree Encoding. *Computer Graphics and Image Processing* 19 (June 1982), 129–147.

[27] Kurt Mehlhorn, Ralf Osbild, and Michael Sagraloff. 2006. Reliable and Efficient Computational Geometry Via Controlled Perturbation.. In *ICALP (1)* (2006-07-03) *(Lecture Notes in Computer Science)*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.), Vol. 4051. Springer, 299–310. http://dblp.uni-trier.de/db/conf/icalp/icalp2006-1.html#MehlhornOS06

[28] Gang Mei and John C. Tipper. 2013. Simple and Robust Boolean Operations for Triangulated Surfaces. *CoRR* abs/1308.4434 (2013). http://arxiv.org/abs/1308.4434

[29] Sylvain Pion and Andreas Fabri. 2011. A generic lazy evaluation scheme for exact geometric computations. *Sci. Comput. Program.* 76, 4 (Apr. 2011), 307 – 323.

[30] Jonathan Richard Shewchuk. 1997. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discret. & Comput. Geom.* 18, 3 (Oct. 1997), 305–363.

[31] Chee Keng Yap. 1988. Symbolic treatment of geometric degeneracies. In *System Modelling and Optimization: Proc. 13th IFIP Conference*, Masao Iri and Keiji Yajima (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 348–358. https://doi.org/10.1007/BFb0042803

[32] Jing Yongbin, Wang Liguan, Bi Lin, and Chen Jianhong. 2009. Boolean Operations on Polygonal Meshes Using OBB Trees. In *ESIAT 2009*, Vol. 1. IEEE, 619–622.

[33] Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh Arrangements for Solid Geometry. *ACM Trans. Graph.* 35, 4, Article 39 (July 2016), 15 pages.

Soft Subdivision Motion Planning for Complex Planar Robots*

Bo Zhou† Yi-Jen Chiang‡ Chee Yap§

1 Introduction

Motion planning is widely studied in robotics [7, 8, 3]. Many planners are heuristic, i.e., without a priori guarantees of their performance (see below for what we mean by guarantees). In this paper, we are interested in non-heuristic algorithms for the **basic planning problem**: this basic problem considers only kinematics and the existence of paths. The robot R_0 is fixed, and the input is a triple (α, β, Ω) where α, β are the start and goal configurations of R_0, and $\Omega \subseteq \mathbb{R}^d$ is a polyhedral environment in $d = 2$ or 3. The algorithm outputs an Ω-avoiding path from α to β if one exists, and `NO-PATH` otherwise. See Figure 1 for some rigid robots, and also Figure 2 for our GUI interface for path planning.

The basic planning problem ignores issues such as the optimality of paths, robot dynamics, planning in the time dimension, non-holonomic constraints, and other considerations of a real scenario. Despite such an idealization, the solution to this basic planning problem is often useful as the basis for finding solutions that do take into account the omitted considerations. E.g., given a kinematic path, we can plan a smooth trajectory with a homotopic trace.

The algorithms for this basic problem are called "planners." In theory, it is possible to design *exact* planners because the basic path planning is a semi-algebraic (non-transcendental) problem. Even when such algorithms are available, exact planners have relatively high complexity and are non-adaptive, even in the plane (see [10]). So we tend to see inexact implementations of exact algorithms, with unclear guarantees. When fully explicit algorithms are known, exact implementation of exact planners is possible using suitable software tools such as the `CGAL` library [5].

In current robotics [8, 3], practical algorithms that come with some guarantees may be classified as either resolution-based or sampling-based. Each offers a completeness guarantee: roughly speaking, *if there exists a path*, then

– **resolution completeness** says that a path will be found if the resolution is fine enough;

– **sampling completeness** says that a path will be found with high probability if "enough" random samples are taken.

But notice that if there is no path, these criteria are silent; indeed, such algorithms would not halt except by artificial cut-offs. Thus a major effort in the last 20 years of sampling research has been devoted to the so-called "Narrow Passage" problem. It is possible to view this problem as a manifestation of the **Halting Problem** for the sampling approaches: how can the algorithm halt when there is no path? (A possible approach to address this problem might be to combine sampling with exact computation, as in [11].)

Motivated by such issues, as well as trying to avoid the need for exact computation, we in [13, 15] introduced the following replacement for resolution complete planners: a **resolution-exact planner** takes an extra input parameter $\epsilon > 0$ in addition to (α, β, Ω), and it always halts and outputs either an Ω-avoiding path from α to β or `NO-PATH`. The output satisfies this condition: there is a constant $K > 1$ depending on the planner, but independent of the inputs, such that:

– if there is a path of clearance $K\epsilon$, it must output a path;

– if there is no path of clearance ϵ/K, it must output `NO-PATH`.

Notice that if the optimal clearance lies between $K\epsilon$ and ϵ/K, then the algorithm may output either a path or `NO-PATH`. So there is output indeterminacy. Note that the traditional way of using ϵ is to fix $K = 1$, killing off indeterminacy. Unfortunately, this also leads us right back to exact computation which we had wanted to avoid. We believe that indeterminacy is a small price to pay in exchange for avoiding exact computation [13]. The practical efficiency of resolution-exact algorithms is demonstrated by implementations of planar robots with 2, 3 and 4 degrees of freedom (DOF) [13, 9, 14], and also 5-DOF spatial robots [6]. All these robots perform in real-time in non-trivial environments. In view of the much stronger guarantees of performance, resolution-exact algorithms might

*The conference version of this paper has appeared in *Proc. European Symposium on Algorithms (ESA '18)*, pages 73:1–73:14, August 2018, and is available on-line at http://cse.poly.edu/chiang/esa18-final.pdf.

†Department of Computer Science and Engineering, New York University, Brooklyn, NY; bz387@nyu.edu.

‡Department of Computer Science and Engineering, New York University, Brooklyn, NY; chiang@nyu.edu.

§Department of Computer Science, New York University, New York, NY; yap@cs.nyu.edu.

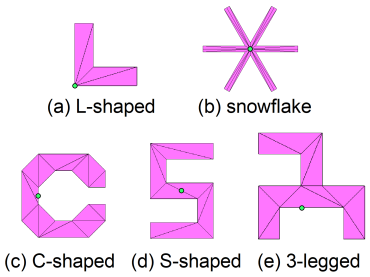


Figure 1: Some rigid planar robots ((a)-(b): star-shaped; (c)-(e): general shaped).

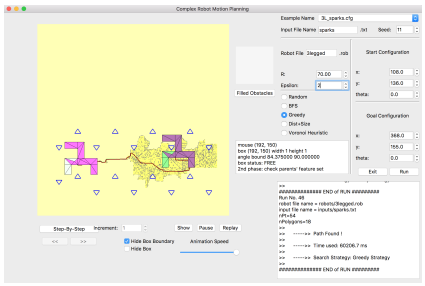


Figure 2: GUI interface for planner for a 3-legged robot.

reasonably be expected to have a lower efficiency compared to sampling algorithms. Surprisingly, no such tradeoffs were observed: resolution-exact algorithms consistently outperform sampling algorithms. Our 2-link robot [9, 14] was further generalized to have thickness (a feat that exact methods cannot easily duplicate), and can satisfy a non-self-crossing constraint, all without any appreciable slowdown. Finally, these planners are more general than the basic problem: they all work for parametrized families $R_0(t_1, t_2 \ldots)$ of robots, where t_i's are robot parameters. All these suggest the great promise of our approach.

What is new in this paper. In theoretical path planning, the algorithms often considered simple robots like discs or line segments. In this paper, we consider robots of complex shape, which are more realistic models for real-world robots. We call them "complex robots" (where the complexity comes from the robot geometry rather than from the degrees of freedom). We focus on planar robots that are rigid and connected. Such a robot can be represented by a compact connected polygonal set $R_0 \subseteq \mathbb{R}^2$ whose boundary is an m-sided polygon, i.e., an m-gon. Informally, we call R_0 a "complex robot" if it is a non-convex m-gon for "moderately large" values of m, say $m \geq 5$. By this criterion, all the robots in Figure 1 are "complex." According to [17], no exact algorithms for $m > 3$ have been implemented; in this paper, we have robots with $m = 18$. To see why complex robots may be challenging, recall that the free space of such robots may have complexity $O((mn)^3 \log(mn))$ (see [1]) when the robot and environment have complexity m and n, respectively. Even with m fixed, this can render the algorithm impractical. For instance, if $m = 10$, the algorithm may slow down by 3 orders of magnitude. But our subdivision approach does not have to compute the entire free space before planning a path; hence the worst-case cubic complexity of the free space is not necessarily an issue.

More importantly, we show that the complexity of our new method grows only linearly with m. To achieve this, we exploit a remarkable property of soft predicates called "decomposability." We show how an arbitrary complex robot can be decomposed (via triangulation that may introduce new vertices) into an ensemble of "nice triangles" for which soft predicates are easy to implement. As we see below, there is a significant difference between a single triangle and an ensemble of triangles. *In consequence of our new techniques, we can now routinely construct resolution-exact planners for any reasonably complex robot provided by a user.* This could lead to a flowering of experimentation algorithmics in this subfield.

Technically, it is important to note that the previous soft predicate construction for a triangle robot in [13, 16] requires that the rotation center, i.e., the origin of the (rotational) coordinate system, be chosen to be the circumcenter of the triangle. But for our new soft predicates the triangles in the triangulation of the complex robot cannot be treated in the same way. This is because all the triangles of the triangulation must share a **common origin**, to serve as the rotation center of the robot. To ensure easy-to-compute predicates, we introduce the notion of a "nice triangulation" relative to a chosen origin: all triangles must be "nice" relative to this origin. These ideas apply for arbitrary complex robots, but we also exploit the special case of star-shaped robots to achieve stronger results.

Figure 2 shows our experimental setup for complex robots. A demo showing the real-time performance of our algorithms is found in the video clip available through this web link: https://cs.nyu.edu/exact/gallery/complex/complex-robot-demo.mp4.

Remark. Although it is not our immediate concern to address noisy environments and uncertainties, it is clear that our work can be leveraged to address these issues. E.g., users can choose $\epsilon > 0$ to be correlated with the uncertainty in the environment and the precision of the robot sensors. By using weighted Voronoi diagrams [2], we can achieve practical planners that have obstacle-dependent clearances (larger clearance for "dangerous" obstacles).

Table 1: Running Our Planner (R: radius of the robot's circumcircle around its rotation center; P?: path found? (Yes/No); Time is in s; S-shaped*: thin version).

Exp#	Robot	Envir.	R	ϵ	α	β	P?	Time
0	L-shaped	gateway	50	2	(18, 98, 340°)	(458,119,270°)	Yes	10.106
1	L-shaped	gateway	50	4	(18, 98, 340°)	(458,119,270°)	No	8.431
2	snowflake	sparks	56	2	(108, 136, 0°)	(358, 155, 0°)	Yes	17.846
3	snowflake	sparks	56	2	(108, 136, 0°)	(358, 155, 180°)	Yes	3.370
4	S-shaped	sparks	74	4	(132, 80, 90°)	(333, 205, 90°)	Yes	34.284
5	S-shaped	sparks	74	4	(132, 80, 90°)	(333, 205, 60°)	No	57.371
6	3-legged	sparks	70	2	(108, 136, 0°)	(368, 155, 0°)	Yes	41.745
7	L-shaped	corridor	68	2	(75, 420, 0°)	(370, 420, 0°)	Yes	4.012
8	L-shaped	corridor	68	3	(75, 420, 0°)	(370, 420, 0°)	Yes	1.926
9	L-shaped	corridor	68	5	(75, 420, 0°)	(370, 420, 0°)	Yes	2.684
10	L-shaped	corridor-L	68	5	(75, 420, 0°)	(370, 420, 0°)	No	2.908
11	L-shaped	corridor-L	68	3	(75, 420, 0°)	(370, 420, 0°)	Yes	2.255
12	C-shaped	corridor-S	80	4	(80, 450, 0°)	(380, 450, 0°)	Yes	26.200
13	S-shaped	maze	38	2	(38, 38, 0°)	(474, 474, 90°)	No	90.097
14	S-shaped*	maze	38	2	(38, 38, 0°)	(474, 474, 90°)	Yes	79.518

Table 2: Comparing with OMPL ("#": Exp#; "Time/P?": our run time (in s)/path found? (Y/N). Each OMPL method: Average Time (in s)/Standard Deviation/Success Rate, over 10 runs).

#	Time/P?	PRM	RRT	EST	KPIECE
0	10.106/Y	**4.18**/2.53/1	42.13/38.49/1	76.22/110.44/0.9	300/0/0
2	17.846/Y	**9.22**/6.82/1	210.41/144.25/0.3	271.75/89.31/0.1	240.00/126.47/0.2
3	**3.370**/Y	300/0/0	300/0/0	300/0/0	300/0/0
4	34.284/Y	**5.93**/7.20/1	217.33/134.53/0.3	300/0/0	300/0/0
5	**57.371**/N	300/0/0	300/0/0	300/0/0	300/0/0
6	41.745/Y	**2.72**/4.89/1	154.22/141.77/0.5	104.32/78.10/0.7	3.16/4.28/1
8	1.926/Y	0.63/0.55/1	300/0/0	3.02/4.71/1	**0.41**/0.28/1
11	2.255/Y	**1.49**/0.84/1	300/0/0	241.24/124.88/0.2	1.58/1.47/1
12	26.200/Y	**3.16**/4.21/1	300/0/0	172.506/120.38/0.7	93.88/88.03/0.8
13	**90.097**/N	300/0/0	300/0/0	300/0/0	300/0/0
14	79.518/Y	300/0/0	236.72/106.44/0.3	300/0/0	**39.81**/91.57/0.9

2 Experimental Results

We have implemented our approaches in `C/C++` with `Qt` GUI platform. The software and data sets are freely available from the web site for our open-source `Core Library` [4]. All experiments are reproducible as targets of Makefiles in `Core Library`. Our experiments are on a PC with one 3.4GHz Intel Quad Core i7-2600 CPU, 16GB RAM, nVidia GeForce GTX 570 graphics and Linux Ubuntu 16.04 OS. The results are summarized in Table 1 and Table 2. Table 1 is concerned only with the behavior of our complex robots; Table 2 gives comparisons with the open-source OMPL library [12]. The robots are as shown in Figure 1.

We select some interesting experiments to analyze characteristic behavior of our planner. Please see Table 1 and the video (`https://cs.nyu.edu/exact/gallery/complex/complex-robot-demo.mp4`). In Exp0-1, we show how the parameter ϵ affects the result. With a narrow gateway, when we change ϵ from 2 to 4, the output changes from a path to `NO-PATH` for the same configuration. In Exp2-3, we observe how the snowflake robot rotates and maneuvers to get from the start to two different goals. For Exp4-5, the difference is in the angles of the goal configuration; in Exp5 this is designed to be an isolated configuration and the planner outputs `NO-PATH` as desired. Exp6 shows how the robot squeezes among the obstacles to move its complex shape through the environment. Exp7-9 use the same L-shaped robot, α, β configurations and the environment; only ϵ varies. The planner can find three totally different paths. When ϵ is small (Exp7), the path is very carefully adjusted to move the robot around the obstacles. When ϵ is larger (Exp8), the planner finds an upper path with a higher clearance. When ϵ is even larger (Exp9), the planner chooses a very safe but much longer path at the bottom. Note that using a larger ϵ usually makes the search faster, since we stop splitting boxes smaller than ϵ, but a longer path can make the search slower. In Exp10-11, we modify the environment of Exp7-9 by putting a large obstacle at the bottom, which forces the robot to find a path at the top. Exp12 uses an environment similar to those in Exp7-11 but with much smaller scattered obstacles. It is designed for the C-shaped robot, which can rotate while having an obstacle in its pocket. Exp13-14 use a challenging environment where the small scattered obstacles force the S-shaped robot to rotate around and only the "thin" version (Exp14, also in Fig. 3 "maze") can squeeze through.

In Table 2 we compare our planner with several sampling algorithms in OMPL: PRM, RRT, EST, and KPIECE.

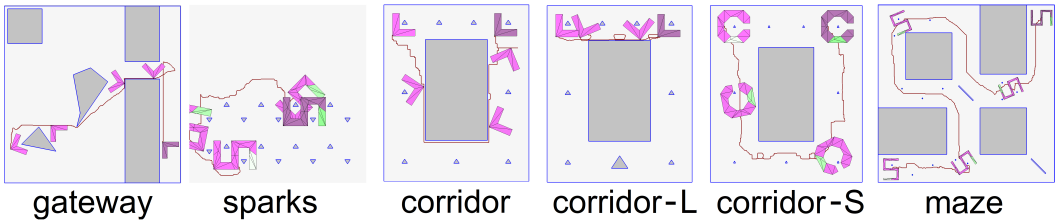


Figure 3: Six Environments in our experiments.

These experiments are correlated to those in Table 1 (see the Exp #). Each OMPL planner is run 10 times with a time limit 300 seconds (default), where all planner-specific parameters use the OMPL default values. We see that for OMPL planners there are often unsuccessful runs and they have to time out even when there is a path. On the other hand, our algorithm consistently solves the problems in a reasonable amount of time, often much faster than the OMPL planners, in addition to being able to report `NO-PATH`.

References

[1] F. Avnaim, J.-D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Geometry and Robotics*, LNCS Vol 391, 1989.

[2] H. Bennett, E. Papadopoulou, and C. Yap. Planar minimization diagrams via subdivision with applications to anisotropic Voronoi diagrams. *Eurographics Symposium on Geometric Processing*, 35(5), 2016.

[3] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[4] Core Library. https://cs.nyu.edu/exact/core_pages/downloads.html.

[5] D. Halperin, E. Fogel, and R. Wein. *CGAL Arrangements and Their Applications*. Springer-Verlag, 2012.

[6] C.-H. Hsu, Y.-J. Chiang, and C. Yap. Rods and rings: Soft subdivision planner for **R**^3 x **S**^2, 2018. Available at http://cse.poly.edu/chiang/rod-ring18.pdf.

[7] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[8] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.

[9] Z. Luo, Y.-J. Chiang, J.-M. Lien, and C. Yap. Resolution exact algorithms for link robots. In *Proc. Intl. Workshop on Algorithmic Foundations of Robotics (WAFR '14)*, volume 107 of *Springer Tracts in Advanced Robotics (STAR)*, pages 353–370, 2015.

[10] V. Milenkovic, E. Sacks, and S. Trac. Robust complete path planning in the plane. In *Proc. Workshop on Algorithmic Foundations of Robotics (WAFR 2012)*, Springer Tracts in Advanced Robotics, vol.86, pages 37–52, 2012.

[11] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin. Motion planning via manifold samples. In *Proc. European Symp. Algorithms (ESA)*, 2011.

[12] I. Şucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. http://ompl.kavrakilab.org.

[13] C. Wang, Y.-J. Chiang, and C. Yap. On Soft Predicates in Subdivision Motion Planning. *Comput. Geometry: Theory and Appl. (Special Issue for SoCG'13)*, 48(8):589–605, Sept. 2015.

[14] C. Yap, Z. Luo, and C.-H. Hsu. Resolution-exact planner for thick non-crossing 2-link robots. In *Proc. Intl. Workshop on Algorithmic Foundations of Robotics (WAFR '16)*, 2016.

[15] C. K. Yap. Soft Subdivision Search in Motion Planning. In A. Aladren et al., editor, *Proc. Workshop on Robotics Challenge and Vision (RCV 2013)*, 2013. Robotics Science and Systems Conference (RSS 2013), Berlin. In arXiv:1402.3213. Full paper: http://cs.nyu.edu/exact/papers/.

[16] C. K. Yap. Soft Subdivision Search and Motion Planning, II: Axiomatics. In *Frontiers in Algorithmics*, volume 9130 of *Lecture Notes in Comp.Sci.*, pages 7–22. Springer, 2015. Plenary Talk at 9th FAW. Guilin, China. Aug 3-5, 2015.

[17] L. Zhang, Y. J. Kim, and D. Manocha. Efficient cell labeling and path non-existence computation using C-obstacle query. *Int'l. J. Robotics Research*, 27(11–12):1246–1257, 2008.

Package Delivery with Trucks and Drones: The Horsefly Problem

Joseph. S. B. Mitchell
Stony Brook University

Gaurish Telang
Stony Brook University

I. Introduction

With recent advances in drone technology and their widespread availability, several companies, including Amazon and UPS, have been exploring the use of drones for delivering packages, potentially with higher throughput at lower cost. *"UPS has also estimated that cutting off just one mile for the routes of each of the company's 66,000 delivery drivers would amount to $50 million in savings. For this reason, UPS is testing drone deliveries, using the top of its vans as a mini-helipad."* [1]. The video in [2] shows this concept in action.

We consider an optimal package delivery problem utilizing a truck and a drone, specified as follows. Let $S = \{p_1, \ldots, p_n\}$ be a given set of n customer sites in $\mathbb{R}^2$. A truck full of packages starts at point s (the depot) and has a delivery drone on its rooftop. The drone can carry one package at a time to a customer, then return to the truck for another package. The truck moves at maximum speed 1, while the drone flies at speed $\varphi \geq 1$; we refer to φ as the *speed ratio*. The goal is to compute a route for the truck and for the drone in order to complete the delivery of all n packages (and have the drone return back to the empty truck) as soon as possible – i.e., we seek to minimize *makespan* of the delivery process. We describe algorithms to address this optimal delivery problem. We give both provable approximation results (an $O(\log n)$-approximation algorithm) and some experimental results comparing some heuristics.

The above problem has been called the *Horsefly* problem [3]. It is easy to see that the Horsefly problem is NP-hard, from the Euclidean TSP. (If the speed ratio $\varphi = 1$, then an optimal solution is for the drone and truck to travel together from site to site on a TSP path.) Fig. 1 shows an example of a routing for a truck and drone for 30 sites, with $\varphi = 5$.

The challenge in computing a solution to the Horsefly problem is to determine both the order of the sites in which they must be serviced by the drone and the rendezvous points of the truck and the drone (the points where the drone lands back on the truck to pick up the next package, and then depart to the next customer).

It is also assumed in the above model that the truck itself cannot make any deliveries: it is always the drone that must deliver a package to every site. (The variant in which the truck/driver also makes deliveries is also of interest but we defer that discussion to a full paper.) We also focus here on the case of the truck being able to move freely within the Euclidean plane; more realistically, the truck may be restricted to move on a network of roads, or in an obstacle-cluttered environment – some of the methods discussed here can apply more generally and will be discussed in a full paper (forthcoming).

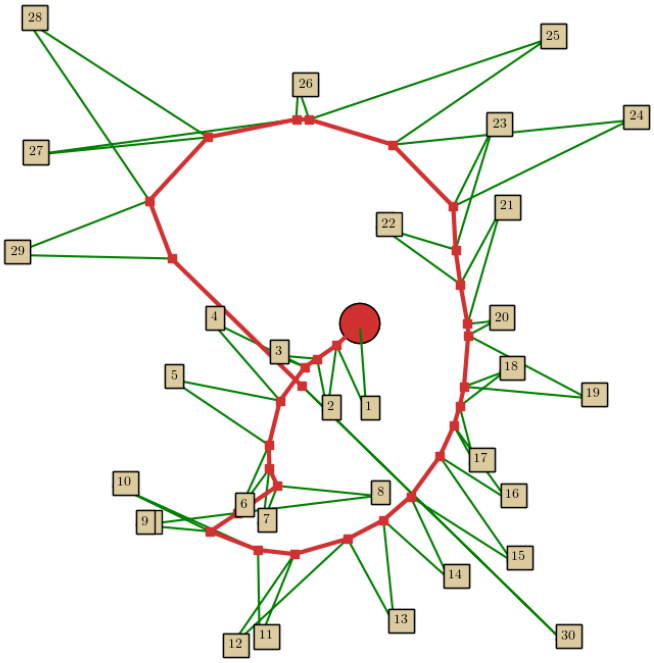


Fig. 1: Delivery with truck and drone, with speed-ratio $\varphi = 5$. The truck and drone travel along the red and green paths, respectively. The large red dot indicates the initial position s of the truck and drone.

II. An $O(\log n)$-Approximation Algorithm

The Horsefly problem in the plane has an $O(\log n)$-approximation algorithm. We briefly sketch the proof. The algorithm is based on computing, via dynamic programming, a least expensive solution of a particular structure: The truck traverses the edges of an orthogonal binary space partition (BSP), while the drone's routes are doubled line segments connecting each customer site p_i to the closest point on the boundary of the BSP face that contains it. The expense of a solution is a weighted sum of the edge lengths: φ times the total edge length of the BSP network (the truck network), plus 1 times the doubled segment lengths for the drone paths. We

argue that an optimal solution to the Horsefly problem can be converted to a BSP-structured solution at a cost of a factor $O(\log n)$ in the makespan. From the optimal BSP-structured network, we can extract optimal routes for the drone and truck, and a feasible delivery schedule, which must be within factor $O(\log n)$ of optimal.

III. Case: Delivery Order is Given

We consider first the case in which the order in which customer sites receive deliveries is given: $(p_1, p_2, \ldots, p_n)$.

We make some simple observations about the structure of an optimal solution:

(1) The truck route and the drone route are polygonal, with vertices at a set of departure/rendezvous points, $s_1 = s, s_2, s_3, \ldots, s_n$, where point s_i is a point along the truck route where the drone departs to deliver the package to customer p_i. (And, thus, s_i is also the point on the truck route where the drone returns to the truck after making the delivery to the customer at point p_{i-1}, for $i \geq 2$.)

(2) The truck and the drone move always at their maximum speeds (1 and φ, respectively).

Given the ordering of the sites, the problem of computing the optimal rendezvous points s_i can be formulated as a convex program, which can be solved by standard non-linear optimization solvers, such as the Sequential Least Squares Programming (SLSQP) solver from [7], which we use in our experiments. In order to speed up this computation, we formulate the problem of computing optimal tours in the L_1 (Manhattan) metric, which becomes a linear program.

Fig. 2 and Fig. 3 show an example using SLSQP versus LP, based on the site-ordering given by a greedy heuristic (described below), in both cases. The SLSQP-based method took about 2 minutes to compute the tour, whereas the LP-based method computed the tour in less than a second. (Both times include the time taken for computing the site ordering by the greedy heuristic.)

To test the approximation properties of the linear programming formulation, we consider the ratio of the lengths of the tours obtained using LP versus the exact SLSQP solver, for different speed ratios on 100 uniformly distributed sites in the unit square $[0, 1] \times [0, 1]$. We perform the experiment on 40 random instances. The initial position of the horse and drone in all experiments was set to the middle of the square $(0.5, 0.5)$. We used the MOSEK [8] optimization package for solving the resulting linear programs.

The ratios of the tour lengths for each run and speed-ratio are plotted in the figure below, for various values of n (horizontal axis).

From the figure it appears that the ratio of the tour-lengths is bounded, for a fixed speed ratio φ. Further the worst-case ratio seems to increase slowly as a function of φ.

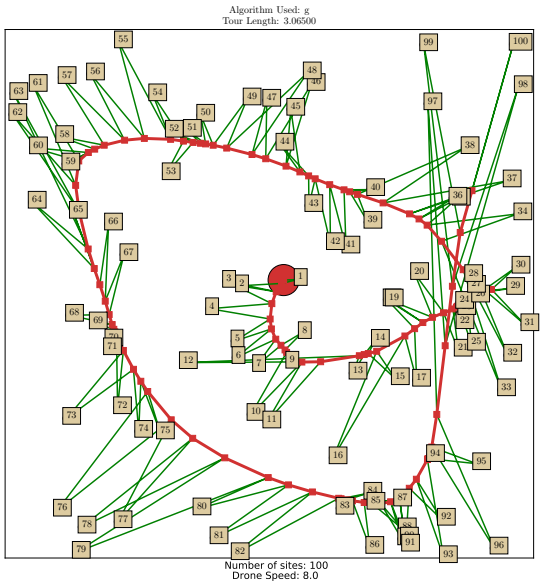


Fig. 2: Tour using SLSQP on site ordering returned by the Greedy heuristic: Tour length is 3.06500

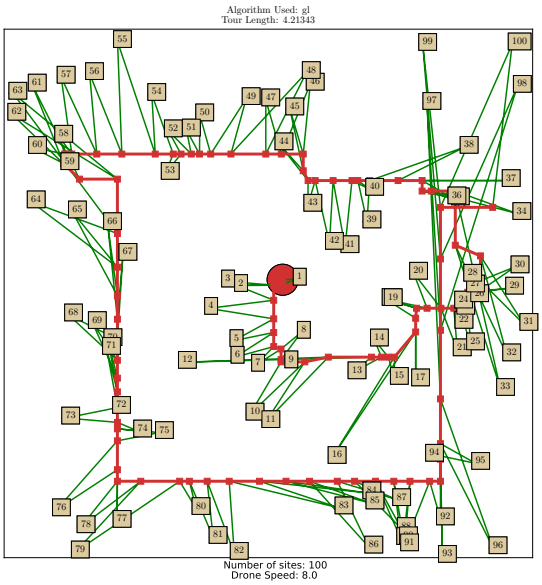
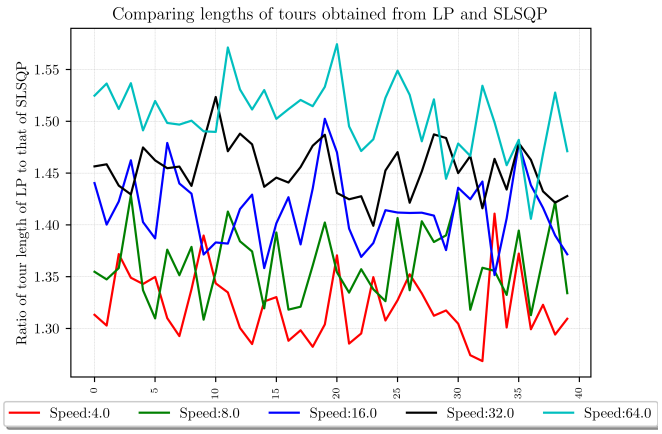


Fig. 3: Tour using LP on site ordering returned by the Greedy heuristic: Tour length is 4.21343

IV. Two Heuristics

A. A Greedy Heuristic

We first introduce a special case of the Horsefly problem, which we call *Collinear-Horsefly*. Here, the objective function is again to minimize the tour-length of the drone, with the additional restriction that the truck must always be moving in a straight line towards the site on the line-segment joining itself and the site, while the drone is also restricted to travelling along the same line segment. See Fig. 4 for an example instance of this problem, for $\varphi = 3$. With this additional



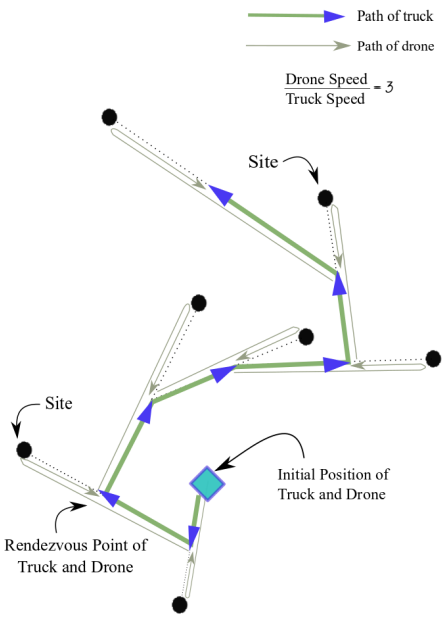


Fig. 4: The Collinear-Horsefly problem.

restriction, the possible rendezvous points for the truck and the drone becomes finite. We show that an optimal (unrestricted) Horsefly solution can be converted to a collinear-Horsefly solution, at a constant factor increase in the makespan.

Similar to the nearest-neighbor insertion heuristic for the standard TSP, a natural greedy strategy can be formulated for the Collinear-Horsefly problem; the truck always moves towards an unvisited site nearest to its current position, while the drone takes off from the truck, services the site, and flies back towards the truck along the line joining the truck and the site again; refer to Fig. 4.

Once the ordering of the sites is determined by the heuristic, we use the fixed-ordering optimization program (using a convex program in the L_2 case, or a linear program in the L_1 case) to compute the best route for the truck (and thus for the drone), using the given ordering.

B. A Clustering-Based Heuristic

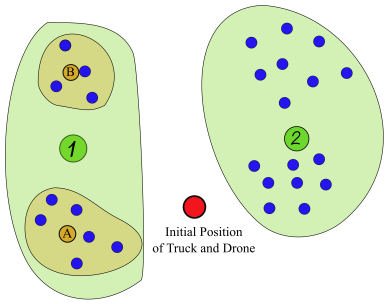


Fig. 5: The first two steps in the k2means heuristic for the Horsefly problem.

We describe another heuristic, which we call the "k2means" heuristic: Given the set S of n sites, we first compute the 2-centers along with associated cluster points for each of the 2-centers. We then use the exact algorithm to decide which center (and hence cluster) to visit first.

In Fig. 5, for instance, we decide that the truck and drone will coordinate to visit all of the sites in the left cluster first and then the right. Within the chosen cluster, we again compute the 2-center and then use the exact algorithm for 2 points to decide which sub-cluster should the truck and drone visit first. For instance, in the example above, the heuristic decides to visit all of the sites associated with sites of the 2-center A and then of the 2-center B.

We continue recursively, for each cluster, until we reach a cluster size of 1 or 2.

In the above figure, once we finish visiting all of the sites in the left cluster, we use the ending point of this subtour as the initial point of the truck and drone to decide the order in which we must visit the sites in the right cluster.

Once we finish calculating the order of the sites according to the heuristic, we discard the computed path, and use the non-linear solver to compute the optimal truck (and drone) path for this ordering.

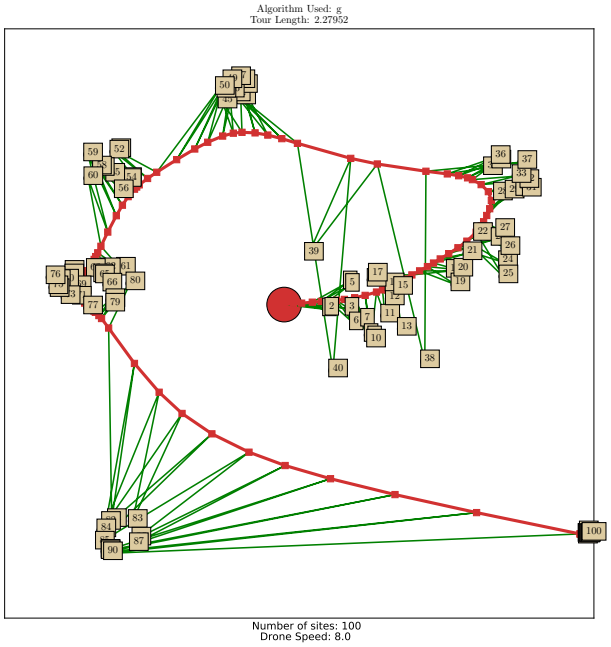
C. An Example

The figure below compares two heuristics on one example of 100 sites distributed inside the unit-square, with $\varphi = 8$.

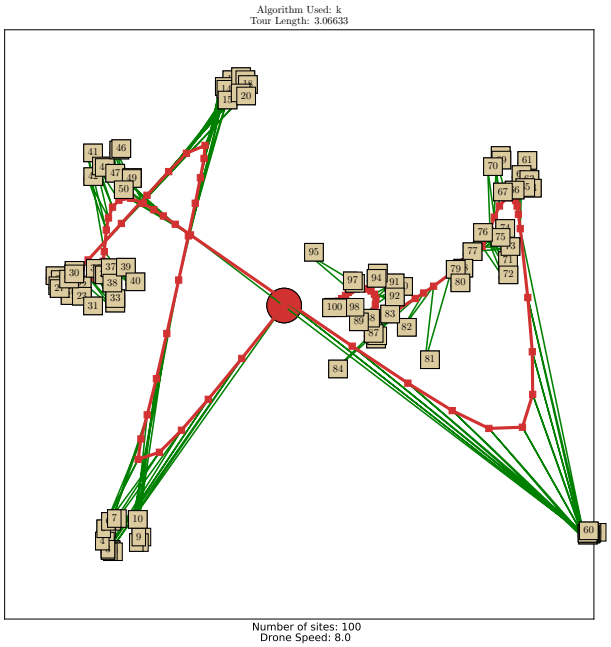
D. Comparing tour-lengths for the two heuristics

Here we notice that the greedy heuristic consistently outperforms the k2means heuristic by a slowly growing function of n. The same behaviour was observed for other speed-ratios.

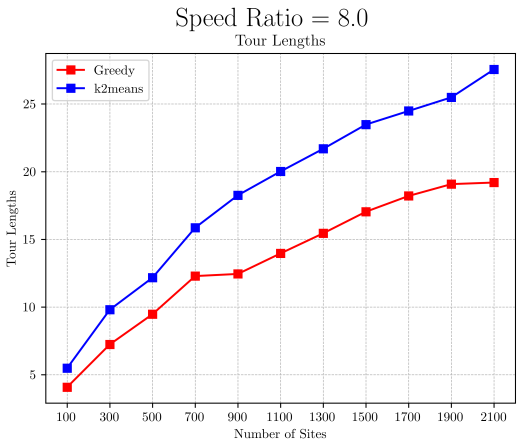
Acknowledgements: This work is part of a collaboration with Sujoy Bhore, John Gunnar Carlsson, Sándor Fekete, Supantha Pandit, and others from the CG Group at Stony Brook. We acknowledge support from DARPA, the National



(a) Tour obtained with the greedy heuristic.



(b) Tour obtained with the k2means heuristic.



Science Foundation (CCF-1526406), and the US-Israel Binational Science Foundation (project 2016116).

REFERENCES

[1] *https://www.businessinsider.com/amazon-and-ups-are-betting-big-on-drone-delivery-2018-3*

[2] UPS tests residential drone delivery. *https://youtu.be/P5hQHBNpd7s*

[3] John Gunnar Carlsson, Siyuan Song *Coordinated Logistics with a Truck and a Drone* Management Science, 2017, *https://doi.org/10.1287/mnsc.2017.2824*

[4] Ha, Q. M. and Deville, Y. and Pham, Q. D. and Hà, M. H. *On the Min-cost Traveling Salesman Problem with Drone*, *arXiv preprint arXiv:1512.01503*, 2015

[5] Campbell, James F and Sweeney, DC and II, Zhang J, *Strategic design for delivery with trucks and drones* Technial Report, 2017

[6] Chao, I.M. *A tabu search method for the truck and trailer routing problem* Computers & Operations Research, Elsevier 2002 volume 29, number 1, pages 33-51

[7] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html

[8] https://www.mosek.com/

[9] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Topological Distance Between Nonplanar Transportation Networks

Ahmed Abdelkader[1], Geoff Boeing[2], Brittany Terese Fasy[3], David L. Millman[3]
[1] University of Maryland, College Park
[2] Northeastern University
[3] Montana State University

For presentation at the Fall Workshop on Computational Geometry (FWCG 2018)

1 Introduction

In recent years, street network analysis has received substantial attention in scholarly and professional urban planning and transportation engineering [7]. Street networks are typically modeled as a graph with intersections as nodes connected to one another by street segments as edges. This graph model enables the computation of various metric, connectivity, and topological measures by efficient algorithms, including accessibility [36, 40], connectivity [21, 25, 28], centrality [15, 30, 39] eccentricity [34], betweenness [3], clustering [27], block lengths [4, 9], and average circuity [6, 8, 20, 23].

One drawback of most existing research in street network analysis is the common assumption of planarity [11] or approximate planarity [15] of the network. In a planar model, the street network is represented in two dimensions such that grade-separated edge crossings such as bridges and tunnels create artificial nodes in the graph. While planar simplifications can be useful for computational tractability, they misrepresent many real-world urban street networks. The failure of this assumption can lead to substantial errors in analytical results [10].

For the present study, we focus in particular on *map comparison*, which can serve the field of urban morphology [5, 26, 38] and the evaluation of map reconstruction algorithms [2]. A recent line of work showed the advantage of topology-based measures for map comparison [1]. Using the common representation of streets by their centerlines [11, 14, 18, 25, 29, 30, 33], streets are split into a sequence of line segments or *street segments* in 2D. Leveraging insights from the nascent field of topological data analysis, the shape of the network can be described by tracing how the connectivity of street segments evolves as segments are gradually thickened [16]. These methods of topological map comparison can be applied in transportation engineering and urban planning research/practice to quantify the difference between two graphs—accounting for non-planarity—to measure street network evolution over time, to compare the structural and functional differences among proposed urban design alternatives, or to match trip trajectories to pre-existing infrastructure models.

In this abstract, we extend and enhance the topology-based measure presented in [1] to accommodate grade-separated nonplanar street networks. Our work is similar in spirit to the study of *multi-layered environments* [35] in motion planning. With the aim of extending known algorithms and data structures for 2D spaces to accommodate surfaces embedded in 3D, the surface is partitioned into *layers*. Each layer is required to project onto the ground plane without self-intersection. The connections between layers can be thought of as staircases and ramps. Ignoring heights, paths across layers are measured by the projected distance in $\mathbb{R}^2$ [32]. Similar ideas have been explored in architecture and urban planning to model circulation paths through multi-story buildings [31, 37].

2 Preliminaries

Topological data analysis is fueled by the concept of persistent homology, which exposes the shape of data by tracing the evolution of topological features [17]. By varying the scale, a sequence of

nested complexes derived from the data yields a *filtration*. Computing the persistent homology on the filtration reveals all the topological features in the data, where each feature is described by its dimension, and the scales at which it appears and disappears [41]. This is often encoded as a *barcode*, which consists of an interval representing the lifespan of each feature from its birth time to its death time [19]. In that way, persistent homology enables us to compare two data sets by comparing their respective barcodes. One way to carry out this comparison is to find the minimum-cost matching between the intervals in each barcode as used to define the *bottleneck distance*. The validity of this distance measure stems from its *stability* against small changes in the data [12]. For further information, the reader is referred to [16, 22].

Our work is an enhancement and extension of the topology-based distance for street networks proposed in [1]. Under the assumption of planarity, a point is fixed in the plane and the goal is to define a *topological signature* of the local neighborhood of this point. To define the neighborhood, a *window* is placed with the point at its center and the street networks are restricted within this window by clipping any segments that extend beyond. A filtration is defined by tracing the topology of the thickened clipped graph, relative to the boundary of the clipping region. The persistence diagram summarizing this filtration is the *local persistent homology* (LPH) diagram. Given two street networks, the signatures can be compared by computing their bottleneck distance. This distance can be integrated by varying the scale and center point used to define the window to obtain a pseudo-metric; see [1] for the details.

3 Defining and Computing Layered LPH Distance

In this abstract, we propose a new distance between street networks which relaxes the planarity assumption. We also describe the computation of this distance by explaining how to compute the Layered Čech Filtration. If all street segments lie in the same layer, the desired distance can be found by examining the segment Voronoi diagram [24], and using the proposed distance for planar LPH distance in [1]. In what follows, we define the (local) Layered Čech Filtration by 'thickening' the graph G while allowing travel through portals. Similar to the one-layer case, our zero-simplices of the Layered Čech Filtration correspond to straight-line embedded edges and appear at radius zero, our one-simplices appear at the radius when two thickened edges overlap. We note that the edges may thicken *through* a portal, which is where the interesting part of the algorithm lies. Finally, the two-simplices correspond to a three-way intersection of thickened edges (again, possibly thickening through portals). Since our domain is a street network, the most interesting topology lies in the connected components and looping behavior, so we do not compute higher-dimensional simplices.

Assumptions and Notation Let $G = (V, E)$ be our piecewise-linear graph immersed in $\mathbb{R}^2$. In what follows, we assume that an oracle $\ell\colon E \to \mathbb{N}$ assigns an edge $e \in E$ to layer $\ell(e)$. We may further assume that $\ell(e) \leq m$ for all edges $e \in E$, for some $m \in \mathbb{N}$. We denote a layer $L_i := \ell^{-1}(i)$. Whenever $\ell(e) = i$, with $e = (u, v)$, we say say that the vertices u, v are also in L_i. We define a *portal* as a vertex shared between at least two layers, and assume any layer has at most k portals, for some $k \in \mathbb{N}$. Notice that while edges map to a single layer, a portal may map to multiple layers (which happens when adjacent edges map to different layers).

3.1 The Layered Čech Filtration

Since we assume that the graph has at most m layers, we think of this graph as m graphs, each of which resides in its own copy of $\mathbb{R}^2$, with portals defining identifications between the layers. We denote this embedding space (m copies of $\mathbb{R}^2$

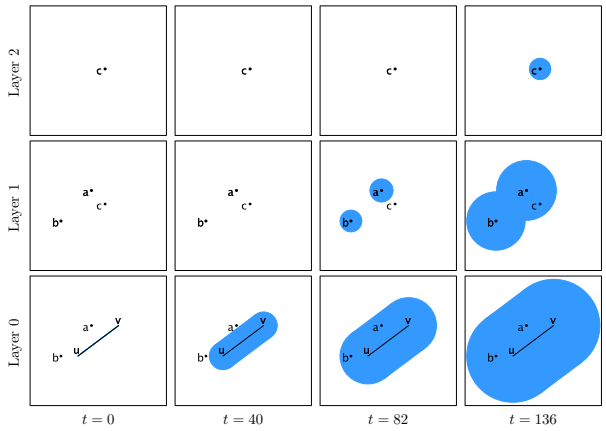


Figure 1: Offset of a segment $\overline{uv}$ across three layers. Notice that the thickening in the third layer starts when the portal c is reached in the second layer.

with an equivalence relation induced from portals) by $\mathbb{L}$. For $\epsilon > 0$, we define the thickened graph G^ϵ to be the tubular neighborhood around G of radius ϵ. We can also think of this as the union of thickened edges. A thickened edge looks like a tubular neighborhood in one layer and a union of disks centered at portals in other layers. The disks need not have the same radius.

One way to approximate the union of thickened sets is through computing the *nerve*. In our setting, we call this nerve filtration the *Layered Čech Filtration*. Specifically, the zero-simplices correspond to the set of edges in the input graph G, the one-simplices correspond to pairwise intersections between edges (and appears at half the distance needed for that intersection to be nontrivial), and the two-simplices correspond to the three-way intersections between thickened edges. When intersections between sets are collapsible, the *nerve lemma* states that the nerve of the set is homotopic to the union of sets. In our setting, however, the portals allow for nontrivial intersections between thickened edges (in fact, a thickened edge might not be contractible). Nonetheless, we use the nerve, as the local connectivity information is important to capture.

3.2 Computation

To compute the Layered Čech Filtration, we must be able to determine values for the thickening radius such that pairs and triples of thickened edges begin to intersect. Since the thickening radius between two edges is half the distance between the edges, first, we describe how to compute distances between edges in $\mathbb{L}$. Then, we consider three-way intersections of thickened edges in $\mathbb{L}$.

Given two edges $e_1, e_2 \in E$, as the triangle inequality still holds between layers, if e_1 and e_2 are on the same layer L, there exists a shortest path between e_1 and e_2 that remains within layer L. If, however, the edges are in different layers, the shortest path passes through multiple layers, perhaps even some layers that do not contain either e_1 or e_2. To compute the length of the path, we introduce an auxiliary data structure called the *Portal Graph*.

The Portal Graph The only way to move between layers is through portals. To help with subsequent distance computations, we precompute a complete weighted graph $\mathcal{P}$ on the $O(km)$ portals. We create a graph $\widetilde{\mathcal{P}}$ with the vertex set corresponding to the portals, and the initial weight of the edge between two portals p and q is:

$$w_0(p,q) = \begin{cases} 0, & \text{if } p = q, \\ d_2(p,q), & \text{if } \exists i \ s.t. \ p, q \in L_i, \\ \infty, & \text{otherwise.} \end{cases}$$

Using this initial weight assignment, we run an all-pairs shortest paths algorithm on this weighted graph to obtain the shortest-path distance $d_P \colon P \times P \to \mathbb{R}$ between all pairs of portals. The portal graph $\mathcal{P}$ is the graph where the edge-weights are the lengths of the shortest paths in $\widetilde{\mathcal{P}}$. Precomputation costs $O(k^3m^3)$ by standard algorithms [13].

Pairwise Distances (Two-way Intersections) Next, we describe how to compute distances between edges in $\mathbb{L}$. Let edges $e_i \in L_i$

and $e_j \in L_j$. If $i = j$, the distance between e_i and e_j in $\mathbb{L}$ is the Euclidean distance between e_i and e_j. If $i \neq j$, we compute the distance between e_i and e_j in $\mathbb{L}$ by augmenting $\mathcal{P}$ as follows. First, let $\mathcal{P}^*$ be the subgraph of $\mathcal{P}$ consisting of portals in L_i and L_j and the edges between the portals in the two layers. Add a vertex u to $\mathcal{P}^*$ where u represents e_i. For each portal $p \in L_i$, add edge (u, p) to $\mathcal{P}^*$ weighted as the Euclidean distance between e_i and p. Similarly, add a vertex v representing e_j and edges to all portals in L_j. The length of the shortest path from u to v in $\mathcal{P}^*$ is the distance from e_i to e_j in $\mathbb{L}$. Since $\mathcal{P}^*$ has at most $2k + 2$ vertices and at most $k^2 + 2k$ edges, the cost of computing a shorted path between u and v in $\mathcal{P}^*$ is $O(k^2)$. Once we have the distance d between edges e_i and e_j in $\mathbb{L}$, the thickening radius where e_i and e_j intersect is $d/2$.

Two-Simplices (Three-way Intersections) Given a triplet of edges, we are interested in finding the thickening radius where the edges intersect. We begin with a few observations. First, observe that a three-way intersection in $\mathbb{L}$ may occur in any layer. Second, observe that for a thickened edge $e \in L_i$, when e thickens enough to reach a portal $p_i \in L_i$, edge e continues to thicken on every layer in which p_i connects. In particular, let $p_j \in L_j$ be a portal on layer L_j in which p_i connects. As we continue to thicken e, we observe that a disk centered at p_j starts growing.

From the observations, we can enumerate the four configurations of how thickened objects can intersect on a layer. In particular the configurationss are: (C1) three thickened edges, (C2) two thickened edges and a disk, (C3) one thickened edge and two disks, (C4) three disks.

We can further use the observations to identify the thickening radius with brute force. In particular, for each layer L_i, we enumerate all configurations and determine r_i the least thickening radius yielding a non-trivial three-way overlap over all configurations on L_i. Finally, the thickening radius r is $\min(\{r_i, \ldots, r_m\})$.

4 Discussion

In this paper, we extend the results of [1] to the more realistic setting that allows for bridges and tunnels to be represented. In particular, we break the graph into layers. We assume the layering structure is part of the input and focus on the problem of computing a topology-based signature for the purposes of map comparison. Future work includes improving the algorithm by exploring the order-k Voronoi diagram of additively-weighted line segments, extending the definition of the Layered Čech Filtration to the *Local* Layered Čech Filtration, and implementing the algorithm.

Acknowledgements BTF would like to acknowledge the generous support of the National Science Foundation under grant CCF-1618605.

References

[1] M. Ahmed, B. T. Fasy, and C. Wenk. Local persistent homology based distance between maps. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 43–52, 2014.

[2] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. *Quality Measures for Map Comparison*, pages 71–83. 2015.

[3] M. Barthélemy. Betweenness centrality in large complex networks. *The European Physical Journal B*, 38(2):163–168, Mar 2004.

[4] M. Barthelemy. From paths to blocks: New measures for street patterns. *Environment and Planning B: Urban Analytics and City Science*, 44(2):256–271, Mar. 2017.

[5] M. Barthelemy, P. Bordin, H. Berestycki, and M. Gribaudi. Self-organization versus top-down planning in the evolution of a city. *Scientific Reports*, 3, July 2013.

[6] M. Barthlemy. Spatial networks. *Physics Reports*, 499(1):1 – 101, 2011.

[7] M. Batty. Big data, smart cities and city planning. *Dialogues in Human Geography*, 3(3):274–279, 2013.

[8] G. Boeing. The Morphology and Circuity of Walkable and Drivable Street Networks. In L. D'Acci, editor, *Mathematics of Urban Morphology (forthcoming)*. Birkhuser, Cham, Switzerland, 2018.

[9] G. Boeing. A Multi-Scale Analysis of 27,000 Urban Street Networks: Every US City, Town, Urbanized Area, and Zillow Neighborhood. *Environment and Planning B: Urban Analytics and City Science*, online first, 2018.

[10] G. Boeing. Planarity and street network representation in urban form analysis. *Environment and Planning B: Urban Analytics and City Science*, in press, 2018.

[11] A. Cardillo, S. Scellato, V. Latora, and S. Porta. Structural properties of planar graphs of urban street patterns. *Phys. Rev. E*, 73, Jun 2006.

[12] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, Jan 2007.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[14] P. Crucitti, V. Latora, and S. Porta. Centrality in networks of urban streets. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(1):015113, 2006.

[15] P. Crucitti, V. Latora, and S. Porta. Centrality measures in spatial networks of urban streets. *Phys. Rev. E*, 73, Mar 2006.

[16] H. Edelsbrunner and J. Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[17] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 454–463, Nov 2000.

[18] B. Frizzelle, K. Evenson, D. Rodriguez, and B. Laraia. The importance of accurate road data for spatial applications in public health: customizing a road network. *International Journal of Health Geographics*, 8(24), 2009.

[19] R. Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[20] D. J. Giacomin and D. M. Levinson. Road network circuity in metropolitan areas. *Environment and Planning B: Planning and Design*, 42(6):1040–1053, 2015.

[21] A. Hajrasouliha and L. Yin. The impact of street network connectivity on pedestrian volume. *Urban Studies*, 52(13):2483–2497, Oct. 2015.

[22] A. Hatcher. *Algebraic topology*. 2005.

[23] J. Huang and D. M. Levinson. Circuity in urban transit networks. *Journal of Transport Geography*, 48:145–153, Oct. 2015.

[24] M. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *Proc. 1 st Int. Symp. on Voronoi Diagrams in Science and Engineering*, pages 51–62, 2004.

[25] P. L. Knight and W. E. Marshall. The metrics of street network connectivity: their inconsistencies. *Journal of Urbanism: International Research on Placemaking and Urban Sustainability*, 8(3):241–259, July 2015.

[26] A. P. Masucci, K. Stanilov, and M. Batty. Limited urban growth: London's street network dynamics since the 18th century. *PLOS ONE*, 8(8):1–10, 08 2013.

[27] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155 – 163, 2009.

[28] D. O'Sullivan. *Spatial Network Analysis*, pages 1253–1273. 2014.

[29] S. Porta, P. Crucitti, and V. Latora. The network analysis of urban streets: A dual approach. *Physica A: Statistical Mechanics and its Applications*, 369(2):853 – 866, 2006.

[30] S. Porta, P. Crucitti, and V. Latora. The network analysis of urban streets: A primal approach. *Environment and Planning B: Planning and Design*, 33(5):705–725, 2006.

[31] J.-C. Thill, T. H. D. Dao, and Y. Zhou. Traveling in the three-dimensional city: applications in route planning, accessibility assessment, location analysis and beyond. *Journal of Transport Geography*, 19(3):405 – 421, 2011.

[32] W. V. Toll, A. F. C. Iv, M. J. V. Kreveld, and R. Geraerts. The medial axis of a multi-layered environment and its application as a navigation

mesh. *ACM Trans. Spatial Algorithms Syst.*, 4(1):2:1–2:34, June 2018.

[33] A. Turner. From axial to road-centre lines: A new representation for space syntax and a new model of route choice for transport network analysis. *Environment and Planning B: Planning and Design*, 34(3):539–555, 2007.

[34] D. Urban and T. Keitt. Landscape connectivity: A graph-theoretic perspective. *Ecology*, 82(5):1205–1218.

[35] W. van Toll, A. F. Cook, and R. Geraerts. Navigation meshes for realistic multi-layered environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3526–3532, 2011.

[36] P. Waddell, I. Garcia-Dorado, S. M. Maurer, G. Boeing, M. Gardner, E. Porter, and D. Aliaga. Architecture for modular microsimulation of real estate markets and transportation. *arXiv preprint arXiv:1807.01148*, 2018.

[37] E. Whiting, J. Battat, and S. Teller. Topology of urban environments. In *Computer-Aided Architectural Design Futures (CAADFutures) 2007*, pages 114–128, 2007.

[38] C. Zhong, S. M. Arisona, X. Huang, M. Batty, and G. Schmitt. Detecting the dynamics of urban structure through spatial network analysis. *International Journal of Geographical Information Science*, 28(11):2178–2199, 2014.

[39] C. Zhong, M. Schlpfer, S. M. Arisona, M. Batty, C. Ratti, and G. Schmitt. Revealing centrality in the spatial structure of cities from human activity patterns. *Urban Studies*, 54(2):437–455, 2017.

[40] D. Zielstra and H. Hochmair. Comparative Study of Pedestrian Accessibility to Transit Stations Using Free and Proprietary Network Data. *Transportation Research Record: Journal of the Transportation Research Board*, 2217:145–152, 2011.

[41] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, Feb 2005.

Red-Blue-Partitioned MST, TSP, and Matching

Matthew P. Johnson*

Abstract

Arkin et al. [2] recently introduced *partitioned pairs* network optimization problems: given a metric-weighted graph on n pairs of nodes, the task is to color one node from each pair red and the other blue, and then to compute two separate *network structures* or disjoint (node-covering) subgraphs of a specified sort, one on the graph induced by the red nodes and the other on the blue nodes. Three structures have been investigated by [2]—*spanning trees*, *traveling salesperson tours*, and *perfect matchings*—and the three objectives to optimize for when computing such pairs of structures: *min-sum*, *min-max*, and *bottleneck*. We provide improved approximation guarantees and/or strengthened hardness results for these nine NP-hard problem settings.

1 Introduction

We consider the class of *partitioned pairs* network optimization problems recently introduced by Arkin et al. [2]. Given a complete metric-weighed graph G whose vertex set consists of n pairs $\{p_1, q_1\}, ..., \{p_n, q_n\}$ (with n even), the task is to color one node from each pair red and the other blue, and then to compute two *network structures* or disjoint (node-covering) subgraphs of a specified sort, one on the graph induced by the blue nodes and the other on the red nodes. One motivation is robustness: if the pairs represent n different types of resources needed to build the desired network structure, with two available instances p_i, q_i of each type i, then solving the problem means computing two separate independent instances of the desired structure, one of which can be used as a backup if the other fails.

The structures that have been investigated are *spanning trees*, *traveling salesperson*, and *perfect matchings*. A solution consists of a disjoint pair of subgraphs covering all nodes, i.e., two (partial) matchings, two trees, or two cycles, and there are different potential ways of evaluating the cost of the pair. The optimization objectives that have been considered are: 1) minimize the sum of the two structures' costs (*min-sum*), 2) minimize the maximum of the two structures' costs (*min-max*), and 3) minimize the weight of the heaviest edge used in either of the structures (*bottleneck*).

Contributions. We provide a variety of results for these nine problem settings (all of which turn out to be NP-hard; see Table 1), including algorithms with improved approximation guarantees and/or stronger hardness results for each. In particular, we provide tighter analyses of the approximation factors of Arkin et al. [2]'s min-sum/min-max 2-MST algorithm, which is equivalent to Algorithm 1 below. We show that the algorithm provides approximation guarantees of 3 and 4 for 2-MST with objectives min-sum and min-max, respectively. We also show that a simple extension of this algorithm (see Algorithm 2 below) provides a 4-approximation for 2-TSP for both min-sum and min-max. All four approximation factors are tight. See the full version of the paper for omitted proofs.

Related work. The primary antecedent of this work is Arkin et al. [2] (see also references therein), which introduced the class of 2-partitioned network optimization problems. Earlier related problem settings include optimizing a path visiting at most one node from each pair [8], generalized MST [15, 17, 3], generalized TSP [3], constrained forest problems [9], adding conflict constraints to MST [18, 12, 6] and to perfect matching [16, 6], and balanced partition of MSTs [1].

2 2-MST

2.1 Min-sum/min-max 2-MST: algorithm

In this section we give a simple algorithm (see Algorithm 1) that provides an approximation guarantee for 2-MST under both the min-sum and min-max objectives. The key lemma the approximation guarantee relies on proves a property about the result of partitioning a metric-edge-weighted spanning tree into a 2-component spanning forest.

Initially we show that *for any 2-coloring* $V_b \cup V_r = V$ of an arbitrary metric-edge-weighted graph (*even without the constraint of specified pairs being colored differently*), the sum of the costs of MSTs on V_b and V_r will be at most three times the cost of an MST on V, and each of them alone will be at most two times this; both inequalities are tight.

Lemma 1 *Let V be the nodes of a metric-weighted graph. Let T be an MST on V, and let $V_b \cup V_r = V$ be any 2-coloring of V, and let T_b and T_r be MSTs of V_b and V_r, respectively. Then we have:*

(a) $c(T_b) + c(T_r) \leq 3c(T)$, *and*

*Lehman College and The Graduate Center, CUNY

Table 1: Summary of results. $R, B \subseteq E$ denote the red and blue solutions, respectively. UB values indicate the approximation factors we obtain, *all for general metric spaces*; LB values indicate hardness of approximation lower bounds, *all (except min-sum and min-max TSP) for the special case of metric weights* $\{1, 2\}$. Best prior bounds (all due to [2]) are also shown, where $\rho_{\text{St}} \leq 2$ denotes the underlying metric space's Steiner ratio (conjectured to be $\frac{2}{\sqrt{3}} \approx 1.1547$ in Euc. 2D [11]), and ρ_{tsp} denotes TSP's best achievable approximation factor in the underlying metric space (currently $\rho_{\text{tsp}} = 1.5$ in general [4]).

		min-sum $c(R) + c(B)$	min-max $\max\{c(R), c(B)\}$	bottleneck $\max\{w_e : e \in B \cup R\}$
MST	**our UB**:	**3**	**4**	–
	[2]'s UB:	$(3\rho_{\text{St}})$	$(4\rho_{\text{St}})$	(9)
	our LB:	**NP-h**	**NP-h**	**2**
	[2]'s LB:	(-)	(NP-h in metric)	(-)
TSP	**our UB**:	**4**	**4**	–
	[2]'s UB:	$(3\rho_{\text{tsp}})$	$(6\rho_{\text{tsp}})$	(18)
	our LB:	$\mathbf{123/122 \approx 1.00819}$ ***with metric weights*** $\{.5, 1, 1.5, 2\}$		**2**
	[2]'s LB:	(-)	(-)	(-)
matching	**our UB**:	–	–	–
	[2]'s UB:	(2)	(3)	(3)
	our LB:	$\mathbf{\frac{8305}{8304} \approx 1.00012}$	$\mathbf{\frac{8305}{8304} \approx 1.00012}$	**2**
	[2]'s LB:	(NP-h in metric)	(weakly NP-h in 2D Euc.)	(-)

Algorithm 1 Min-sum/min-max 2-MST approx

1: $T \leftarrow$ an MST on the $2n$ nodes
2: $\{T_L, T_R\} \leftarrow$ result of deleting a max-weight edge $e_\times$ from T
3: **for** each node pair $(p_i, q_i) \in V_L \times V_R$ **do** color p_i blue and q_i red
4: **for** each other node pair (p_i, q_i) **do** assign p_i, q_i arbitrary distinct colors
5: **for** $c \in \{b, r\}$: $T_c \leftarrow$ a minimum-weight tree spanning the color-c nodes
6: **return** $\{T_b, T_r\}$

(b) $\max\{c(T_b), c(T_r)\} \leq 2c(T)$.

Proposition 1 *There exist families of graphs showing that bounds (a) and (b) of Lemma 1 are (simultaneously) tight.*

Then we refine the proof of this key lemma to improve the combined cost of the two trees slightly, reducing it by the weight of three heavy edges.

Finally we analyze Algorithm 1, which forms trees T_L, T_R by deleting a max-weight edge $e_\times$ (of weight $w_\times$) from an MST T computed on the $2n$ nodes, and then colors all "lone" nodes appearing without their partners in T_L blue and all lone nodes in T_R red, and assigns arbitrary distinct colors to all other node pairs.

Theorem 2 *Algorithm 1 provides a* 3*-approximation for min-sum 2-MST.*

The proof analyzes three cases, depending on whether one, both, or neither T_L, T_R contains a pair, the first two cases of which imply that OPT must cross between T_L and T_R at least once or twice, respectively. The challenge is that $c(OPT)$ is lower-bounded by $c(T_L)+c(T_R)$ but not by $c(T) = c(T_L) + w_\times + c(T_R)$. We upper-bound ALG by carefully applying the refinement of Lemma 1 to $ALG_b + ALG_r$, and we obtain a lower bound on $c(OPT)$ including $w_\times$ or $2w_\times$, permitting the two bounds to be compared, by subtracting max-weight edges from one or both sides.

This immediately implies that the same algorithm provides 6-approximation for min-max 2-MST, but we perform a tighter analysis.

Theorem 3 *Algorithm 1 provides a* 4*-approximation for min-max 2-MST.*

Extending Proposition 1, we obtain:

Proposition 2 *There exist families of instances showing that the 2-MST min-sum and min-max approximation ratios are both tight.*

2.2 Min-sum/min-max/bottleneck: hardness

We provide a reduction inspired by the reduction of [7] from Three-Dimensional Matching to the problem of partitioning a bipartite graph into two connected components, each containing exactly half the vertices.

In our reduction, however, we reduce the traditional 3-SAT problem.

Given the 3-SAT formula, we construct the following graph (see Fig. 1). For each clause, create a path of length p. For each variable x_i, we create create two nodes, x_i and $\bar{x}_i$. We also create a path of length p_b called b and a path of length p_r called r. From each x_i

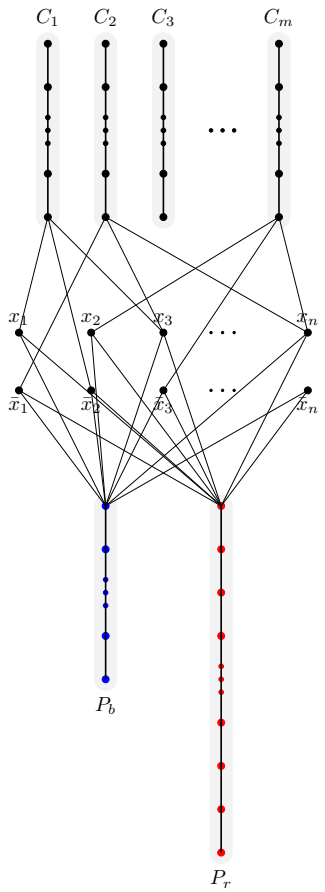


Figure 1: Spanning tree reduction.

or $\bar{x}_i$, we draw an edge to the final nodes of the paths corresponding the clauses that the literal appears in. Finally, from each x_i *and* $\bar{x}_i$, we draw edges to the final nodes paths b and r. All the edges defined have 1; all non-defined edges have weight 2. (In all cases when we refer to the "final" node of one of these $m+2$ paths, we mean the node with degree > 2.)

The path lengths are defined as follows: $p_r = (n+1) \cdot n^3 + n + n + 1, p_b = n^3 + n + 1, p = n^3 + 1$.

Then the total number of nodes in the graph constructed is: $|V| = n \cdot p + p_b + p_r + m, = 2 \cdot (n_r + m)$.

Finally, we must specify the $\{p_i, q_i\}$ pair relationships of these nodes. Each pair $\{x_i, \bar{x}_i\}$ is a $\{p_i, q_i\}$ pair. All p_r nodes of path p_r are p_i s. All p_b nodes of path p and all p nodes of path corresponding to an element are q_i nodes. Observe that results in an equal number of p_i and q_i nodes since $p_b + n \cdot p = p_r$.

Lemma 4 *The formula is satisfiable iff the constructed graph admits a 2-MST solution using only weight-1 edges.*

Thus we conclude:

Theorem 5 *In the special case of metric graphs with weights 1 and 2, min-sum and min-max, 2-MST are both (strongly) NP-Complete, and bottleneck 2-MST is NP-hard to approximate with factor better than 2.*

Algorithm 2 Min-sum/min-max 2-TSP approx

Identical to Alg, 1, except with lines 5,6 replaced by:

5: $C \leftarrow$ a TSP tour, computed from T by edge-doubling
6: **for** $c \in \{b, r\}$: $C_c \leftarrow$ a tour of the color-c nodes, computed by shortcutting C
7: **return** $\{C_b, C_r\}$

3 2-TSP

3.1 Min-sum/min-max/bottleneck 2-TSP: hardness

Clearly the min-sum and min-max objectives for 2-TSP are at least as hard to approximate as ordinary TSP in the same metric space (e.g., hard to approximate with factor better than 123/122 [13], even with edge weights $\{.5, 1, 1.5, 2\}$): to reduce TSP to either of these, simply introduce a co-located pair $\{p_v, q_v\}$ for each node v in the TSP instance. Similarly, the same reduction implies that the bottleneck objective for 2-TSP is at least as hard to approximate as ordinary bottleneck TSP in the same metric space (e.g., hard to approximate with factor better than 2, even with edge weights $\{1, 2\}$).

3.2 Min-sum/min-max 2-TSP: algorithm

Now we adapt Algorithm 1 above to obtain a 4-approximation algorithm for min-sum and min-max 2-TSP (see Algorithm 2).

The proof again analyzes three cases, depending on whether one, both, or neither T_L, T_R contains a pair. Unlike with 2-MST, 2-TSP's $c(OPT)$ *is* lower-bounded by $c(T)$ in the first two cases, and so we can compare it to the simple upper bound on $c(ALG)$ of $4c(T)$.

Theorem 6 *Algorithm 2 is a 4-approximation algorithm for min-sum 2-TSP.*

Theorem 7 *Algorithm 2 is a 4-approximation algorithm for min-max 2-TSP.*

Proposition 3 *There exist families of instances showing that the 2-TSP min-sum and min-max approximation factors are both tight.*

4 2-Matching

4.1 Preliminaries

In the case of perfect matching we require that the number of pairs n be even. It will be convenient to re-express the 2-Matching problem as an equivalent problem concerning cycle covers. This is done as follows. First, for

each pair $\{p_i, q_i\}$, draw a length-2 path (of unit-weight edges) between them, separated by a dummy node d_i, and in the resulting $3n$-node graph G' consider instead the task of finding a 2-factor, i.e., a node-disjoint cycle cover, of minimum cost. In particular, consider seeking a cycle cover that uses only unit-weight edges, which would have cost $3n$.

One lemma we prove shows that any 2-factor (*with all cycle lengths multiples of 6*) in G' will induce a matching in G using only unit-weight edges (proof omitted).

4.2 Bottleneck 2-Matching: hardness

To prove hardness, we give a reduction inspired by Papadimitriou's reduction [5] from 3-SAT to the problem of deciding whether a graph can be partitioned into a node-disjoint collection of cycles, each of size *at least 6*. (Details omitted.)

Theorem 8 *In the special case of metric graphs with weights 1 and 2, bottleneck 2-Matching is NP-hard to approximate with factor better than 2 (and min-sum and min-max 2-Matching are both (strongly) NP-Complete).*

4.3 Min-sum/min-max 2-Matching: hardness

By reduction from a special case of Max 1-in-3 SAT, we can obtain a hardness of approximation result for the min-sum and min-max objectives. Let Max 1-in-3 SAT-5 denote Max 1-in-3 SAT under the restriction that each variable appears in at most 5 clauses.

Lampis has shown (implicitly in [14][1]) the following:

Lemma 9 *There exists a family of* Max 1-in-3 SAT-5 *instances with* $15m$ *clauses and* $8.4m$ *variables, each appearing in at most 5 clauses, for which, for any* $\epsilon > 0$*, it is NP-hard to decide whether the minimum number of unsatisfiable clauses is at most* ϵm *or at least* $(0.5-\epsilon)m$*.*

For concreteness, let Min Not-1-in-3 SAT-5 indicate the optimization problem of minimizing the number of unsatisfied clauses in a 1-in-3 SAT-5 formula.

Now we argue that the same construction used above provides an approximation-preserving reduction from Min Not-1-in-3 SAT-5.

Corollary 1 *Min-sum and min-max 2-Matching are both, in the special case of metric graphs with weights 1 and 2, NP-hard to approximate with factor better than* $8305/8304 \approx 1.00012$.

Acknowledgements. This work was supported in part by NSF award INSPIRE-1547205, and by the Sloan Foundation via a CUNY Junior Faculty Research Award. We thank Ali Assapour, Ou Liu, and Elahe Vahdani for useful discussions.

[1] Karpinksi et al. [13] provide a similar construction yielding a stronger hardness of approximation lower bound for Metric TSP, but adapting that construction to our present problem actually leads to a slightly weaker lower bound.

References

[1] M. Andersson, J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Balanced partition of minimum spanning trees. *International Journal of Computational Geometry & Applications*, 13(04):303–316, 2003.

[2] E. M. Arkin, A. Banik, P. Carmi, G. Citovsky, S. Jia, M. J. Katz, T. Mayer, and J. S. B. Mitchell. Network optimization on partitioned pairs of points. In *ISAAC*, pages 6:1–6:12, 2017.

[3] B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In *COCOA*, pages 110–125. 2015.

[4] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 88, Management Sciences Research Group, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[5] G. Cornuejols and W. Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29(2):135–159, 1980.

[6] A. Darmann, U. Pferschy, J. Schauer, and G. J. Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011.

[7] M. E. Dyer and A. M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

[8] H. N. Gabow, S. N. Maheshwari, and L. J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, (3):227–231, 1976.

[9] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[10] P. Hell and D. G. Kirkpatrick. Packings by cliques and by finite families of graphs. *Discrete Mathematics*, 49(1):45–59, 1984.

[11] A. O. Ivanov and A. A. Tuzhilin. The Steiner ratio Gilbert–Pollak conjecture is still open. *Algorithmica*, 62(1-2):630–632, 2012.

[12] M. M. Kanté, C. Laforest, and B. Momege. Trees in graphs with conflict edges or forbidden transitions. In *TAMC*, pages 343–354. Springer, 2013.

[13] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.

[14] M. Lampis. Improved inapproximability for TSP. In *APPROX/RANDOM*, pages 243–253. Springer, 2012.

[15] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[16] T. Öncan, R. Zhang, and A. P. Punnen. The minimum cost perfect matching problem with conflict pair constraints. *Computers & Operations Research*, 40(4):920–930, 2013.

[17] P. C. Pop. New models of the generalized minimum spanning tree problem. *Journal of Mathematical Modelling and Algorithms*, 3(2):153–166, 2004.

[18] R. Zhang, S. N. Kabadi, and A. P. Punnen. The minimum spanning tree problem with conflict constraints and its variations. *Discrete Optimization*, 8(2):191–205, 2011.

The Strength of Marble-Powered Computing

Matthew P. Johnson*

Abstract

Turing Tumble is a toy mechanical computer (generalizing the classic Digi-Comp II, principally through the addition of *gears*), in which marbles roll down a board, along paths determined by the *locations* of ramps, toggles and gears, which are chosen by the "programmer", and by their current *states*, which the marbles affect when visiting them. Aaronson proved Digi-Comp II was CC-Complete, i.e., equivalent to evaluating *comparator circuits*, and posed the question of what additional functionality might raise its computational power beyond CC, speculating that a capability for toggles to affect one another's states (which Turing Tumble's gears provide) might suffice. This turns out to be correct: we show that Turing Tumble is P-Complete.

1 Introduction

Turing Tumble is a toy mechanical (or "marble-powered") computer, which was recently recently introduced after raising $400k on Kickstarter.[1] It consists of a two-dimensional square grid pegboard, which stands at an angle; a collection of pieces including ramps, toggles (or "bits"), gears, crossovers, and interceptors; and a set of marbles (of two colors). The user "programs" the board through the placement of the pieces. The program is run by pressing a button to release a marble from the top, causing it to roll down the board, passing through some *diagonally contiguous* sequence of pieces (see Fig. 1), perhaps affecting their state, until it reaches a paddle on the bottom-left or -right, where it causes the release of a second marble (from the upper-left or -right, respectively), and so on. The path traversed depends both on the placement of the pieces (the program) and on their current state (the memory).

Toggles can be in one of two states (*left* or *right*, or 0 or 1), which determines which direction the marble rolls from there. Crucially, each marble's visit to a toggle *flips its state*, toggling it back and forth. Moreover, special toggles (which we will simply call *gears*) can be connected to one another by gears, entangling them, so that *flipping any one of their states flips them all*.

The final state of (some subset of) the toggles can be interpreted as the program's output. Alternatively, with two marble colors and two starting points, the color pattern of marbles collected at the bottom of board can also be interpreted as output. Examples from the Turing Tumble website and instruction manual include programs for arithmetic and counting in binary, as well as various marble color patterns.

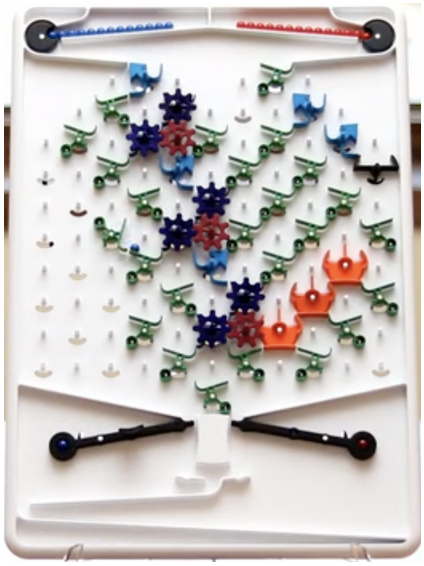


Figure 1: Image from the Turing Tumble website.

Turing Tumble generalizes the Digi-Comp II[2], a toy mechanical computing first sold in the 1960s, whose behavior was more restricted in a number of ways: rather than a general grid, its collection of paths id hardcoded (specifically, carved into wood), with its ramps and toggles in fixed locations; it has no gears, crossovers or interceptors, and it has only one color (and source) of marbles. (Programming it consists entirely in initializing the toggles' states.) Yet it also can perform interesting computations such as arithmetic and binary counting. Aaronson [1] investigated the computational power of a kind of generalized Digi-Comp II in which the structure of paths and toggle locations is specified by an arbitrary DAG with a unique *source* and designated target *sink*, and whose internal nodes represent toggles. He defined DIGICOMP as the problem of deciding whether, for a given instance (i.e., the DAG, the toggles' initial states, and the number T of marbles, encoded in unary) whether any marbles released at the DAG's source will eventually eventually reach its sink.

Aaronson showed that DIGICOMP is not circuit-universal, i.e., is not P-Complete, but is instead merely CC-Complete [2]. That is, deciding DIGICOMP is equivalent not to evaluating Boolean circuits but rather (under log-space reductions) to evaluating *comparator* circuits. One of the open questions Aaronson raised

*Lehman College and The Graduate Center, City University of New York

[1] `https://www.turingtumble.com`

[2] `https://digi-compii.com`

asked what additional potential features to the model might render DIGICOMP P-Complete. In particular, he speculated that the addition of direct causal interaction between the pieces' states ("toggles and switches controlled by other toggles") might suffice. In this abstract we show that the interaction provided by Turing Tumble's gears indeed results in P-Completeness.

2 Model

It is clear that simulating Turing Tumble with all its features is in P. For proving hardness, we make a number of simplifying restrictions. We assume that there is only one marble color, only one *source*, and two out-degree-0 nodes, one triggering the release of the next marble and the other being the designated target *sink*. We can assume all nodes have in-degree and out-degree at most 2, or indeed that toggles, ramps, and (bit-storing) gears can only be placed at *odd* grid points (see Fig. 1), while non-bit-storing gears can be placed at even grid points. (It is illegal for a marble to free-fall.) Although it will be convenient to draw the construction using edge crossings, by an argument of Alexander Meiburg in a comment on Aaronson's blogpost, we may assume w.l.o.g. that there are no edge crossings, without using crossover pieces. We also do not use interceptors.

To have their states entangled, a collection of gears must be *(rectilinearly) contiguous* (see Fig. 1). Call a maximal contiguous set of gears a *gear component*. No two pieces can intersect; thus we can assume gear components do not have holes.

Lemma 1 *Small gear components C can be constructed such that: when a marble visits it via one entry point C's state is flipped as usual, but visits via another entry point leave C's state unchanged. Moreover, C can be designed to have four different exit points, encoding C's state* and *whether it was flipped by the exiting marble.*

Therefore we abstract away from gears to a simpler, weaker model in which gear components of this kind are subsumed by a more powerful kind of toggle node which can be visited in two different ways, reading its state and either flipping it or not. Therefore we label an edge (i, j) (when degrees are greater than one) as being from F_i^0, F_i^1, R_i^0, or R_i^1 (encoding i's bit value read and whether it was *flipped* or merely *read*) to F_j or R_j (encoding whether j's bit value *will be* flipped or merely read). (The actual bit-storing gears underlying the component will have out-degree at most 2.) Out-degree-1 nodes are stateless.

The DAG described will include nodes with greater than two outgoing edges, which are *ordered*, meaning that each successive marble leaving the node crosses the next edge in this list, but this is only for convenience; such a node can easily be expanded to a directed binary tree whose (consistent) sequence of paths traversed by marbles visiting its root is fully determined by its internal nodes' initial states.

TURINGTUMBLE is the problem of deciding whether, when a given n-piece instance is run on $3n$ marbles, any of them reach the sink. Note that the only way in which the formalization *generalizes* the actual product is in parameterizing the number of pieces (and thus the board size). Note also that unlike in the case of DIGICOMOP's CC-Completeness proof, we need not specify the number of marbles as part of the problem instance since the number of marbles the decision problem concerns is linear in n.

3 Reduction

We reduce from Sequential NOR-CVP [3], a P-Complete specialized version of the Circuit Value Problem (CVP). In this version (which does not involve input variables), a circuit is specified by a sequence of n gates having values $G_1 = 1$, $G_2 = 0$, and for all $i > 2$, $G_i = !(G_{i-1} \vee G_{i'})$ where $p(i) < i$. The task is to evaluate the state of G_n. By performing [3]'s reduction on NANDCVP with fanout 2, rather than on general CVP with ANDs and NOTs, it can be seen that we may assume that NOR gates in Sequential NOR-CVP have fanout at most 3. Define out-deg$(G_i) = |p^{-1}(i)|$.

Theorem 2 *TURINGTUMBLE is P-Complete under log-space reductions.*

Proof. (Sketch.) Given the Sequential NOR-CVP formula, we construct a Turing Tumble instance as follows. We create a node for each gate G_i, for $i \in [3, n]$, initializing it to 0 if $p(i) = 1$, and otherwise to 1. We create out-deg(G_i) edges from the source to R_i, for $i \in [3, n-1]$, and one to R_n; these edges are *ordered* by index i. For each pair $(G_{p(i)}, G_i)$ with $p(i) > 0$, we create an edge from $R^1_{p(i)}$ to F_i. Finally, we create an edge from R^1_n to the sink. (Any missing edges at exit points can be assumed to go to the non-sink terminal.) It is clear that the construction can be performed in log space.

When run, for each $i \in [3, n-1]$, out-deg(G_i) marbles will travel from the source to G_i, and, if $G_i = 1$, thence to G_i's children, flipping their states to 0. (Notice that this happens to each child at most once.) Finally, a marble travels from the source to G_n, and thence, if $G_n = 1$, to the sink. □

References

[1] S. Aaronson. The power of the Digi-Comp II: My first conscious paperlet, *Shtetl-Optimized*, 2010. https://www.scottaaronson.com/blog/?p=1902.

[2] S. A. Cook, Y. Filmus, and D. T. M. Lê. The complexity of the comparator circuit value problem. *ACM ToCT*, 6(4):15, 2014.

[3] A. Okhotin. A simple P-complete problem and its language-theoretic representations. *TCS*, 412(1-2):68–82, 2011.