Young Reseachers Forum: Book of Abstracts

Computational Geometry Week 2015, Eindhoven, The Netherlands

2048 is NP-Complete

Ahmed Abdelkader [*] Aditya Acharya [†] Philip Dasler [‡]

Abstract

2048 is a single-player online puzzle game that went viral in March 2014. The game is played on a 4×4 board by sliding around and merging equal valued tiles to create tiles of higher value. The player wins by creating the *2048* valued tile, hence the name. We study the complexity of a slightly adapted version and prove that a number of natural decision problems turn out to be NP-Complete. We reduce from `3SAT` and implement our reduction as an online game.

1 Introduction

Each turn, the player picks a move from $\{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ to slide all tiles on the board. Tiles slide as far as possible in the chosen direction until they hit either another tile or an edge of the board. When a sliding tile runs into a stationary one of equal value, they merge into a tile of double that value. Trailing tiles following a tile that just merged continue to slide uninterrupted and may merge among themselves as they come to rest one after the other. However, newly merged tiles cannot merge further in the same move. After each move, a *2* or *4* tile is generated in one of the empty cells. The player wins when a *2048* tile is created, hence the name of the game. Otherwise, the player loses when the board is full and no merges can be performed.

2048 combines features from two families of games: Candy Crush Saga [1] and PushPush [2]. A more detailed draft of this work appears in [3], where we also discuss the *attempt* in [4]. The interactive gadgets and *playable reduction* can be accessed at [5].

1.1 Adaptations and Problem Definition

We adapt the original *2048* as follows: (1) The input encodes the complete board configuration and no new tiles are generated. (2) The board is a rectangular grid of arbitrary size. In this paper, we are primarily concerned with the following decision problem:

Definition 1 (`2048-GAME`) *Given a configuration of tiles on an $m \times n$ board, is it possible to obtain a tile of value* 2048*? (More generally, 2^k with $k \geq 8$.)*

[*]University of Maryland, College Park, `akader@cs.umd.edu`
[†]University of Maryland, College Park, `acharya@cs.umd.edu`
[‡]University of Maryland, College Park, `daslerpc@cs.umd.edu`

[6] presented a proof of membership in NP, that applies to `2048-GAME`. The crucial piece is to bound the number of moves between two consecutive merges. Using a canonical orientation, all moves are interpreted as flips of the board, which send tiles along orbits of $O(mn)$ length. A pair of tiles that end up merging requires no more than $LCM(O(mn), O(mn)) = O(m^2n^2)$ moves.

In this paper, we prove NP-hardness by a reduction from 3SAT and obtain the main result.

Theorem 1 `2048-GAME` *is NP-Complete.*

2 Reduction from 3SAT

Given an instance of `3SAT` with n variables and m clauses, we produce an instance of `2048-GAME`. The board is filled using a 2-4 lattice to provide a rigid base for placing gadgets and planning their movements. We allow no merges using lattice tiles, which requires preserving their parity. This confines all merges to multiples of 2×2 blocks. We use *row* and *column* to denote a 2-row and a 2-column, respectively.

Displacers: These are the building blocks of all gadgets which allow us to communicate signals across the board. They come in two main forms: horizontal D and vertical D^T. Typically, a displacer starts in an *inactive* state where the middle 2×2 block, highlighted below, is shifted perpendicularly to the displacer's direction. An inactive displacer cannot merge, by any move sequence, before it is activated. The only way to activate it is to use another properly aligned displacer to engage its middle block. Collapsing tiles in a displacer shrinks it to a 2×2 block, which results in a *parity-preserving pull* in a row or a column.

$$D = \left[\begin{array}{c|cc|c} 8 & 8 & 16 & 16 \\ 32 & 32 & 64 & 64 \end{array} \right]$$

Variable Gadget: Each variable is represented by two horizontal displacers on the same *row.* This enables variables to move the portion of its row between their two displacers to the right or left. We enforce the assignment of variables in the order of their indices. A variable is assigned T or F using a $\rightarrow$ or $\leftarrow$ move, respectively. The displacers of x_1 come activated in the initial configuration to allow the game to start. No matter how variable x_i is assigned, the *connector displacers* in its row get activated and allow x_{i+1} on top of it to be activated by a $\downarrow$ move in the following turn.

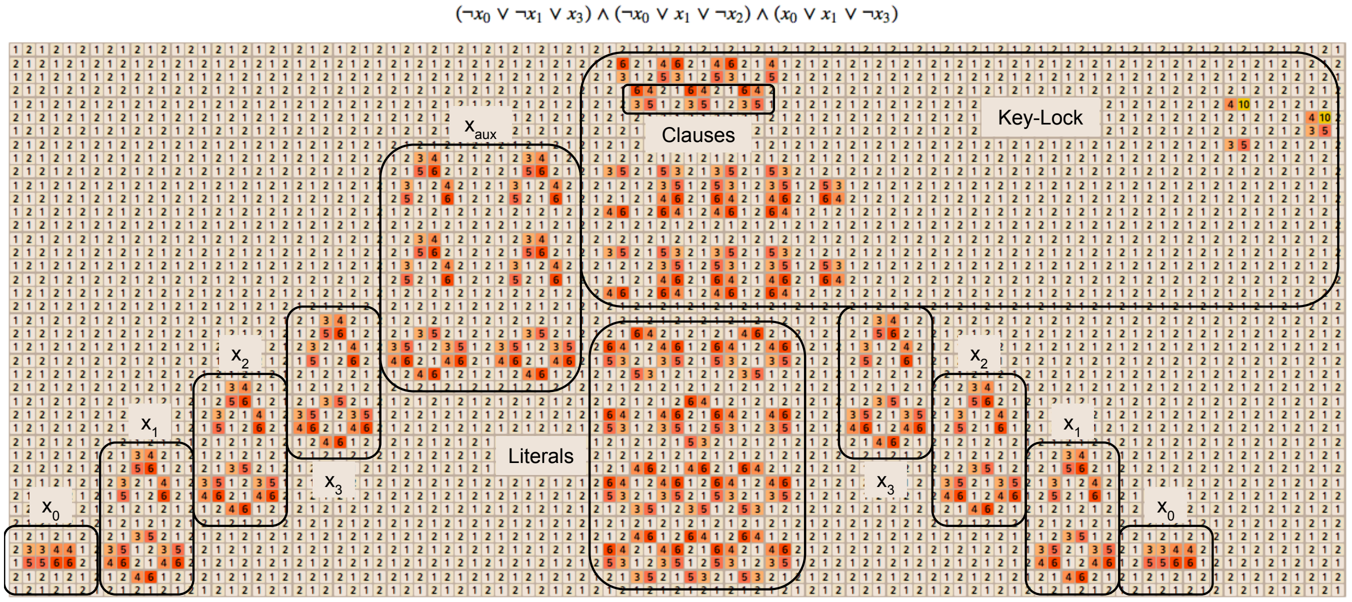

$(\neg x_0 \vee \neg x_1 \vee x_3) \wedge (\neg x_0 \vee x_1 \vee \neg x_2) \wedge (x_0 \vee x_1 \vee \neg x_3)$
x_aux
Clauses
Key-Lock
Literals
x_0
x_1
x_2
x_3

Figure 1: Annotated reduction. Only x_0 is active. We apply $\log_2$ and hide paddings to help display a large board.

Clause Gadget: Literals are encoded using a similar mechanism to the connectors in the variable gadget. but are only activated by the appropriate assignment. When a literal is activated, it allows a $\downarrow$ pull in the clause's *column* to effectively satisfy this clause. Each satisfied clause eventually contributes one horizontal displacer by providing its middle block.

Key-Lock Gadget: To check that all clauses are satisfied, it helps to arrange for a special event to happen only after all variables have been assigned. To achieve this, an auxiliary variable $x_{aux} = x_{n+1}$ activates the lock portion of this gadget. Satisfying all clauses corresponds to using the correct key. Together, the activated key-lock gadget is a sequence of displacers that can activate a unique displacer with two 1024 tiles. Collapsing that unique displacer creates the desired *2048* tile.

2.1 Properties of the Reduction

Size: As variables are stacked on top of each other all the way up to x_{aux} and the key-lock gadget, the number of rows is $O(n)$. Then, each variable has to activate the connectors to the next variable. We get a pyramid shape with variable displacers on both sides and literals in the middle, plus the unique displacer taking up $2(m+1)$ columns far to the right, for a total of $O(m+n)$ columns.

Gaps and Padding: A gap is created iff two tiles merge. The construction guarantees that gaps are only created near the edges of the board and accumulate at the corners. To make sure such gaps do not result in undesired shifts within the core, it has to be surrounded by enough padding. As the number of active gadgets is $O(m+n)$ and each gadget contributes a constant number of gaps, a padding of $O(m+n)$ thickness suffices.

Game Play: When no merges happen, two consecutive moves in opposite directions leave the board unchanged e.g. $[\leftarrow, \rightarrow, \leftarrow]$ is effectively reduced to $[\leftarrow]$. *Effective* moves alternate between horizontal and vertical. The alternation accumulates newly created gaps, resulting from the merge, at the corners so the decision encoded by the previous move cannot be altered. Furthermore, any row or column may witness merges during at most one turn. In particular, clause columns cannot experience more than one $\downarrow$ pull. This implies consistent assignments. Finally, $\uparrow$ moves are useless since they must be canceled or otherwise the player cannot win.

Hardness: Aligning the two 1024 tiles requires a $2m$ shift, which only satisfied clauses can provide with each satisfied clause contributing 2. Hence, the *2048* tile can be created iff the 3SAT instance is satisfiable.

References

[1] Luciano Gualà, Stefano Leucci, and Emanuele Natale. Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard. *CoRR*, abs/1403.5830, 2014.

[2] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush is NP-hard in 2D. *CoRR*, cs.CG/0001019, 2000.

[3] Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. On the complexity of slide-and-merge games. *CoRR*, abs/1501.03837, 2015.

[4] Rahul Mehta. 2048 is (PSPACE) Hard, but Sometimes Easy. *CoRR*, abs/1408.6315, 2014.

[5] Ahmed Abdelkader. 2048 gadgets. `http://cs.umd.edu/~akader/projects/2048/index.html`.

[6] Christopher Chen. 2048 is in NP. `http://blog.openendings.net/2014/03/2048-is-in-np.html`.

Covering Exactly One of Each Pair [*]

Esther M. Arkin [†] Aritra Banik [‡] Paz Carmi [‡] Gui Citovsky [†] Matthew J. Katz [‡]
Joseph S. B. Mitchell [†] Marina Simakov [‡]

Abstract

Let $P = \{P_1, P_2, \ldots, P_n\}$ be a set of pairs of points. We explore various covering problems in both one and two dimensions where exactly one point from each pair must be covered by an interval or an axis-aligned square.

1 Introduction

Let $P = \{P_1, P_2, \ldots, P_n\}$ be a set of pairs of points. We show that the following four problems are NP-hard; the decision versions are easily seen to be in NP.

Problem 1. Let P be on a line. Find a minimum-cardinality set $\mathcal{I}$ of unit length intervals (assuming a feasible solution exists), such that exactly one point from each pair is covered by an interval in $\mathcal{I}$.

Problem 2. Let P be on a line. Decide whether or not there exists a set of unit length intervals, $\mathcal{I}$, such that exactly one point from each pair is covered.

Problem 3. Let P be on a line. Find a minimum-cardinality set $\mathcal{I}$ of intervals of arbitrary length, such that exactly one point from each pair is covered by an interval in $\mathcal{I}$ (a feasible solution always exists).

Problem 4. Let P be in the Euclidean plane. Find a minimum-cardinality set $\mathcal{S}$ of axis-aligned unit squares (assuming a feasible solution exists), such that exactly one point from each pair is covered by a square in $\mathcal{S}$.

We represent point pairs in Figures 1 - 3 as the tips of a $\sqcap$ shape or the tips of a $\sqcup$ shape. Certain pairs in these figures are drawn in color in order to help explain the constructions.

2 Unit intervals

Theorem 1 *Problem 1 is NP-hard.*

Proof. The reduction is from boolean 3-satisfiability (3-SAT). Given n variables $\{x_1, x_2, \ldots, x_n\}$, and m clauses $\{c_1, c_2, \ldots, c_m\}$, we design the following gadgets.

A clause gadget, c_i, contains 13 points. It contains four consecutive pairs of points, d_{ij}, $1 \leq j \leq 4$ (represented by a $\sqcap$ shape in Figure 1) and another pair of points, d_i, one of which lies between d_{i1} and d_{i2} and the other lies between d_{i3} and d_{i4}. The remaining three points (blue in Figure 1) lie between d_{i1} and d_i, between d_{i2} and d_{i3} and between d_{i3} and d_i. Each of these three blue points is paired to a blue point in a variable gadget (blue pairs are represented by a $\sqcup$ shape in Figure 1). The Euclidean distance between the right point in d_{ij} and the left point in d_{ij+1}, $1 \leq j \leq 3$, is less than one, ensuring that one unit interval can cover both d_{ij} and d_{ij+1}. The points of d_{ij} are spaced unit distance apart.

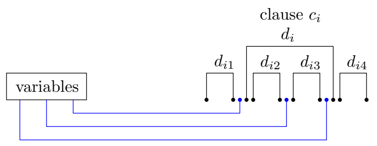


Figure 1: Clause gadget.

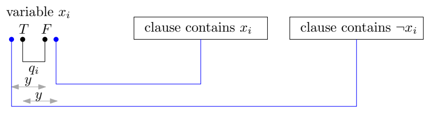


Figure 2: Variable gadget.

Each variable gadget (Figure 2) consists of a consecutive pair of points, q_i, surrounded by blue points on each side. If variable x_i (resp. $\neg x_i$) appears in clause c_i, then one blue point will be placed to the right (resp. left) of q_i and this point will be paired to a blue point in c_i. The blue points that surround q_i are placed a distance of y from their respective farthest points in q_i. We set $y < 1$, ensuring that any unit interval that covers a point in q_i must also cover either the surrounding blue points to the left or right of q_i. Setting x_i to FALSE is equivalent to covering the right point of q_i. Setting x_i to TRUE is equivalent to covering the left point of q_i.

We line up all of the variables, followed by all of the clauses, so that each consecutive gadget is spaced farther than unit distance apart. Note that if a clause evaluates to FALSE, no blue point in a clause can be covered. Therefore, four unit intervals are required to cover this clause. If a clause evaluates to TRUE, then three intervals (and no less) can cover the clause. Pair d_i is vital to this being true.

A satisfying truth assignment in 3-SAT exists if and only if a minimum cover uses $n + 3m$ unit intervals. □

Theorem 2 *Problem 2 is NP-complete.* [Proof is omit-

*This work was supported by the US-Israel Binational Science Foundation (project 2010074). A superset of these results has been submitted (2015) for publication.

†Dept. of Applied Mathematics and Statistics, Stony Brook University. {estie, gcitovsk, jsbm}@ams.stonybrook.edu

‡Dept. of Computer Science, Ben-Gurion University, Israel. {aritrabanik,carmip}@gmail.com, {matya, simakov}@cs.bgu.ac.il

ted in this abstract.]

3 Arbitrary length intervals

Theorem 3 *Problem 3 is NP-hard.*

Proof. The reduction is again from 3-SAT. In this case, spacing of points is irrelevant. Variable gadgets are set up very similarly to the unit interval version. This time, in order to ensure that in a minimum-cardinality cover the blue points (either to the left or to the right of q_i) in a variable gadget are covered with the same interval that covers q_i, we enclose pair q_i with a "safety" pair s_i (see Figure 3). We will see that covering a point in q_i and not using the same interval to cover a point in s_i would be too costly. Clause gadgets are set up as in the unit interval, optimization problem construction (Figure 1).

We break the set of points in the construction into two halves, H_1, which contains the variable and clause gadgets, and H_2, which contains another gadget described below (see Figure 3). Surrounding each variable and each clause we place a cluster of $M >> n + 3m$ points. Note that the points in a cluster are laid out side-by-side (rather than on the same x-coordinate). In H_2, we create M groups of points, where each group is made up of $n + m + 1$ points, one paired to each cluster in H_1. Surrounding these groups are pairs of consecutive points, g_1 and g_2. Pair g_1 lies to the left of the first group and pair g_2 lies to the right of the last group. The gadget in H_2 will help us isolate all of the variable and clause gadgets in H_1.

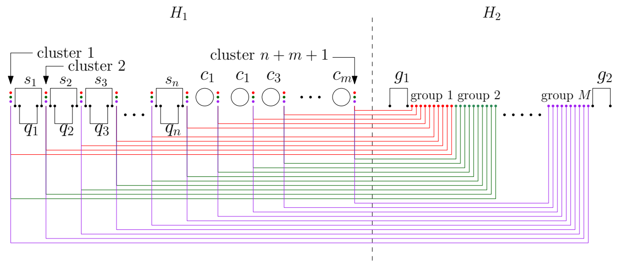


Figure 3: Arbitrary length intervals - the big picture.

First, we show that any feasible solution uses at least $n + 3m + 1$ intervals.

Case 1: No cluster in H_1 is completely covered. The variable and clause gadgets are now isolated. We need at least n intervals to cover the variable gadgets and at least $3m$ intervals to cover the clause gadgets. At least one more interval is needed to cover the remaining points in H_2.

Case 2: At least one cluster in H_1 is completely covered. If any cluster is completely uncovered then at least M intervals will be needed in H_2. If all clusters are "touched" by an interval then at least $n + 3m + 1$ intervals will be used in H_1 (at least $n+m+1$ intervals touch a cluster and at least $2m$ intervals are needed to finish covering the clauses). At least one more interval is needed to cover points in H_2. At least $n + 3m + 2$ intervals are used in total.

Now we claim that there exists a satisfying truth assignment in 3-SAT if and only if a minimum cover uses $n + 3m + 1$ intervals.

Suppose there exists a satisfying truth assignment. Any feasible solution must use at least $n + 3m + 1$ intervals. We achieve this bound by covering pairs in H_1 the same way as in the unit interval optimization version and using one more interval in H_2 to cover g_1, all groups, and g_2.

Now suppose that a minimum cover uses $n + 3m + 1$ intervals. By Case 2, we know that no cluster in H_1 can be completely covered. Thus, variable and clause gadgets are isolated the same way they were in the unit interval version. Recall that in the variable gadgets, a "safety" pair s_i encloses the set of blue points that extend to clause gadgets. If the interval used to cover q_i does not also cover pair s_i, then an extra interval will be needed in the covering; this would be one interval too many. Therefore, we now see that variable gadgets work the same way as in the unit interval version. This means that if any clause would have evaluated to FALSE then at least $n+3m+2$ intervals would have been needed. □

4 Two dimensions

Theorem 4 *Problem 4 is NP-hard, even for point pairs that are horizontal/vertical at unit separation.*

The proof (omitted here) relies on a reduction from PLANAR 3-SAT [2] and a technique used by Fowler et al. [1]. In contrast with the one-dimensional constructions, our proof uses unit separated pairs of points, aligned horizontally or vertically. In one dimension, with unit separated pairs and unit intervals, the optimization problem is polynomial-time solvable with dynamic programming.

5 Conclusion

Our results show the hardness of solving exact one-of-a-set coverage problems. In ongoing work, we are exploring approximation algorithms for the optimization problems mentioned here; a summary of these results will appear in the talk and the full paper. It is worth noting that greedy strategies can perform very poorly.

References

[1] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete . *Information Processing Letters*, 12(3):133 – 137, 1981.

[2] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

Automatic Proofs for Formulae Enumerating Proper Polycubes

Gill Barequet* Mira Shalah*

Abstract

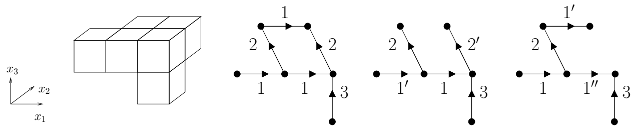
We develop a general framework for computing formulae enumerating polycubes of size n which are proper in $n-k$ dimensions (spanning all $n-k$ dimensions), for a fixed value of k. We reaffirm the already-proven formulae for $k \le 3$, and give the first rigorous proof for $k=4$.

1 Introduction

A d-dimensional polycube of size n is a connected set of n cubes in d dimensions, where connectivity is through $(d-1)$-dimensional faces. Two *fixed* polycubes are considered distinct if they have different shapes or orientations. A polycube is *proper* in d dimensions if it spans all the d dimensions. Following Lunnon [4], we let $\mathrm{DX}(n,d)$ denote the number of fixed polycubes of size n that are proper in d dimensions.

Enumeration of polycubes is a fundamental problem in combinatorics and discrete geometry, originating in statistical physics [3]. No formula is known for $A_d(n)$, the number of fixed polycubes of size n in d dimensions. The main interest in DX stems from the formula $A_d(n) = \sum_{i=0}^{d} \binom{d}{i} \mathrm{DX}(n,i)$ [4]. In a matrix listing the values of DX, the upper triangle and the main diagonal contain only 0s, giving rise to the question whether a pattern can be found in the diagonals $\mathrm{DX}(n,n-k)$.

In statistical physics, Peard and Gaunt [7] predicted that for $k>1$, the diagonal formula $\mathrm{DX}(n,n-k)$ has the pattern $2^{n-2k+1}n^{n-2k-1}(n-k)h_k(n)$, where $h_k(n)$ is a polynomial in n, and explicit formulae for $h_k(n)$ for $k \le 6$ were conjectured. Luther and Mertens conjectured a formula for $k=7$. Using Cayley trees, it can be shown that $\mathrm{DX}(n,n-1) = 2^{n-1}n^{n-3}$ (seq. A127670 in [6]). Barequet et al. [2] gave the first rigorous proof that $\mathrm{DX}(n,n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2-6n+9)$ (seq. A171860). The proof uses a case analysis of the possible structures of spanning trees of the polycubes, and the various ways in which cycles can be formed in their cell-adjacency graphs. Similarly, Asinowski et al. [1] proved that $\mathrm{DX}(n,n-3) = 2^{n-6}n^{n-7}(n-3)(12n^5 - 104n^4 + 360n^3 - 679n^2 + 1122n - 1560)/3$, again, by counting spanning trees of polycubes, yet the reasoning and calculations were significantly more involved. The inclusion-exclusion principle was applied in the proof in order to correctly count polycubes whose cell-adjacency graphs contained so-called "distinguished structures."



(a) Polycube P (b) $\gamma(P)$ (c) Spanning trees

Figure 1: A polycube P, $\gamma(P)$, spanning trees of $\gamma(P)$.

In comparison with $k=2$, the number of such structures is substantially higher. This proof provided a better understanding of the difficulties to be faced in applying this technique to higher values of k. The number of distinguished structures grows rapidly, and the inclusion relations between them are much more complicated. As anticipated [1], it is impractical to achieve a similar proof manually for $k>3$.

In this work we create a theoretical set-up for proving the formula for $\mathrm{DX}(n,n-k)$, for a fixed k. Our method fully automates the manual method presented in [2, 1]. In this generalization, we give a few key observations about polycubes proper in $n-k$ dimensions. We also give a general characterization of distinguished structures, and design algorithms that produce them automatically. Using our implementation of this method, we find the explicit formula (which has never been proven before) for $\mathrm{DX}(n,n-4)$, stated in the following theorem, confirming the formula conjectured in [7].

Theorem 1 $\mathrm{DX}(n,n-4) = 2^{n-7}n^{n-9}(n-4)(8n^8 - 128n^7 + 828n^6 - 2930n^5 + 7404n^4 - 17523n^3 + 41527n^2 - 114302n + 204960)/6$.

2 Overview of the Method

Let $\mathcal{P}_n$ be the set of proper polycubes of size n in $n-k$ dimensions. Let $P \in \mathcal{P}_n$, and let $\gamma(P)$ denote the adjacency graph of P. The vertices of $\gamma(P)$ correspond to the cells of P; two vertices are connected by an edge if their corresponding cells are adjacent; an edge has label i ($1 \le i \le n-k$) if the corresponding cells have different i-coordinate. The direction of the edge is from the lower to the higher cell. See Figure 1. $P \mapsto \gamma(P)$ is an injection. Therefore, we count the graphs obtained from the members of $\mathcal{P}_n$ in this way. We count these graphs by counting their spanning trees. Such a spanning tree has $n-1$ edges labeled by numbers from the set $\{1,2,\ldots,n-k\}$; all these labels are present because the polycube is proper in $n-k$ dimensions. Hence, $n-k$

*Dept. of Computer Science, The Technion, Haifa, Israel. E-mail: {barequet,mshalah}@cs.technion.ac.il

edges of the spanning tree are labeled with the labels $1, 2, ..., n-k$, and the remaining $k-1$ edges repeat labels from the same set. There is a bijection between the possibilities of repeated edge labels and the partitions of the integer $k-1$. Specifically, each partition $p=\{a_1, \ldots, a_h\}$ ($\sum_{i=1}^{h} a_i = k-1$), corresponds to h repeated labels in the spanning tree, such that the ith repeated label appears a_i+1 times. In such case, we say that the tree is "labeled according to p." In a spanning tree of $\gamma(P)$, we distinguish a repeated label i by $i, i', \cdots$. However, when considering $\gamma(P)$, repeated labels are not distinguished. The number and length of the cycles in $\gamma(P)$ are bounded due to the limited multiplicity of labels.

In order to compute $|\mathcal{P}_n|$, we consider all possible directed edge-labeled trees of size n with edge labels as conjectured, and count only those that represent valid polycubes. In this process two things might happen: (a) cells may coincide (Figures 2(a,d)). A tree with overlapping cells is invalid; and (b) two cells which are not connected by a tree edge may be adjacent (Figures 2(b,e)). Such a tree corresponds to a polycube P with cycles in $\gamma(P)$, hence, its spanning tree is not unique. We consider several small structures which are contained in these trees, and which cause the problems above. A *distinguished* structure is the union of all paths that connect two coinciding or adjacent cells. We design an algorithm for producing $\mathcal{DS}_k$; the set of all distinguished structures in $n-k$ dimensions.

Lemma 2 *Let $\sigma \in \mathcal{DS}_k$ be composed of $k^* \geq 1$ trees $s_1, \ldots, s_{k^*}$ with n^* vertices and distinct edge labels. The number of occurrences of σ in trees of size n with distinct edge labels is $\left(\prod_{i=1}^{k^*} |s_i|\right) \frac{(n-n^*+k^*-1)!}{(n-n^*)!} n^{n-n^*+k^*-2}$.*

We build an inclusion-exclusion graph that contains a vertex corresponding to each structure $\sigma \in \mathcal{DS}_k$ and an edge $e=\sigma_1 \rightarrow \sigma_2$ labeled with c if σ_1 contains c occurrences of σ_2. A simple bottom-up procedure implements Lemma 2, and computes, for every node $u \in \mathcal{V}$, $T_p(s(u))$; the number of directed trees labeled according to p that contain *only* the structure $s(u)$ as a subtree.

For $P \in \mathcal{P}$, $\gamma(P)$ can be a tree (if P is a tree), or it can contain cycles. Every tree polycube gives rise to a unique spanning tree. For every possibility of repeated labels p, let $\mathrm{DT}_p(n)$ denote the number of spanning trees of all tree polycubes that are labeled according to p. Let also T_p denote the total number of directed trees with n vertices labeled according to p. It can be shown that $T_p = \mathit{constant} * \mathit{poly}(n) * 2^{n-1} n^{n-3}$. Every such tree corresponds to a tree polycube unless it contains a structure $\sigma \in \mathcal{DS}_k$ as a subtree. Thus, we exclude all the trees that contain every $\sigma \in \mathcal{DS}_k$ as a subtree.Therefore, $\mathrm{DT}(n, n-k) = \sum_p \mathrm{DT}_p(n) = \sum_p T_p - \sum_{\sigma \in \mathcal{DS}_k} T_p(\sigma)$.

Let $\mathcal{C}$ denote the set of all cycle structures of polycubes proper in $n-k$ dimensions. $\mathcal{C}$ can be found using $\mathcal{DS}_k$. For any $\mathcal{C}_i \in \mathcal{C}$, let $P_{\mathcal{C}_i}$ denote the number of polycubes $P \in \mathcal{P}_n$ that contain $\mathcal{C}_i$ in $\gamma(P)$. Let $\sigma \in \mathcal{DS}_k$ have c occurrences in $\mathcal{C}_i$. Then, $P_{\mathcal{C}_i} = \sum_{p \in \Pi(k-1)} \frac{T_p(\sigma)}{c}$. Finally, $\mathrm{DX}(n, n-k) = \mathrm{DT}(n, n-k) + \sum_{i=1}^{|\mathcal{C}|} P_{\mathcal{C}_i}$.

3 Results

The entire method was automated in a C++ program, using `Mathematica` for simplifying the final formulae. The program was run successfully for $k \leq 4$, and our results exactly match the formulae conjectured in the literature of statistical physics. For $k=4$, the parallel computation took about 15 minutes on a supercomputer with 16 processors and 65 GB of RAM. The program found 8,397 distinguished structures, and 179 cycles, and produced data files which document the entire computation, serving as a proof of Theorem 1.

References

[1] A. Asinowski, G. Barequet, R. Barequet, and G. Rote, Proper n-cell polycubes in $n-3$ dimensions, *J. of Int. Sequences*, 15 (2012), #12.8.4.

[2] R. Barequet, G. Barequet, and G. Rote, Formulae and growth rates of high-dimensional polycubes, *Combinatorica*, 30 (2010), 257–275.

[3] S.R. Broadbent and J.M. Hammersley, Percolation processes: I. Crystals and mazes, *Proc. Cambridge Philosophical Society*, 53 (1957), 629–641.

[4] W.F. Lunnon, Counting multidimensional polyominoes, *The Computer Journal*, 18 (1975), 366–367.

[5] S. Luther and S. Mertens, Counting lattice animals in high dimensions, *J. of Statistical Mechanics: Theory and Experiment*, 9 (2011), 546–565.

[6] *OEIS*, available at `http://oeis.org` .

[7] P.J. Peard and D.S. Gaunt, $1/d$-expansions for the free energy of lattice animal models of a self-interacting branched polymer, *J. Phys., A: Math. & Gen.*, 28 (1995), 6109–6124.

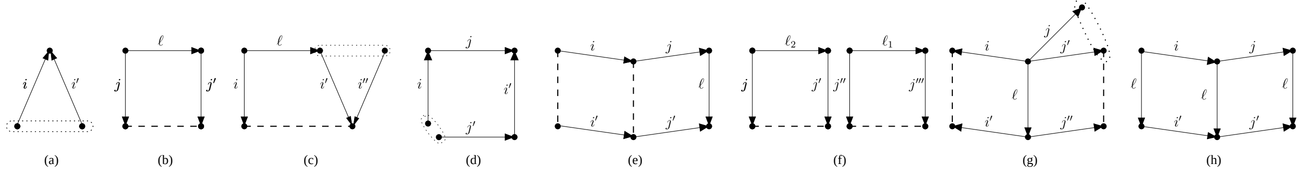


Figure 2: (a–g) A few distinguished structures for k=4; (h) A cycle structure. A dotted line is drawn between every pair of neighboring cells and around every pair of coinciding cells.

1-String B_1-VPG Representations of Planar Partial 3-Trees

Therese Biedl Martin Derka*

Abstract

Planar partial 3-trees are subgraphs of those planar graphs obtained by repeatedly inserting a vertex of degree 3 into a face. In this paper, we show that partial 3-trees have a 1-string B_1-VPG representation, i.e., a representation where every vertex is represented by an orthogonal curve with at most one bend, every two curves intersect at most once, and intersections of curves correspond to edges in the graph.

1 Introduction

A *string representation* is a representation of a graph where every vertex v is assigned a curve $\mathbf{v}$. Vertices u, v are connected by an edge if and only if curves $\mathbf{u}, \mathbf{v}$ intersect. A 1-*string representation* is a string representation where every two curves intersect at most once.

Some previous papers (e.g. [1, 5, 6]) studied string representations that use *orthogonal curves*, i.e., curves consisting of vertical and horizontal segments. These are called B_k*-VPG representations*[1] if every curve has at most k bends. See e.g. [2] for more related results. Felsner et al. [6] showed that every planar 3-tree has a B_1-VPG representation. Moreover, every vertex-curve has the shape of an `L`, which implies that vertex-curves intersect at most once, so the result is a 1-string B_1-VPG representation. In this note, we sketch how to extend the result to more graphs, and in particular, show:

Theorem 1 *Every planar partial 3-tree G has a 1-string B_1-VPG representation.*

Planar partial 3-trees (defined in Section 2) are the same as planar graphs of treewidth at most 3, and include outer-planar graphs, Apollonian networks, series-parallel graphs, Halin graphs and IO-graphs.

The proof of this theorem (see Section 3) uses all 4 possible shapes of 1-bend orthogonal curves. For some of the aforementioned subclasses of planar partial 3-trees, we can show that `L`s suffice for vertex-curves (details are omitted in this short note). Generalizing this to all planar partial 3-trees remains an open problem.

*David R. Cheriton School of Computer Science, University of Waterloo, Canada. `{biedl,mderka}@uwaterloo.ca`. Research of T.Biedl supported by NSERC. Research of M.Derka supported by an NSERC Vanier CGS.

[1]VPG is an acronym for Vertex-Path-Grid since vertices are represented by paths in a rectangular grid.

2 Definitions

A *planar graph* is a graph that can be drawn without crossings. If one such drawing Γ is fixed, then a *face* is a maximal connected region of $\mathbb{R}^2 - \Gamma$. The *outer face* corresponds to the unbounded region; the *interior faces* are all other faces.

A 3-*tree* is a graph that is either a triangle or has a vertex order $v_1, \ldots, v_n$ such that for $i \geq 4$, vertex v_i is adjacent to exactly three predecessors and they form a triangle. A *partial* 3-*tree* is a subgraph of a 3-tree.

Our proof of Theorem 1 employs the method of "private regions" used previously for various string representation constructions [2, 4, 6]. We define the following:

Definition 1 *An* F-shaped area *is a region bounded by a 10-sided polygon with CW or CCW sequence of interior angles* 90°, 270°, 90°, 90°, 270°, 270°, 90°, 90°, 90° *and* 90°. *A* rectangle-shaped area *is a region bounded by an axis-aligned rectangle.*

Definition 2 *Given a* 1-*string representation, a* private region *of vertices $\{a, b, c\}$ is an F-shaped or rectangle-shaped area that intersects (up to permutation of names) curves $\mathbf{a}, \mathbf{b}, \mathbf{c}$ in the way depicted in Figure 1(a), and that intersects no other curves and private regions.*

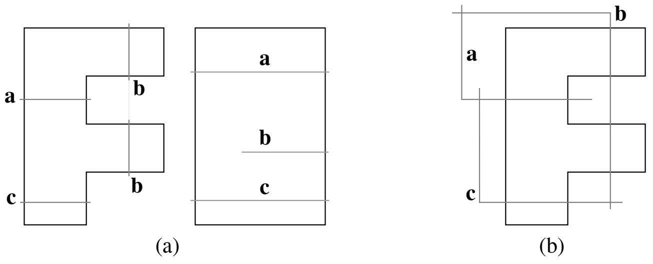


Figure 1: (a) An F-shaped (left) and rectangle-shaped (right) private region of $\{a, b, c\}$. (b) The base case. Intersections among $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ can be omitted as needed.

3 Proof of Theorem 1

Let G be a planar partial 3-tree. By definition, there exists a 3-tree H for which G is a subgraph. One can show [3] that we may assume H to be planar. Let $v_1, \ldots, v_n$ be a vertex order of H such that for $i \geq 4$ vertex v_i is adjacent to 3 predecessors that form a triangle. In particular, v_4 is incident to a triangle formed by $\{v_1, v_2, v_3\}$. One can show (see e.g. [3]) that the vertex order can be chosen in such a way that $\{v_1, v_2, v_3\}$ is the outer face of H in some planar drawing.

For $i \geq 3$, let G_i and H_i be the subgraphs of G (respectively H) induced by vertices $v_1, \ldots, v_i$. We prove Theorem 1 by showing by induction on i:

> G_i has a 1-string B_1-VPG representation with a private region for every interior face of H_i.

In the base case, $i = 3$ and $G \subseteq K_3 \simeq H$. Construct a representation R and find a private region for the unique interior face of H as depicted in Figure 1(b).

Now consider $i \geq 4$. By induction, construct a representation R_0 of G_{i-1} that contains a private region for every interior face of H_{i-1}.

Let $\{a, b, c\}$ be the predecessors of v_i in H. Recall that they form a triangle. Since H is planar, this triangle must form a face in H_{i-1}. Since $\{v_1, v_2, v_3\}$ is the outer face of H (and hence also of H_{i-1}), the face into which v_i is added must be an interior face, so there exists an interior face $\{a, b, c\}$ in H_{i-1}. Let P_0 be a private region that exists for $\{a, b, c\}$ in R_0; it can have the shape of an F or a rectangle.

Observe that in G, vertex v_i may be adjacent to any possible subset of $\{a, b, c\}$. This gives 16 cases (two possible shapes, up to rotation and reflection, and 8 possible adjacencies).

In each case, the goal is to place a curve $\mathbf{v_i}$ inside R_0 such that it intersects exactly the curves of the neighbours of v_i in $\{a, b, c\}$ and none else. Furthermore, having placed $\mathbf{v_i}$ into R_0, we need to find a private region for the three new interior faces in H_i, that is, the three faces formed by v_i and two of $\{a, b, c\}$.

We omit detailed descriptions of the constructions and refer the reader to Figure 2. In all cases we achieve the goal, and Theorem 1 holds by induction.

References

[1] A. Asinowski, E. Cohen, M. Golumbic, V. Limouzy, M. Lipshteyn, and M. Stern. Vertex intersection graphs of paths on a grid. *J. Graph Algorithms Appl.*, 16(2):129–150, 2012.

[2] T. Biedl and M. Derka. 1-string B_2-VPG representations of planar graphs. *Symposium on Computational Geometry (SoCG'15)*, 2015. To appear.

[3] T. Biedl and L.E.R. Velázquez. Drawing planar 3-trees with given face areas. *Comput. Geom.*, 46(3):276–285, 2013.

[4] J. Chalopin, D. Gonçalves, and P. Ochem. Planar graphs have 1-string representations. *Discrete & Computational Geometry*, 43(3):626–647, 2010.

[5] S. Chaplick, V. Jelínek, J. Kratochvíl, and T. Vyskocil. Bend-bounded path intersection graphs: Sausages, noodles, and waffles on a grill. In *WG'12*, volume 7551 of *LNCS*, pages 274–285, 2012.

[6] S. Felsner, K.B. Knauer, G.B. Mertzios, and T. Ueckerdt. Intersection graphs of L-shapes and segments in the plane. In *MFCS'14*, volume 8635 of *LNCS*, pages 299–310, 2014.

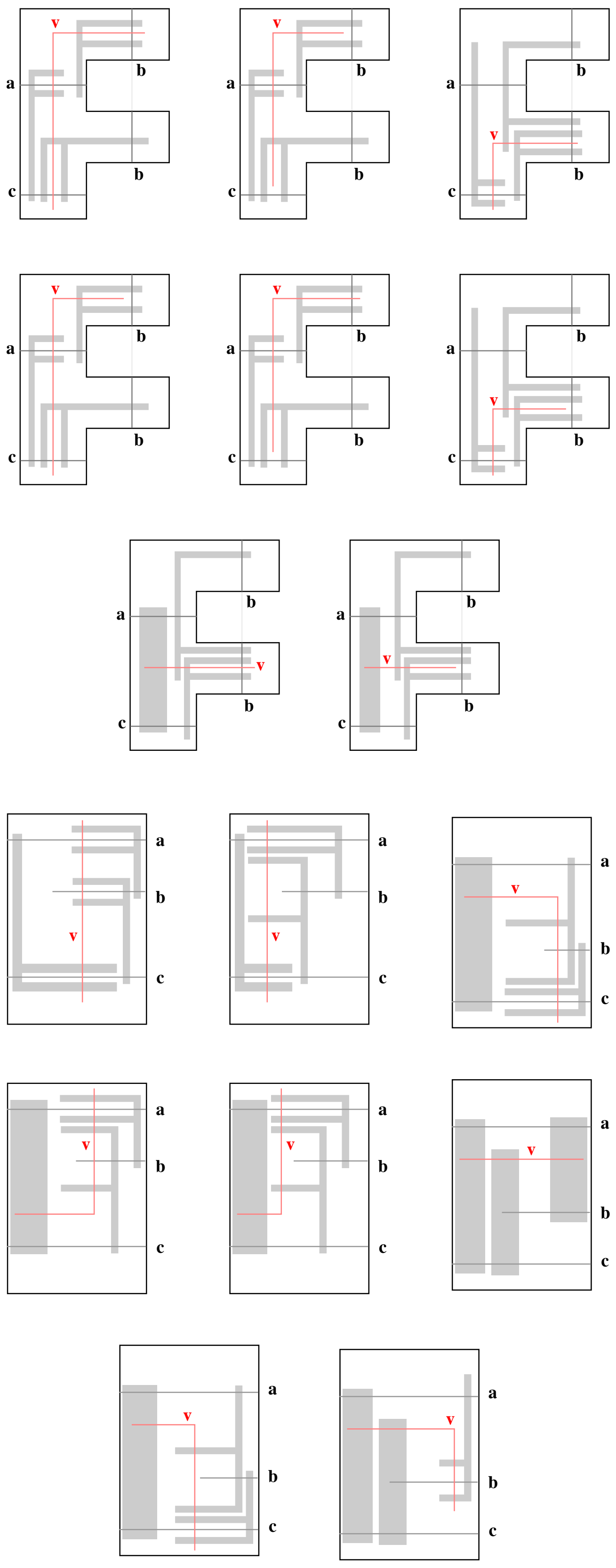


Figure 2: Inserting a curve $\mathbf{v}$ for $v = v_i$ into an F-shaped and rectangle-shaped private region.

Flips in Edge-Labelled Pseudo-Triangulations[*]

Prosenjit Bose[§] Sander Verdonschot[§]

Abstract

We show that $O(n^2)$ flips suffice to transform any edge-labelled pointed pseudo-triangulation into any other.

1 Introduction

A *pseudo-k-gon* is a weakly simple polygon with k convex interior angles, called *corners*, that are connected by reflex chains. Given a set of points P in the plane, a pseudo-triangulation of P is a subdivision of its convex hull into pseudo-triangles. A pseudo-k-gon or pseudo-triangulation is *pointed* if all vertices are incident to a reflex angle in some face (including the outer face; see Figure 1b for an example). All pseudo-triangulations considered in this paper are pointed. Pseudo-triangulations find applications in areas such as kinetic data structures [2] and rigidity theory [3].

A diagonal of a pseudo-k-gon is called a *bitangent* if the pseudo-k-gon remains pointed after insertion of the edge. Every internal edge of a pseudo-triangulation can be *flipped*. A flip removes the edge, leaving a pseudo-quadrilateral, and inserts the unique other bitangent (see Figure 1a). Bereg [1] showed that $O(n \log n)$ flips suffice to transform any pseudo-triangulation into any other, where n is the number of vertices.

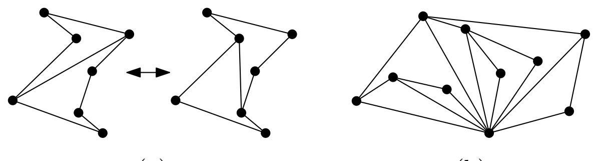


Figure 1: (a) A flip in a pseudo-quadrilateral. (b) A left-shelling pseudo-triangulation.

In this paper, we investigate flips in pseudo-triangulations where each internal edge has a unique label in $\{1, \ldots, |E|\}$. After a flip, the new edge receives the label of the flipped edge. We show that we can transform any edge-labelled pseudo-triangulation into any other with $O(n^2)$ flips. This is harder than the unlabelled problem, since we no longer have the freedom to choose the mapping between edges in the initial and final pseudo-triangulation. Before we can start the proof, we need a few more definitions.

Given a set of points P in the plane, let v_0 be the point with the lowest y-coordinate, and let $v_1, \ldots, v_n$ be the other points in clockwise order around v_0. The *left-shelling* pseudo-triangulation is the union of the convex hulls of $v_0, \ldots, v_i$, for all $2 \leq i \leq n$ (see Figure 1b). Thus, every vertex is associated with two edges: a *bottom* edge connecting it to v_0 and a *top* edge that is tangent to the convex hull of the earlier vertices. The *right-shelling* pseudo-triangulation is similar, with the vertices added in counter-clockwise order instead.

2 Tools

We first describe two tools, called a *sweep* and a *shuffle*, that play a crucial role in the proof of the main result.

Lemma 1 *We can interchange the labels of the edges incident to a vertex v of degree 2 with 3 flips.*

Proof Sketch. Removing v leaves a pseudo-triangle T. There are three bitangents that connect v to T, each corresponding to the geodesic between v and a corner of T. Any choice of two of these bitangents results in a pointed pseudo-triangulation. When one of them is flipped, the only new edge that can be inserted so that the result is still a pointed pseudo-triangulation is the bitangent that was not there before the flip. Thus, we can interchange the labels with 3 flips (see Figure 3). □

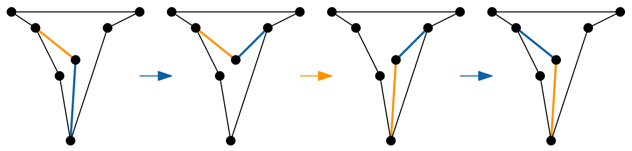


Figure 3: Interchanging the labels of the edges incident to a vertex of degree 2.

Lemma 2 (Sweep) *In the left-shelling pseudo-triangulation, we can interchange the labels of any number of internal top edges and their corresponding bottom edges with $O(n)$ flips.*

Proof Sketch. Let S be the set of vertices whose internal top edge should have their label swapped with the corresponding bottom edge. Consider a ray L from v_0 that starts at the positive x-axis and sweeps through the point set to the negative x-axis. We will maintain the following invariant: the graph induced by the vertices to the left of L is their left-shelling pseudo-triangulation

[*]This work was partially supported by NSERC.

[§]School of Computer Science, Carleton University, Ottawa

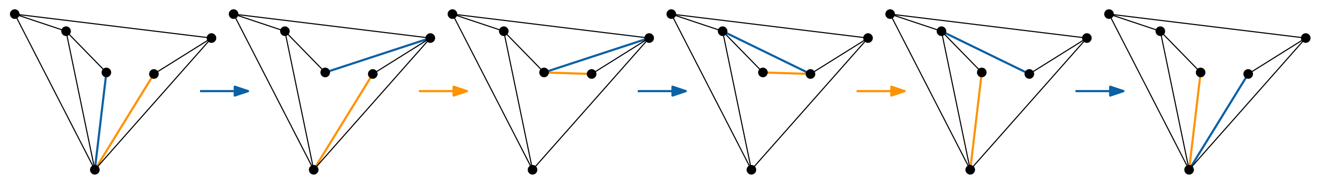


Figure 2: Interchanging the labels of two bitangents of a pseudo-pentagon with five bitangents.

and the graph induced by the vertices to the right of L is their right-shelling pseudo-triangulation. Furthermore, the labels of the top edges of the vertices in S to the right of L have been interchanged with their respective bottom edges. This invariant is satisfied at the start.

Suppose that L is about to pass vertex v_k. If v_k is on the convex hull, its top edge is not internal and no action is required for the invariant to hold after passing v_k. If v_k is not on the convex hull, it has degree 2 and flipping its top edge results in the tangent from v_k to the convex hull of the points to the right of L – exactly the edge needed to add v_k to their right-shelling pseudo-triangulation. In addition, if $v_k \in S$, we can use Lemma 1 to swap the labels of its top and bottom edge with 3 flips. Thus, with $O(1)$ flips, the invariant still holds after passing v_k.

At the end, we have constructed the right-shelling pseudo-triangulation and swapped the desired edges. An analogous transformation without any swapping can transform it back into the left-shelling pseudo-triangulation with $O(n)$ flips in total. □

Lemma 3 *In the left-shelling pseudo-triangulation, we can interchange the labels of two consecutive bottom edges with $O(1)$ flips.*

Proof Sketch. When we remove the two consecutive bottom edges (say a and b), we are left with a pseudo-pentagon P, which can have up to five bitangents. If P has exactly five bitangents, each geodesic between two corners of P corresponds to exactly one bitangent, which implies that the bitangents of P can be swapped just like diagonals of a convex pentagon (see Figure 2). If the pseudo-triangle to the right of b is a triangle, P already has five bitangents. Otherwise, the top endpoint of b is an internal vertex of degree 2 and we can flip its top edge to obtain a new pseudo-pentagon that does have five bitangents. After swapping the labels of a and b, we can flip this top edge back. Thus, in either case we can interchange the labels of a and b with $O(1)$ flips. □

We can use Lemma 3 to reorder the labels of the bottom edges with insertion or bubble sort, as these algorithms only swap adjacent values.

Corollary 4 (Shuffle) *In the left-shelling pseudo-triangulation, we can reorder the labels of all bottom edges with $O(n^2)$ flips.*

3 Upper bound

Theorem 5 *We can transform any edge-labelled pseudo-triangulation with n vertices into any other with $O(n^2)$ flips.*

Proof Sketch. We show how to transform any edge-labelled pseudo-triangulation into a canonical one with $O(n^2)$ flips. The result follows by the reversibility of flips. The canonical pseudo-triangulation is the left-shelling pseudo-triangulation, with the bottom edges labelled in clockwise order around v_0, followed by the top edges in the same order (based on their associated vertex). We call the labels assigned to bottom edges *low*, and the labels assigned to top edges *high*. We first ignore the labels and transform the given pseudo-triangulation into the left-shelling pseudo-triangulation with $O(n \log n)$ flips.

In the first step, we use a shuffle (see Corollary 4) to match every bottom edge with a high label with a top edge with a low label. Then we exchange these pairs of labels with a sweep (see Lemma 2). Now all bottom edges have low labels and all top edges have high labels, so all that is left is to sort the labels. We can sort the low labels with a second shuffle. To sort the high labels, we sweep them to the bottom edges, shuffle to sort them there, then sweep them back. This results in the canonical triangulation. Since we used a constant number of shuffles and sweeps, it takes $O(n^2)$ flips in total. □

Acknowledgements

We would like to thank Anna Lubiw and Vinayak Pathak for helpful discussions.

References

[1] S. Bereg. Transforming pseudo-triangulations. *Inform. Process. Lett.*, 90(3):141–145, 2004.

[2] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *Internat. J. Comput. Geom. Appl.*, 12(1-2):3–27, 2002.

[3] I. Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete Comput. Geom.*, 34(4):587–635, 2005.

Realization of simply connected polygonal linkages[*]

Clinton Bowen[†] Stephane Durocher[‡] Maarten Löffler[§] Anika Rounds[¶] André Schulz[||] Csaba D. Tóth[†¶]

Abstract

We consider body-and-joint frameworks. We show that it is strongly NP-hard to decide whether a given polygonal linkage (body-and-bar framework) is realizable in the plane when the bodies are convex polygons and their contact graph is a tree; the problem is weakly NP-hard already for a chain of rectangles; but efficiently decidable for a chain of triangles hinged at distinct vertices.

1 Introduction

Complex structures in nature are often composed of elementary pieces that obey simple local composition rules. Molecular biology, nanomanufacturing, and self-assembly are prime examples. Mathematical models for this phenomenon typically rely on rigidity theory and formal languages. In this paper, we study the realizability of complex structures that are specified by their local geometry.

A *polygonal linkage* is a set $\mathcal{P}$ of convex polygons and a set H of hinges where each hinge $h \in H$ corresponds to two points on the boundaries of distinct polygons in $\mathcal{P}$. A *realization* of a polygonal linkage is an interior-disjoint placement of congruent copies of the polygons in $\mathcal{P}$ such that the points corresponding to each hinge are identified. A *realization with orientation* uses only translated or rotated copies of the polygons in $\mathcal{P}$ (no reflections) and for each hinge, the cyclic order of incident polygons is given. The topology of a polygonal linkage can be represented by the *hinge graph*, a bipartite graph where the vertices correspond to polygons in $\mathcal{P}$ and the hinges in H, and edges represent the polygon-hinge incidences.

The *Polygonal Linkage Realizability (PLR)* problem asks whether a given polygonal linkage admits a realization; and *PLR with fixed orientation* asks whether it admits a realization with a given orientation. *PLR* allows the use of reflections at hinges, whereas *PLR with fixed orientation* only allows rotations and translations of the polygons in $\mathcal{P}$.

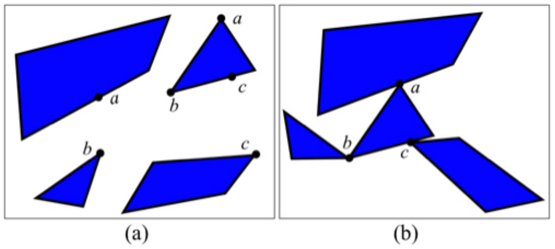


Figure 1: (a) A set of convex polygons and hinges. (b) A realization of the polygonal linkage (with fixed orientation).

In general these problems are known to be NP-hard (see details below). However, the hardness reductions crucially rely on graphs with a large number of cycles. In this paper we consider these problems for simply connected topologies, where the hinge graph is a tree.

Summary of results. Our main result is that the realizability problems remains NP-hard for simply connected polygonal linkages. The only exceptions are chains of triangles or rectangles hinged at distinct vertices. Some variants are always realizable, some have easy hardness reductions, and some reductions required substantial new machinery.

Related Previous Work. Polygonal linkages are a generalization of classical linkages (bar-and-joint frameworks) in rigidity theory. A linkage is a graph $G = (V, E)$ with given edge lengths. A realization of a linkage is a (crossing-free) straight-line embedding of G in the plane. Bhatt and Cosmadakis [3] proved that the realizability of linkages is NP-hard.

Note that every *tree* linkage can be realized in $\mathbb{R}^2$ with almost collinear edges. According to the celebrated *Carpenter's Rule Theorem* [4, 6] every realization of a path (or a cycle) linkage can be continuously moved (without self-intersection) to any other realization. However, there are trees of maximum degree 3 with as few as 8 edges whose realization space is disconnected [2]; and deciding whether the realization space of a tree linkage is connected is PSPACE-complete [1]. Connelly et al. [5] showed that the Carpenter's Rule Theorem generalizes to certain polygonal linkages, which are obtained by replacing the edges of a path linkage with special polygons. For more details see the full version.

[*]Our results were developed at the *First International Workshop on Drawing Algorithms for Networks of Changing Entities (DANCE 2014)*, held in Langbroek, the Netherlands, and supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208. Research by Rounds and Tóth was supported in part by the NSF awards NSF CCF-1422311 and CCF-1423615.

[†]California State University Northridge, Los Angeles, CA

[‡]University of Manitoba, Winnipeg, Canada

[§]Utrecht University, the Netherlands

[¶]Tufts University, Medford, MA, USA

[||]Münster University, Germany

2 PLR for Chains of Polygons

In this section we consider polygonal linkages whose hinge graph is a path. We call such a linkage a *chain of polygons*, given by a sequence of convex polygons $(P_1, \ldots, P_n)$ and $n-1$ hinges, where the ith hinge corresponds to a pair of points on the boundaries of P_i and P_{i+1} for $i = 1, \ldots, n-1$. A line L is said to be *tangent* to polygon P at a boundary point $h \in P$ if L passes through h but avoids the interior of P. We formulate a simple sufficient condition for the realizability of a chain of polygons.

Proposition 1 *Consider a chain of convex polygons $(P_1, \ldots, P_n)$ with $n-1$ hinges. If P_i admits parallel tangent lines through all of its hinges for $i = 1, \ldots, n$ then the chain of polygons is realizable with fixed orientation. A realization can be computed in $O(n)$ time.*

Proof. Rotate the polygons $P_1, \ldots, P_n$ such that they each admit vertical tangents through their hinges; the hinge of P_1 is a rightmost point in P_1; and for $i = 2, \ldots, n$ the common hinge of P_{i-1} is a leftmost point of P_i. (Refer to Fig. 2). Then translate the polygons such that the corresponding hinges coincide. For the common hinge h_i between P_i and P_{i+1}, all polygons P_j, $j \leq i$, lie on the left of h_i, and all polygons $P_{j'}$, $i+1 \leq j'$, lie on the right of h_i. Consequently, the polygons have disjoint interiors as required. We can compute the tangents of P_i at both hinges, hence a suitable rotation angle, in $O(1)$ time, so the total time is $O(n)$. □

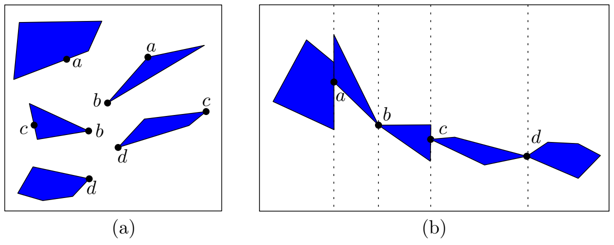


Figure 2: (a) A chain of 5 polygons with 4 hinges. (b) A realization of (a) where the parallel tangents are vertical.

Corollary 2 *Every chain of triangles hinged at distinct vertices is realizable with fixed orientation.*

Surprisingly the realizability of a chain of arbitrary polygons is already NP-hard, even if the polygons are convex quadrilaterals hinged at vertices or triangles hinged at arbitrary boundary points. We reduce the problem from **Partition**, which is weakly NP-hard (i.e., NP-hard when the input is a sequence of n integers between 1 and 2^n).

Theorem 3 *It is weakly NP-hard to decide whether a chain of rectangles is realizable.*

Theorem 4 *It is weakly NP-hard to decide whether a chain of convex polygons is realizable with fixed orientation.*

We give NP-hardness proofs for these in the full paper.

3 PLR with Fixed Orientation

Theorem 5 *It is strongly NP-hard to decide whether a polygonal linkage whose hinge graph is a* **tree** *can be realized with fixed orientation.*

Our proof for Theorem 5 is a reduction from **Planar-3-SAT** (P3SAT). See Fig. 3(left).

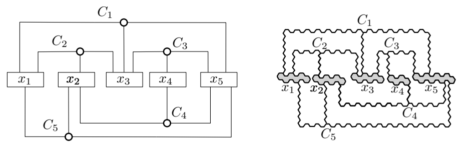


Figure 3: Left: the associated graph $A(\Phi)$ for a Boolean formula Φ. Right: the schematic layout of the variable, clause, and transmitter gadgets in our construction.

The big picture. Given an instance Φ of P3SAT with n variables and m clauses, we construct a simply connected polygonal linkage $(\mathcal{P}, H)$ of polynomial size in n and m such that Φ is satisfiable iff $(\mathcal{P}, H)$ admits a realization with fixed orientation. We construct a polygonal linkage in two main steps: First, we construct an auxiliary structure where some of the polygons have fixed position in the plane (called *obstacles*), while other polygons are flexible hinged to an obstacle. Second, we modify the auxiliary construction into a polygonal linkage by allowing the obstacles to move freely, and by adding new polygons and hinges as well as an exterior *frame* that holds the obstacle polygons in place. All polygons in our constructions are regular hexagons or long and skinny rhombi because these are the polygons that we can "simulate" with coin graphs. The details of our construction can be found in the full version.

References

[1] H. Alt, C. Knauer, G. Rote, and S. Whitesides, On the complexity of the linkage reconfiguration problem, in *Towards a Theo. of Geom. Graphs*, AMS, 2004, 1–14.

[2] B. Ballinger, D. Charlton, E. D. Demaine, M. L. Demaine, J. Iacono, C.-H. Liu, S.-H. Poon, Minimal locked trees, in *Proc. 11th WADS*, LNCS 5664, Springer, 2009, 61–73.

[3] S. N. Bhatt and S. S. Cosmadakis, The complexity of minimizing wire lengths in VLSI layouts, *Inform. Process. Lett.* **25** (4) (1987), 263–267.

[4] R. Connelly, E. D. Demaine, and G. Rote, Straightening polygonal arcs and convexifying polygonal cycles. *Discrete Comput. Geom.* **30** (2) (2003), 205-239.

[5] R. Connelly, E. D. Demaine, M. L. Demaine, S. P. Fekete, S. Langerman, J. S. B. Mitchell, A. Ribó, G. Rote, Locked and unlocked chains of planar shapes, *Discrete Comput. Geom.* **44** (2010), 439–462.

[6] I. Streinu, Pseudo-triangulations, rigidity and motion planning, *Disc. Comput. Geom.* **34** (4) (2005), 587–635.

[7] H. Whitney, Congruent graphs and the connectivity of graphs, *Amer. J. Math.* **54** (1932), 150–168.

Model-based Classification of Trajectories

Maike Buchin[*] Stef Sijben[*]

Abstract

We present algorithms for classifying trajectories based on a movement model parameterized by a single parameter. Classification is the problem of assigning trajectories to classes of similar movement characteristics. We give an efficient algorithm to compute an optimal classification for a discrete set of parameter values. Although classification is NP-hard if the parameter values are allowed to vary continuously, we present an algorithm that solves the problem in polynomial time under mild assumptions on the input.

1 Introduction

Advances in tracking technology lead to increasing amounts of trajectory data, i.e., sequences of time-stamped positions of a moving entity. We study the fundamental task of classifying trajectory data. A *classification* of a set of trajectories $\mathcal{T}$ is a partition of $\mathcal{T}$ into disjoint *classes* that cover $\mathcal{T}$. In movement ecology, the aim of classification is to discover classes of similar movement behaviour [3]. For instance, Figure 1 shows a classification of fisher (animal) data by our algorithm. We tackle this problem by fitting a parameterized movement model to the data. Taking such an approach is essential for applications – such as ecology– that use movement data in a statistical analysis.

Movement models are used to infer a continuous motion from discrete samples of the movement path. In ecology, mostly random movement models, like the Brownian bridge movement model (BBMM) [1], are used. These models have a parameter describing how quickly the probability distribution of the position diffuses. To determine the optimal value of this parameter, a maximum likelihood method is used, which is based on the probability of observing the given trajectories, conditioned on the parameter value. A classification $\mathcal{C}$ has a log-likelihood, which is defined as $L_{\mathcal{C}} = \sum_{C\in\mathcal{C}} \sum_{\tau\in C} L_\tau(x(C))$. That is, the log-likelihood of each class C is the sum over the log-likelihoods of its trajectories τ, while the log-likelihoods of the classes are summed to obtain the log-likelihood of $\mathcal{C}$.

We could now define an optimal classification as one that maximizes the log-likelihood, but then it would be optimal to put each trajectory into its own class, resulting in the largest possible number of degrees of freedom for the model. To avoid this, an information criterion (IC) is frequently used [2]. An IC assigns a value to each classification based on its likelihood and the number of classes. In particular we consider ICs of the form $IC(\mathcal{C}) = -2L_{\mathcal{C}} + |\mathcal{C}| \cdot p$, where $L_{\mathcal{C}}$ is the log-likelihood of the model instance and $|\mathcal{C}|$ is the number of classes. The number p is a penalty factor for adding complexity to the model that counteracts overfitting. We now define an optimal classification to be one that minimizes the value of the IC among all classifications and parameter values for $\mathcal{T}$.

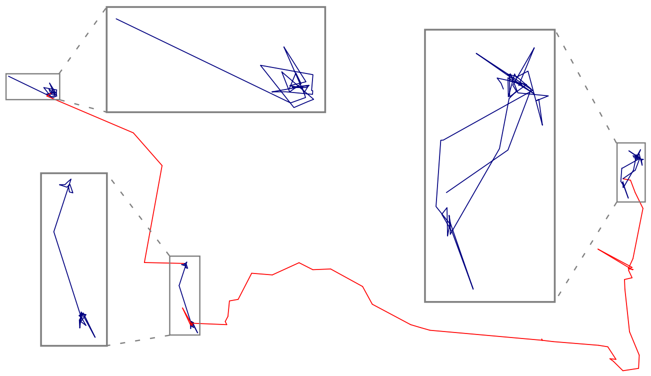


Figure 1: Example of a classification. Classes indicated by colour.

We develop efficient algorithms to compute optimal classifications. We first consider a discrete case, where the parameter values $x(C_i)$ are from a finite set $X = \{x_1, \ldots, x_m\}$ (in sorted order). Then we consider the continuous case where the $x(C_i)$ come from an interval on the real line.

Preliminaries. When classifying a set of trajectories $\mathcal{T}$, we use ℓ to denote the number of classes and $C_1, \ldots, C_\ell \subseteq \mathcal{T}$ to denote the classes. A class C_i is assigned a value of the model parameter $x(C_i)$. Each trajectory $\tau_i \in \mathcal{T}$ has a log-likelihood function $L_i(x)$, which we assume to be bitonic. We also assume w.l.o.g. that these functions are given in increasing order by the parameter value at which they reach their maximum. That is, if we define $M_i := \arg\max_x(L_i(x))$, then $i < j \Rightarrow M_i \leq M_j$.

Problem statement. Given a set of trajectories $\mathcal{T}$, an *optimal classification* $\mathcal{C}_{opt}$ is the classification $\{C_1, \ldots, C_\ell\}$ and selection of model parameters for the classes $x(C_i)$, $1 \leq i \leq \ell$, that achieves the minimum value for the information criterion among all classifications and parameter values for $\mathcal{T}$. Our goal is to compute optimal classifications.

[*]Department of Mathematics, Ruhr-Universität Bochum, Germany, {Maike.Buchin,Stef.Sijben}@rub.de

2 Discrete Classification

We compute the optimal classification in the discrete case by dynamic programming. A natural approach would be to process the trajectories in the order by their maximum. However it is not necessarily the case that this order is reflected in the classes they are associated with.

Observation 1 *Assume an optimal classification assigns the parameter values $x(C_1) < \cdots < x(C_\ell)$ to the classes. Then a trajectory that reaches its maximum likelihood in the interval $[x(C_j), x(C_{j+1}))$ will be either assigned to C_j or C_{j+1} by the bitonicity of the likelihood function. In particular if some $x(C_i)$ is selected then $x(C_j)$ with $j < i$ does not depend on any of the trajectories with maximum $\geq x(C_i)$.*

Using this, we can compute an optimal classification by dynamic programming as follows. Consider a set of log-likelihood functions $\{L_1, \ldots, L_k\}$ and candidate parameter values $\{x_1, \ldots, x_m\}$. We add dummy values $x_0 := -\infty$ and $x_{m+1} := \infty$. Let $\mathcal{L}_{i,j} := \{L_l \mid x_i \leq M_l < x_j\}$ with $0 \leq i < j \leq m+1$ denote the set of functions that reach their maximum value between x_i and x_j. We define Opt_i as the optimal classification of $\mathcal{L}_{0,i}$, conditioned on the fact that the last class C_ℓ of Opt_i has parameter value $x(C_\ell) = x_i$.

We can iteratively construct Opt_i for $i = 1, 2, \ldots, m$ by noting that the second to last class has a parameter value $x_j \in \{x_1, x_2, \ldots, x_{i-1}\}$. Thus, using Observation 1, Opt_i consists of Opt_j extended with a new class C_ℓ with $x(C_\ell) = x_i$. The functions that were already classified in Opt_j stay in the same class, and the functions in $\mathcal{L}_{j,i}$ are assigned to either $C_{\ell-1}$ or C_ℓ. Opt_i is computed by trying all possible values of j and selecting the one with the lowest IC. The optimal classification of the whole input is one of the Opt_i, extended by adding the remaining functions to the last class.

Theorem 1 *The optimal classification of k trajectories with respect to an information criterion can be computed in $O(m^2 + km(\log m + \log k))$ time and $O(km)$ space, where m is the number of candidate parameter values.*

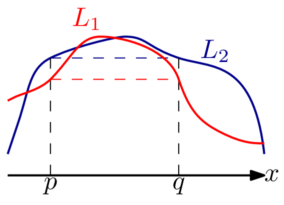
3 Continuous Classification

When the parameter values are taken from a continuous domain $\mathcal{D}$, finding the optimal classification is NP-hard for arbitrary bitonic functions, as we have shown using a reduction from SET COVER. We now describe a polynomial time algorithm for certain log-likelihood functions.

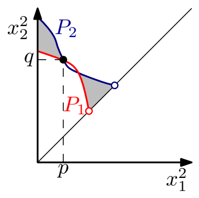
For a set of input functions $\mathcal{L} := \{L_1, \ldots, L_k\}$, let Opt_i be the optimal classification having exactly i classes. Let C^i_j denote the jth class in Opt_i (by increasing parameter value) with parameter value x^i_j and let $x^i_0 := M_1$ and $x^i_{i+1} := M_k$.

The algorithm iteratively computes Opt_{i+1} from Opt_i, for $i \in \{1, \ldots, k-1\}$. Observe that there is only one classification with one class (up to changing the parameter value).

When constructing Opt_{i+1}, we can show using Observation 1 that a function L_l is in one of only two classes of Opt_{i+1} and the parameter value x^{i+1}_j for a class C^{i+1}_j lies in $[x^i_{j-1}, x^i_j]$. A choice of parameter values for a pair of consecutive classes can be represented as a point in $\mathcal{D}^2$.



(a) Log-likelihoods L_1, L_2.



(b) The arrangement $\mathcal{A}$ induced by P_1, P_2.

Figure 2: Log-likelihood functions and their induced arrangement. Both functions have $L(p) = L(q)$, and thus $\mathcal{A}$ has a vertex at (p, q). $\mathcal{A}$ has 5 faces, the shaded faces correspond to the same classification $C^2_1 = \{L_1\}$, $C^2_2 = \{L_2\}$.

Let $P_l := \{(p, q) \in \mathcal{D}^2 \mid L_l(p) = L_l(q) \wedge p < q\}$ be the Jordan curve that separates the region with L_l in the lower class from that with L_l in the higher class. The curves $P_1, \ldots, P_k$ define an arrangement $\mathcal{A}$, where each face corresponds to the combinatorial structure of a possible classification. When constructing Opt_2, one of the faces f in $\mathcal{A}$ represents Opt_2, in the sense that $C^2_1 = \{L_l \mid P_l \text{ lies below } f\}$ and $C^2_2 = \{L_l \mid P_l \text{ lies above } f\}$. Opt_2 is determined by considering each face and optimizing x^2_1 and x^2_2 separately.

When computing Opt_{i+1} from Opt_i ($i \geq 2$), i faces of $\mathcal{A}$ jointly define Opt_{i+1}, and two faces contribute to each class. The optimal classification is computed by computing a longest path in a DAG G_i with source r and sink t. The vertices of G_i represent the possible selections of faces. An edge (u, v) of G_i fixes a set of functions $\rho_u \cup \lambda_v$ that is a candidate for C^{i+1}_j. The edge weight is the maximum log-likelihood for this class. An r–t path in G_i visits i internal vertices and represents a classification, with each edge defining a class. The log-likelihood of the classification is the path's length. So, a longest r–t path in G_i represents Opt_i.

Thus, if $\mathcal{A}$ can be computed in polynomial time, we can compute an optimal classification in polynomial time. In cases of practical interest like the BBMM, this is often true.

References

[1] J. Horne, E. Garton, S. Krone, and J. Lewis. Analyzing animal movements using Brownian bridges. *Ecology*, 88(9):2354–2363, 2007.

[2] B. Kranstauber, R. Kays, S. LaPoint, M. Wikelski, and K. Safi. A dynamic Brownian bridge movement model to estimate utilization distributions for heterogeneous animal movement. *Journal of Animal Ecology*, 2012.

[3] J. Shamoun-Baranes, R. Bom, E. E. van Loon, B. J. Ens, K. Oosterbeek, and W. Bouten. From sensor data to animal behaviour: An oystercatcher example. *PLoS ONE*, 7(5):e37997, 05 2012.

Clustering time series under the Fréchet distance

Anne Driemel[*] Amer Krivošija[†] Christian Sohler[‡]

Abstract

The problem of clustering of time series under the Fréchet distance is studied. The Fréchet distance is a popular distance measure for curves that captures well the similarities between the curves. We consider the previously posed clustering algorithms k-center and k-median and adapt them. We wish to find k cluster centers, each of complexity bounded by a constant ℓ. We give the first $(1+\varepsilon)$-approximation algorithms for these problems. Their running time is near-linear in the input size.

1 Introduction

A *time series* is a recording of a signal that changes over time. It represents a sequence of discrete measurements of a continuous signal. The signal/data can be multidimensional, but we limit ourselves to the univariate case. Examples of such data are stock market values, temperature and tide measurements, number of queries on search engines, etc.

Formally, it is a series $(w_1, t_1), \ldots, (w_m, t_m)$ of measurements w_i of a signal taken at times t_i. Without loss of generality let $0 = t_1 < \ldots < t_m = 1$. A time series may be viewed as a function $\tau : [0,1] \to \mathbb{R}$ by linearly interpolating $w_1, \ldots, w_m$ in order of t_i, $i = 1, \ldots, m$. We obtain a polygonal curve τ with *vertices* $w_1 = \tau(t_1), \ldots, w_m = \tau(t_m)$, we simply refer to it as a *curve*. We say that such a curve τ has *complexity* m.

Clustering is an important tool to analyze the curves. For clustering in general metric spaces the k-center and the k-median problems have been extensively studied. For these problems the best known approximation ratio and the approximation lower bound for any polynomial time algorithm is 2 for the k-center problem [3], and 2.733 and 1.736 respectively for the k-median problem [6, 8].

We want to study the following clustering problems under the Fréchet distance: let Δ_m be the set of all univariate curves of complexity at most m. Given a set of n curves $P = \tau_1, \ldots, \tau_n \subseteq \Delta_m$ and parameters $k, \ell \in \mathbb{N}$, we define a (k,ℓ)-*clustering* as a set of k curves $C = c_1, \ldots, c_k$ taken from Δ_ℓ which minimize one of the following cost functions:

$$\mathrm{cost}_\infty(P, C) = \max_{i=1,\ldots n} \min_{j=1,\ldots k} d_F(\tau_i, c_j)\,, \qquad (1)$$

$$\mathrm{cost}_1(P, C) = \sum_{i=1,\ldots n} \min_{j=1,\ldots k} d_F(\tau_i, c_j)\,, \qquad (2)$$

where d_F is the Fréchet distance.

We refer to these clustering problems as (k,ℓ)-*center* and (k,ℓ)-*median* respectively. We prove that these problems are NP-hard and that the (k,ℓ)-center problem is NP-hard to approximate within a factor of 2.

The Fréchet distance is formally defined as follows: let $\mathcal{H}$ be the set of continuous and increasing functions $f : [0,1] \to [0,1]$ with the property that $f(0) = 0$ and $f(1) = 1$. For two given input curves $\tau : [0,1] \to \mathbb{R}$ and $\pi : [0,1] \to \mathbb{R}$, their *Fréchet distance* is

$$d_F(\tau, \pi) = \inf_{f \in \mathcal{H}} \sup_{t \in [0,1]} \|\tau(f(t)) - \pi(t)\|. \qquad (3)$$

The function f that realizes the Fréchet distance is called a *matching*. The Fréchet distance over curves is a metric (if we observe the space of the equivalence classes of the curves with Fréchet distance 0). The Fréchet distance between two curves of complexity m_1 and m_2 can be computed in $O(m_1 m_2 \log(m_1 m_2))$ time using the algorithm by Alt and Godau [2].

The only known algorithm to compute 1-center clustering (minimum enclosing ball) under the (weak) Fréchet distance is given by Har-Peled and Raichel [4], whose running time is exponential in the number of input curves. Indyk [5] studied the nearest neighbor problem via embeddings of the (discrete) Fréchet distance.

2 Main technique

A simplification of a curve is a curve which has lower complexity than the original curve and which is similar to the original curve. The δ-signature is a special type of curve simplification, such that its Fréchet distance to the original curve is at most δ. It captures the critical points of the original curve, while reducing its complexity. Figure 1 shows an example of such δ-signature defined by the subset of time stamps s_i, $1 \le i \le \ell$.

We want to find low-complexity cluster centers that well describe the input set. We show that the signatures always exist and have a strict hierarchical structure, that enables us to calculate them efficiently, both

[*]Department of Mathematics and Computer Science; TU Eindhoven; the Netherlands

[†]Department of Computer Science; TU Dortmund; Germany

[‡]Department of Computer Science; TU Dortmund; Germany.

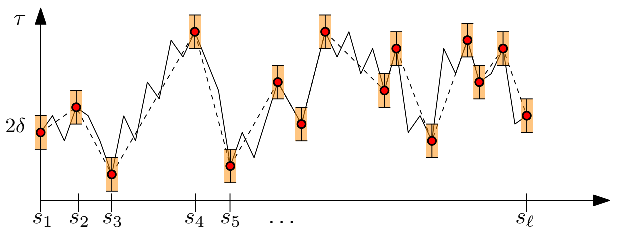


Figure 1: Example of a δ-signature of a curve

in the case when the error δ or goal complexity ℓ is given (in times $O(m)$ and $O(m \log m)$ respectively), where m is the complexity of the input curve.

Two important properties of δ-signatures are given in the following lemma. It enables us to ignore the vertices of the curves that are far from the signature vertices.

Lemma 1 *Let $\sigma = (v_1, \dots, v_\ell)$ be a δ-signature of $\tau = (w_1, \dots, w_m)$. Let $r_j = [v_j - \delta, v_j + \delta]$ be ranges centered at the vertices of σ ordered along σ, where $r_1 = [v_1 - 4\delta, v_1 + 4\delta]$ and $r_\ell = [v_\ell - 4\delta, v_\ell + 4\delta]$. Let π be a curve with $d_F(\tau, \pi) \le \delta$. Then*

(i) π has a vertex in each range r_j, such that these vertices appear on π in the order of j;

(ii) if π' is a curve obtained by removing some vertex u from π with $u \notin \bigcup_{1 \le j \le \ell} r_j$, then it holds that $d_F(\tau, \pi') \le \delta$.

3 Main results

The (k, ℓ)-center clustering algorithm works as follows. We start by computing a constant factor approximation. Using the approximated cost value δ as the range size and the vertices of the δ-signatures of the input curves as the centers of ranges, we can apply Lemma 1(ii) to limit ourselves only to these ranges in the further search for a solution. We compute the union $\mathcal{U}$ of the ranges. We cover $\mathcal{U} \subseteq \mathbb{R}$ with a grid and obtain a candidate set $\mathcal{V}$ for vertices of cluster centers, such that (roughly) $\forall x \in \mathcal{U}, \exists y \in \mathcal{V} : |x - y| \le \varepsilon\delta$. The size of this candidate set can be bounded using Lemma 1(i) and the fact that an optimal solution can use at most $k\ell$ vertices. Namely, the size does not depend on m and n. We form the candidate center curves by taking all possible ℓ-tuples from this set. We evaluate all solutions and find a $(1+\varepsilon)$-approximation.

The result on (k, ℓ)-median clustering uses the algorithm of Kumar *et al.* [7] and the analysis of Ackermann *et al.* [1]. Their result could be directly applied if the doubling dimension of the metric space (Δ_m, d_F) would be bounded. However, the doubling dimension is unbounded, even if m is bounded. Instead, we prove that the *modified sampling property* (Lemma 2) holds for the Fréchet distance and our problem definition (2).

By the analysis of Ackermann *et al.*, the modified sampling property implies a near-linear time algorithm for the (k, ℓ)-median problem in our setting.

Lemma 2 *There exist integer constants $m_{\varepsilon,\lambda,\ell}$ and $t_{\varepsilon,\lambda,\ell}$ such that given a set of curves $P = \{\tau_1, \dots, \tau_n\}$ for a uniform sample multiset $S \subseteq P$ of size $m_{\varepsilon,\lambda,\ell}$ we can generate a candidate set $\Gamma(S) \subset \Delta_\ell$ of size $t_{\varepsilon,\lambda,\ell}$ satisfying*

$$\Pr\left[\exists q \in \Gamma(S) : \mathrm{cost}_1(P, q) \le (1+\varepsilon)\,\mathrm{opt}^{(1)}_{1,\ell}(P)\right] \ge 1 - \lambda,$$

where $\mathrm{opt}^{(1)}_{1,\ell}(P)$ is the cost of the optimal $(1, \ell)$-median clustering of P. Furthermore, we can compute $\Gamma(S)$ in time depending on ℓ, λ and ε only.

Theorem 3 *Let $\varepsilon, \lambda > 0$ and $k, \ell \in \mathbb{N}$ be given. For a given set of curves $P = \{\tau_1, \dots, \tau_n\} \subset \Delta_m$, we can compute a $(1+\varepsilon)$-approximation to the optimal (k, ℓ)-center (resp. (k, ℓ)-median) clustering of P in time*

$$O\left(t_{k,\ell,\varepsilon,\lambda} mn \log m\right),$$

where $t_{k,\ell,\varepsilon,\lambda}$ is a constant depending only on ε, λ, k and ℓ. The result on (k, ℓ)-median clustering is obtained with constant probability.

References

[1] M. R. Ackermann, J. Blömer, and C. Sohler. Clustering for metric and nonmetric distance measures. *ACM Trans. Algorithms*, 6(4):59:1–59:26, 2010.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *IJCGA*, 5:75–91, 1995.

[3] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(0):293 – 306, 1985.

[4] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014.

[5] P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *SOCG*, pages 102–106, 2002.

[6] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *STOC*, pages 731–740, 2002.

[7] A. Kumar, Y. Sabharwal, and S. Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010.

[8] S. Li and O. Svensson. Approximating k-median via pseudo-approximation. In *STOC*, pages 901–910, 2013.

Experiments on Parallel Polygon Triangulation Using Ear Clipping

Günther Eder* Martin Held* Peter Palfrader*

Abstract

We present an experimental study of different strategies for triangulating polygons in parallel. As usual, we call three consecutive vertices of a polygon an ear if the triangle that is spanned by them is completely inside the polygon. Extensive tests on thousands of sample polygons indicate that about 50% of the vertices of most polygons form ears, which suggests that polygon-triangulation algorithms based on ear-clipping might be well-suited for parallelization. We discuss three different on-core approaches to parallelizing ear clipping and report on our experimental findings. Extensive tests show that the most promising method achieves an average speedup of about 3 on a quad-core processor.

1 Introduction

Three consecutive vertices (v_{i-1}, v_i, v_{i+1}) form an ear of a simple polygon P if $\overline{v_{i-1}, v_{i+1}}$ is a diagonal of P. Cutting along this diagonal removes the vertex v_i, the "base" of the ear, thus reducing the number of vertices of P by one. The basic idea of ear clipping is to iteratively cut off ears until the polygon has shrunk to a triangle. The algorithm's correctness hinges upon Meisters' two-ears theorem which states that every non-trivial simple polygon has at least two non-overlapping ears [3].

A typical ear-clipping algorithm operates in two phases. *Classification:* iterate along P to determine all instances of three consecutive vertices that form an ear of P. These potential ears are stored in a queue. *Clipping:* iteratively pick a candidate ear from the queue and clip it if it is still valid. As an ear (v_i, v_j, v_k) is clipped and stored in a triangle list, its two outer vertices v_i and v_k have to be checked to determine whether they form the bases of new ears now. Every new ear is added to the queue. The process ends for an n-vertex polygon when $n-3$ ears have been clipped.

Held's fast industrial-strength triangulation tool FIST [2] is a polygon triangulation framework based on ear-clipping. While the basic ear-clipping algorithm has an $\mathcal{O}(n^2)$ worst-case complexity, FIST employs multi-level geometric hashing to speed up the computation to near-linear time for almost all inputs.

*Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; supported by Austrian Science Fund (FWF) Grant P25816-N15; {geder,held,palfrader}@cosy.sbg.ac.at.

Surprisingly little work has been done on computing triangulations of simple polygons in parallel. The literature focuses mostly on (Delaunay) triangulations of point sets rather than polygons, see for instance Rong et al. [5] and Xin et al. [7]. In 2013, Qi et al. [4] introduced a primarily GPU-based algorithm to compute constrained Delaunay triangulations.

We analyzed the prevalence of ears in our vast set of test data and find that in most polygons about half the vertices form the bases of ears. If we look only at convex vertices then a vast majority (98 %) of them belong to ears. This suggests that clipping many ears simultaneously is feasible. Our contribution are three algorithms for parallelizing ear-clipping on a multi-core processor, which we implemented and tested extensively.

2 Parallel Ear-Clipping Algorithms

We discuss three algorithms for extending the classic FIST tool such that it can operate in parallel. The algorithms differ in how they split the polygon and its ears such that the subsequent ear clipping can be carried out in parallel.

Divide and Conquer. The basic idea is to split the polygon P into as many independent, equally sized sub-polygons as CPU cores are available. Since it seems costly to determine suitable diagonals that achieve balanced splits, we simply use vertical lines to split the polygon. Using a variant of the Sutherland-Hodgman polygon clipping algorithm [6], we can split a polygon along a line ℓ (which does not pass through a vertex of P) in time $\mathcal{O}(n)$, at a cost of at most $\mathcal{O}(n)$ Steiner points caused by the number of intersections between ℓ and the polygon boundary. In our tests, the number of Steiner points seems to be bounded by $\sqrt{n}$ for almost all but contrived inputs. We then run one (sequential) FIST instance per core to obtain a triangulation of each sub-polygon. Glueing the triangulations of all sub-polygons together yields a triangulation of P, albeit with Steiner points which have to be removed.

So consider a pair of Steiner points s_a, s_b that are consecutive along ℓ: We create a hole H by removing all triangles incident in s_a, s_b. Since s_a, s_b lie in the interiors of edges of the boundary of H and since H is "double-star-shaped", the hole H can be triangulated easily without re-creating triangles incident to s_a or s_b.

Partition and Cut. In this approach, we pick k equally-spaced vertices along the boundary of the polygon and partition the boundary into k chains, one for each thread. (Two adjacent chains then always share one of these selected vertices.) In order to run parallel classification steps, one for each chain, we use a per-thread queue to store potential ears instead of a single global queue. Then, in the clipping phase, each thread processes all ears from its queue. As usual, clipping the ear (v_{i-1}, v_i, v_{i+1}) involves checking whether $v_{i\pm1}$ has become the basis of a new ear. However it is only added to the queue if it is not a vertex shared with a neighboring chain. Since each thread only clips ears in its own chain and care is taken to never remove vertices shared between chains, both the classification and clipping step of each thread can run independently of other threads.

After all k queues are empty and the parallel clipping threads have ended, some part of the original polygon remains untriangulated. This part is then processed using a final, sequential run of FIST which completes the triangulation of the polygon.

Mark and Cut. The key observation of this approach is the following: Every second ear along the polygon's boundary is non-overlapping. Therefore, we can process every second ear and clip it independently from all other ears. The phases used by this approach thus differ slightly from the previous two algorithm: In the *mark phase*, we walk along the polygon boundary and store the index of every second vertex in an array A. In the *cut phase*, which can be run by many threads in parallel, we consider every vertex in A, and if it forms the basis of an ear we clip it immediately.

One thread is tasked with running the mark phase. As soon as it has processed half of the polygon boundary, the remaining threads launch a cut phase on the indices stored in A so far while the first thread continues until it has processed the remainder of the boundary.

Once all threads are finished, the cutting threads are re-launched on the vertices that have since been added to A. The marking thread now revisits what remains after the parallel ear-clipping on the first half of the polygon boundary. This continues until only a small number of ears are found in a cut phase. (In our tests we switched to the sequential FIST once fewer than 20 new triangles were generated in one cutting phase.) We then use one sequential run of FIST on the remaining polygon to finish the triangulation.

3 Experimental Results

We implemented all three parallel variants of FIST as an on-core parallelization by the use of OpenMP/C++. Our test system runs CentOS 6.5 on an 2014 Intel Xeon E5-2667 v3 CPU at 3.20 GHz with 8 cores and 132 GB RAM.

Our implementations were tested on about 20 000 polygons with up to four million vertices per input, consisting of both real-world and synthetic data that exhibits various characteristics. The test data was collected over the past 30 years by Martin Held and includes proprietary CAD/CAM designs, sampled printed-circuit board layouts, geographic maps, sampled spline curves and font outlines, closed fractal and space filling curves, as well as star-shaped and various types of "random" polygons generated by RPG [1].

In our tests, we compare the runtime of our parallel algorithms to the runtime of the sequential FIST tool. With eight cores a speedup of about 3 is observed with both the mark-and-cut and the partition-and-cut variants; cf. Fig. 1. The speedup of the divide-and-conquer approach is slightly lower. Most desktop computers have four cores and in such a setting the mark-and-cut variant produces a speedup of about 2–3. Our implementations are not (yet) tuned and, likely, there is room for further improvement.

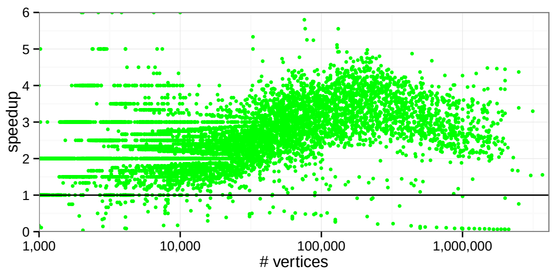


Figure 1: Speed-up obtained by Mark and Cut shown as a function of input size for eight cores.

References

[1] T. Auer and M. Held. Heuristics for the Generation of Random Polygons. In *Proc. 8th Canad. Conf. Comput. Geom (CCCG 1996)*, pages 38–44, Aug. 1996.

[2] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 30(4):563–596, 2001.

[3] G. H. Meisters. Polygons have Ears. *The American Mathematical Monthly*, 82(6):648–651, June 1975.

[4] M. Qi, T. Cao, and T. Tan. Computing 2D Constrained Delaunay Triangulation Using the GPU. *IEEE Trans. Visualizat. Comput. Graph.*, 19(5):736–748, May 2013.

[5] G. Rong, T.-S. Tan, T.-T. Cao, and Stephanus. Computing Two-Dimensional Delaunay Triangulation using Graphics Hardware. In *Proc. ACM Symp. Interactive 3D Graphics*, I3D '08, pages 89–97, 2008.

[6] I. E. Sutherland and G. W. Hodgman. Reentrant Polygon Clipping. *C. ACM*, 17(1):32–42, Jan. 1974.

[7] S.-Q. Xin, X. Wang, J. Xia, W. Mueller-Wittig, G.-J. Wang, and Y. He. Parallel Computing 2D Voronoi Diagrams using Untransformed Sweepcircles. *Computer-Aided Design*, 45(2):483–493, 2013.

The Offset Filtration of Convex Objects [*]

Dan Halperin[†1], Michael Kerber[‡2], and Doron Shaharabani[§1]

[1]Tel Aviv University, Tel Aviv, Israel
[2]Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

We consider offsets of a union of convex objects. We aim for a filtration, a sequence of nested cell complexes, that captures the topological evolution of the offsets for increasing radii. We describe methods to compute a filtration based on the Voronoi diagram of the given convex objects. We prove that, in two and three dimensions, the size of the filtration is proportional to the size of the Voronoi diagram. Our algorithm runs in $\Theta(n \log n)$ time in $\mathbb{R}^2$ and in expected time $O(n^{3+\epsilon})$, for any $\epsilon > 0$, in $\mathbb{R}^3$. Our approach is inspired by alpha-complexes for point sets, but requires a more involved analysis primarily since Voronoi regions of general convex objects do not form a good cover. We show by experiments that our approach results in a similarly fast and topologically more stable method for computing a filtration compared to approximating the input by point samples.

1 Introduction

The theory of *persistent homology* [2] has led to a new way of understanding data through its topological properties, commonly referred as *topological data analysis.* In general, the theory studies the sublevel sets of functions over an arbitrary space. One of the most common specializations for geometric inputs is to consider offsets ("thickenings") of the input shapes with increasing offset parameter and to study the changes in the hole structure during this process. We refer to this sequence of increasing shapes as the *offset filtration.* For any dimension p, persistent homology provides p*-barcode* which constitutes a summary of the p-th homology of the offset shape for any offset parameter value. For instance, the 0-barcode captures the evolution of connected components; Figure 1 describes an example of a 1-barcode.

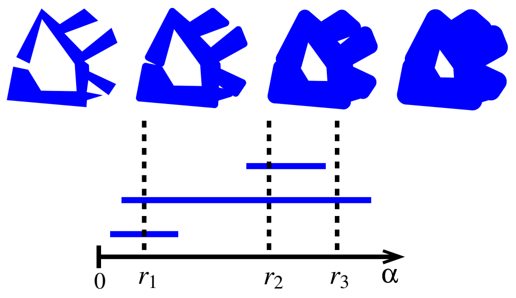


Figure 1: An example shape, its offsets for increasing radii $r_1 < r_2 < r_3$ (top row), and its 1-barcode (bottom). While the shape is simply-connected initially, two holes have been formed at radius r_1, one of which disappears for a slightly larger offset value while the other one *persists* for a large range of scales. At r_2, we see the formation of another rather short-lived hole. The 1-barcode summarizes these facts by displaying one bar per (1-dimensional) hole, spanning over the range of offset radii for which the hole is present.

In computational setups, it is common to assume point sets as input set. We pose the question of how to generalize this default framework to the case where the input consists of convex polytopes. While there is no theoretical obstacle to consider distance functions from shapes rather than points (at least for reasonably "nice" shapes), it raises the question: How can the topological information be represented in a combinatorial structure of small size?

We design, analyze, implement, and experimentally evaluate algorithms for computing persistence barcodes of convex polytopes in 2- and 3- dimensions. More precisely, we concentrate on the problem of computing a filtration, a sequence of nested combinatorial cell complexes that undergoes the same topological changes as the offset shapes. Due to lack of space, proofs and certain details are omitted. We refer the reader to the full manuscript [3] for the complete description.

2 Barcodes of shapes in 2- and 3- dimensions

For a set of objects, the *nerve* describes their intersection patterns: it is a simplicial complex with one vertex per object, where k vertices form a simplex if and

[*]Work by D.H. and D.S. has been supported in part by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University. M.K. acknowledges support by the Max Planck Center of Visual Computing and Communication.

[†]danha@post.tau.ac.il
[‡]mkerber@mpi-inf.mpg.de
[§]doron.s@hotmail.com

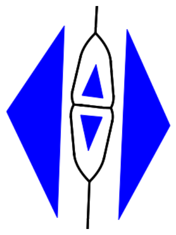
only if the k objects have a common intersection. For instance, the nerve of a collection of balls with same radius is called the *Čech complex*. A set of objects forms a *good cover* if the intersection of any subset is empty or contractible. The *nerve theorem* asserts that for good covers, the union of the objects is homotopically equivalent to their nerve.

Consider a set of convex polytopes and their offset filtration as in Figure 1. The *nerve filtration* is obtained by taking the nerves of the offset polytopes for every offset parameter. While that parameter increases, we obtain an increasing sequence of simplicial complexes which only changes for finitely many offset values. Since offsets of convex polytopes are convex, they form a good cover for every offset parameter, and (an extension of) the nerve theorem implies that offset filtration and nerve filtration yield the same barcode.

The major disadvantage of the construction above is the size of $\Theta(n^{d+1})$ for a nerve filtration of polytopes in d dimensions. A *restricted offset* is the offset of a polytope intersected with its *Voronoi cell*, that is, the region of the space that is closest to the polytope. We note that the union of restricted offsets and the union of non-restricted offsets are equal, which implies that their filtrations are the same. The *restricted nerve filtration* is the filtration of nerves of restricted offsets. Since restricted offsets intersect only at Voronoi cells, the restricted nerve filtration is at most as large as the size of the Voronoi diagram of the input polytopes.

Unlike for point sets, restricted offsets of convex polytopes are not convex nor do they form a good cover. As a result, the nerve theorem does not hold, and the standard proof for barcode equivalence between restricted offset and restricted nerve filtration fails. Our first (surprising) result is that for $d = 2$, the barcodes are equal nevertheless. We state this in the following theorem, which implies that we can obtain a filtration of size $O(n)$ for the 2-dimensional case.

Theorem 1 *For convex polygonal sites in $\mathbb{R}^2$, the 0- and 1-barcode of the restricted nerve filtration are equal to the 0- and 1-barcode of the offset filtration.*



Theorem 1 does not generalize to the 2-barcode: on the left, we see four sites in $\mathbb{R}^2$ where every triple of Voronoi regions intersects, but all four of them do not. Therefore, the nerve consists of the four boundary triangles of a tetrahedron and therefore carries non-trivial 2-homology. In the planar case, the offset filtration can clearly not form any void (a 3-dimensional hole) and we can therefore safely ignore all such features. In $\mathbb{R}^3$, however, the 2-barcode carries information about the offset and such features need to be distinguished from real ones.

Our second result yields an efficient filtration for the case of convex polyhedra in $\mathbb{R}^3$. We build a cell complex with a sweep-line approach that cuts the bi- and trisectors of the Voronoi diagram of the polyhedra. The details of the cutting are determined by analyzing the critical points of the distance function restricted to a Voronoi cell. We obtain the following result.

Theorem 2 *A filtration that has the same barcode as that of the offset filtration of convex, disjoint polyhedra in general position can be computed in time $O(\pi(n)\log(\pi(n)))$, excluding the computation time of the Voronoi diagram, and is of size $O(\pi(n))$, where $\pi(n)$ is the size of the Voronoi diagram.*

$\pi(n)$ is bounded from below by $\Omega(n^2)$ (by a straightforward construction) and from above by $O(n^{3+\epsilon})$ [1, 5], so that the obtained filtration is significantly smaller than the unrestricted nerve filtration of size $O(n^4)$.

3 Experimental evaluation

Computing Voronoi diagrams of polyhedra in space is a difficult problem. While implementation efforts are underway, so far only restricted cases have been completed; see e.g., [4]. We therefore restrict our experiments to the planar case. Already in the plane the restricted nerve approach is much faster than the unrestricted nerve approach which has a cubic running time complexity. In addition, we compare our method with approximating the shapes by point samples. We show that our method is comparably fast, and has the advantage of obtaining an exact barcode while the approximation yields a barcode that contains noise that is hard to distinguish from real small features. For the complete results see [3].

References

[1] P. Agarwal, B. Aronov, and M. Sharir. Computing envelopes in four dimensions with applications. *SIAM J. Comput*, 26:348–358, 1997.

[2] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Disc. Comput. Geom.*, 28(4):511–533, 2002.

[3] D. Halperin, M. Kerber, and D. Shaharabani. The offset filtration of convex objects. *CoRR*, abs/1407.6132, 2014.

[4] M. Hemmer, O. Setter, and D. Halperin. Constructing the exact Voronoi diagram of arbitrary lines in three-dimensional space - with fast point-location. In *Eur. Symp. on Alg.*, pages 398–409, 2010.

[5] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Disc. Comput. Geom.*, 12(1):327–345, 1994.

Generalized Offsetting Using a Variable-Radius Voronoi Diagram

Martin Held* Stefan Huber† Peter Palfrader*

Abstract

We investigate ways to extend offsetting based on skeletal structures beyond the well-known constant-radius and mitered offsets supported by Voronoi diagrams and straight skeletons for which the orthogonal distance of offset elements to their input elements is uniform. We introduce a new geometric structure called the *variable-radius Voronoi diagram*, which supports the computation of variable-radius offsets, i.e., offsets whose distance to the input may vary along the input.

1 Introduction

Offsetting is an important task in diverse applications in the manufacturing business. For a set C in the Euclidean plane, the constant-radius offset with offset distance r is the set of all points of the plane whose minimum distance from C is exactly r. Formally, this offset curve can be defined as the boundary of the set $\bigcup_{p\in C} B(p,r)$, where $B(p,r)$ denotes a disk with radius r centered at the point p.

For polygons such an offset curve will consist of one or more closed curves made up of line segments and circular arcs. Held [2] describes as algorithm using the Voronoi diagram to compute such an offset efficiently and reliably. Mitered offsets differ from constant-radius offsets in the handling of non-convex vertices of an input polygon: Instead of adding circular arcs to the offset curve, the offset segments of the two edges incident to a non-convex vertex get extended until they intersect. This type of offset can be generated in linear time from the straight skeleton [3]; see Figure 1. A common feature of these offsets is that the orthogonal distance of each offset element from its defining contour element is constant.

Several applications in industry, such as for garment manufacture or shoe design, need to construct differently sized pieces from a single master design. The obvious method of scaling is not always desirable as it scales all elements equally. An alternative is to use offsetting, and a common practical requirement is creating non-constant offsets, i.e., offset curves where the distance of each point of the offset to the original input curve changes along the input.

*Universität Salzburg, FB Computerwissenschaften, Austria; supported by Austrian Science Fund (FWF): P25816-N15.

†Institute of Science and Technology, Austria

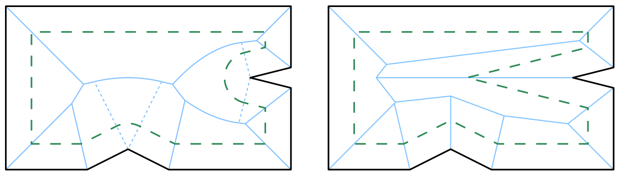


Figure 1: The Voronoi-diagram (left) and the straight skeleton (right) of a polygon enable efficient computation of constant-radius and mitered offsets (dashed).

Prior work on variable-distance offsets [4, 5, 6] seems to concentrate on defining and comparing different offsets and is less concerned with robustly computing offset curves.

2 Main Idea

Consider a planar straight-line graph S in the plane. Let us denote by $\overline{S} \subset \mathbb{R}^2$ the set of points covered by all vertices and line segments of S. Furthermore, we consider a weight function $\sigma\colon \overline{S} \to \mathbb{R}_+$ that assigns to each vertex p of S a positive weight $\sigma(p)$ and for each point on a line segment $\overline{pq}$ of S we linearly interpolate its weight along $\overline{pq}$ from $\sigma(p)$ at p to $\sigma(q)$ at q.

We now place a disk at each point p of $\overline{S}$. In analogy to the so-called prairie fire model, all disks have initially radius zero. As time increases, however, the radius of each disk grows proportional to the weight $\sigma(p)$ of its center point $p \in \overline{S}$. The *variable-radius offset* for a given time is the envelope of this set of disks. As intended, input sites with small weight will induce an offset that is closer to them, and input sites that were assigned larger weights will cause their offsets to be farther away. Formally, this offset is the boundary of the set $\bigcup_{p\in\overline{S}} B(p,\sigma(p)\cdot t)$. Note that the term $\sigma(p)\cdot t$ replaces the constant radius r of standard offsets.

Having used skeletal structures such as the Voronoi diagram and the straight skeleton to construct constant-radius and mitered offsets in the past, we are looking for another Voronoi-like structure to facilitate the computation of non-constant offsets. The variable-radius Voronoi diagram introduced below and defined relative to weighted points and variably-weighted line segments is such a useful structure.

Preliminaries. The Voronoi diagram $\mathcal{VD}(S)$ of a set S of points in the plane, called *sites*, tessellates the plane into interior-disjoint *regions*. Each Voronoi region belongs to exactly one site s and is the locus of all points in the plane whose closest site is s.

We introduce the *variable-radius Voronoi diagram* $\mathcal{VD}_v(S)$ as a generalized Voronoi diagram with generalizations into two directions: First, the set S of input sites is a planar straight-line graph, i.e., a set of both vertices and non-intersecting line segments between pairs of these vertices. Second, we assign multiplicative weights to these sites. As described above, vertices $s \in S$ are assigned positive weight $\sigma(s)$, and the weight of a point on a line segment $\overline{pq}$ changes linearly between its endpoints from $\sigma(p)$ to $\sigma(q)$.

The distance of a point u in the plane to a vertex site s is defined as the Euclidean distance from u to s, divided by the weight of that site: $d(u,s) := \frac{\|u-s\|}{\sigma(s)}$. The distance of u to a line-segment site $\overline{pq}$ is naturally defined as the minimum distance of u to any point of the line segment: $d(u,\overline{pq}) := \min_{v\in\overline{pq}} \frac{\|u-v\|}{\sigma(v)}$.

As in the case of the standard Voronoi diagram, every point in the plane is in the (generalized) Voronoi region of the site that it is closest to. An arc that separates two regions comprises all points that have the same distance to two sites and a larger distance to all other sites.

The variable-radius Voronoi diagram inherits several important properties from the multiplicatively weighted Voronoi diagram of points. For instance, the region of a given site need not be connected and bisectors between two vertices are circles or circular arcs [1]. Other bisectors, however, are more complex curves in general.

The bisectors between a line segment $\overline{pq}$ and its two incident vertices p and q exhibit an interesting property: We can show that they are full circles whose diameters on the line supporting $\overline{pq}$ are bounded by a common point on one side and p and q, respectively, on the other; see Figure 2, left.

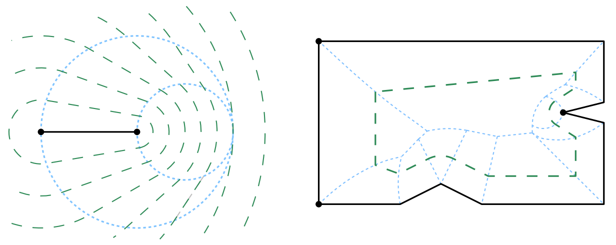


Figure 2: (Left) The variable-radius Voronoi diagram (blue, dotted) of a line segment and its two incident vertices. A family of offset curves is shown in green and dashed. (Right) The variable-radius Voronoi diagram inside a polygonal input with weighted vertices.

Offsetting. While the bisectors of $\mathcal{VD}_v(S)$ consist also of non-trivial curves, it can be shown that the variable-radius offset itself comprises line segments and circular arcs only; see Figure 2, right.

We can compute the variable-radius offset of S for a given time t from the variable-radius Voronoi diagram $\mathcal{VD}_v(S)$. The approach is identical to how constant-distance offsets are computed based on Voronoi diagrams or straight skeletons [2, 3]. Roughly, we iterate through all the arcs of $\mathcal{VD}_v(S)$ and add offset elements in each face that contains points at distance $t \cdot \sigma$. The topological information encoded in $\mathcal{VD}_v(S)$ enables us to do this in time linear in the size of the Voronoi diagram and in a single iteration, without the need to compute all pair-wise self-intersections of offsets.

Construction. Similarly to the standard Voronoi diagram, the variable-radius Voronoi diagram can be obtained from the lower envelope of surfaces in 3D. For vertices, the corresponding surface in 3D is a cone whose dihedral angle depends on the weight of the input vertex. Input line segment induce ruled surfaces in 3D as the offsets of line segments are also line segments. In particular, the surfaces will be subsets of right conoids.

CGAL's 3D envelope computation algorithm is generic in the sense that it can deal with arbitrary terrain surfaces so long as it has some means to learn about certain geometric properties. We are working on proof-of-concept code, based on CGAL, to compute the variable-radius Voronoi diagram of planar straight-line graphs and to compute variable-radius offsets.

3 Conclusion

We investigate one specific variant of a skeletal structure which we call the variable-radius Voronoi diagram. While this structure is of particular interest in itself, we demonstrate its applicability to robustly constructing variable-radius offsets.

An open problem is to generalize the class of input sites to include for instance circular arcs. We hope that this would enable offsets that are $\mathcal{G}^1$ continuous for some class of inputs. However, note that the offset of a variable-weighted circular arc is not a circular arc. Hence, a better understanding of the mathematical characteristics of the resulting offsets and of the corresponding Voronoi bisectors is required.

References

[1] F. Aurenhammer and H. Edelsbrunner. An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane. *Pattern Recognition*, 17(2):251–257, 1984.

[2] M. Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *LNCS*. Springer, 1991. ISBN 978-3-540-54103-5.

[3] P. Palfrader and M. Held. Computing Mitered Offset Curves Based on Straight Skeletons. *Comp.-Aided Design & Appl.*, 12(4):414–424, Feb. 2015.

[4] L. Qun and J. G. Rokne. Variable-Radius Offset Curves and Surfaces. *Mathl. Comput. Modelling*, 26(7):97–108, Oct. 1997.

[5] J. Rossignac. Ball-Based Shape Processing. In *Proc. 16th Int. Conf. Discrete Geom. Comp. Imagery (DCGI 2011)*, volume 6607 of *LNCS*, pages 13–34, Nancy, France, Apr. 2011. Springer.

[6] W. Zhuo and J. Rossignac. Curvature-based Offset Distance: Implementation and Applications. *Computers & Graphics*, 36(5):445–454, Aug. 2012.

Drawing Planar Cubic 3-Connected Graphs with Minimal Visual Complexity[*]

Alexander Igamberdiev[†] Wouter Meulemans[†] André Schulz[†]

1 Introduction

To assess the quality of network visualizations, many criteria have been investigated, such as crossing minimization, bend minimization and angular resolution (see [6] for an overview). The structural complexity of a graph is often measured in terms of its number of vertices or edges. This, however, does not necessarily correspond to its *cognitive load* (mental effort needed to interpret a drawing). Bends and crossings increase the cognitive load, making it harder to interpret a graph visualization, and should be avoided.

We consider the following measure of *visual complexity* for planar graphs [5]: the number of basic geometric objects that are needed to realize the drawing. For example, if a path in the graph is placed along a line, then we do not need one line segment for each edge in this path; one line segment can represent the entire path.

Upper and lower bounds on the necessary visual complexity of various graph classes are known [5]. A lower bound for any graph is $\lceil n/2 \rceil$, where n is the number of odd-degree vertices: at least one geometric object must have its endpoint at such a vertex.

We consider line-segment drawings for *planar cubic* 3-*connected graphs*; unless mentioned otherwise, graph is used to refer to a graph of this class. Any plane drawing has at least three vertices of the same face on its convex hull: such a vertex is the endpoint of the line segment for each incident edge. Thus, we obtain a lower bound of $n/2 + 3$ line segments, as n is even. An algorithm is known which uses $n + 2$ line segments [2].[1]

To compute a plane drawing matching the lower bound, we are given (or pick) three convex hull vertices; these are referred to as the *suspension vertices*. For all other *internal vertices*, we decide which two incident edges lie on the same line segment, that is, which of the three angles is flat. Hence, this corresponds to a *flat-angle assignment*; we refer to plane drawings that match the lower bound as *flat-angle drawings*. Note that any face in a flat-angle drawing is nonstrictly convex.

Aerts and Felsner [1] describe conditions for the stretchability of flat-angle assignments. From a stretchable assignment, a layout can be obtained by solving a system of harmonic (linear) equations with arbitrary edge weights. How to efficiently compute stretchable flat-angle assignments remains an open problem.

Contributions. We present the high-level ideas for two different $O(n^2)$-time algorithms (Sect. 2 and 3) to construct a plane drawing matching the $n/2+3$ lower bound for $n \geq 6$. The algorithms produce drawings directly. From a drawing, a flat-angle assignment can be derived and thus they can also be used to set up a system of harmonic equations [1]. We postprocess any drawing by solving the set of equations using uniform edge weights.

2 Deconstruction algorithm

For the deconstruction algorithm, we define an operation called *edge insertion*[2]: pick two edges that belong to one face, subdivide both edges and add a new edge between the new degree-2 vertices. It is folklore that every cubic 3-connected graph can be obtained from K_4 by a sequence of edge insertions (e.g, [3], page 243). We prove a slightly stronger version: any graph other than K_4 can be constructed from the triangular prism, while not adding new edges in a given outer face (though outer-face edges may be subdivided).

We compute a construction order by repeatedly applying the edge insertion's inverse operation to the desired graph, maintaining planarity, 3-regularity, 3-connectivity and a chosen outer face. This procedure always finishes on a triangular prism which has a trivial flat-angle drawing with any outer face. With this construction order, we reconstruct the desired graph from

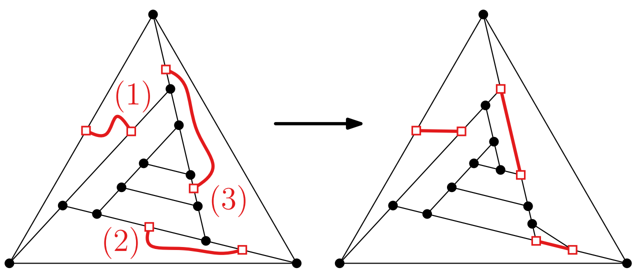


Figure 1: Edge insertion that connects (1) noncollinear edges of a face, (2) collinear edges separated by exactly one vertex, and (3) collinear edges separated by two or more vertices. In case 2 and 3, we reassign the flat angle at the first and/or last separating vertex.

[*]This work was supported by DFG grant SCHU 2458-4/1.

[†]`[alex.igamberdiev|w.meulemans|andre.schulz]@uni-muenster.de`, Westfälische Wilhelms-Universität Münster

[1]Another algorithm matching this lower bound is described [4], but details are insufficient to verify its correctness.

[2]This is *not* the graph-theoretic notion of edge insertion.

the triangular prism, with a sequence of edge insertions maintaining a flat-angle drawing (see Fig. 1).

3 Windmill algorithm

The windmill algorithm computes a flat-angle drawing, working its way inward from the outer face. It does so recursively, using as parameter a simple cycle C in the graph. It assumes that C is drawn as a nonstrictly convex polygon. Its convex corners correspond to suspension vertices or vertices having an edge outside C; any flat vertex has an edge inside C. Initially, C is the outer face, drawn as an equilateral triangle with the suspension vertices as corners (see Fig. 2(a)). A recursive step for a cycle C is done using one of four cases, based on the cyclic sequence F of faces along the inside of C.

1. If at most one vertex lies inside C, we draw all chords as line segments. The one vertex (if present) is positioned to lie on a line segment between two of its neighbors. See Fig. 2(b-c,e-f).
2. If a face occurs more than once in F, we draw its paths inside C as line segments and recurse on a subcycle for each path. See Fig. 2(c-d).
3. If two faces share an edge, but are not consecutive in F, we draw three line segments to represent the paths inside C along the two faces and recurse on the two subcycles created. See Fig. 2(a-b).
4. Otherwise, we create a windmill pattern with the sequence of faces along C. We recurse on the cycle inside the windmill. See Fig. 2(d-e).

There is a subtlety for case 3: the faces must lie on different sides of the polygon for C. Otherwise, case 4 handles this pattern using additional recursive calls.

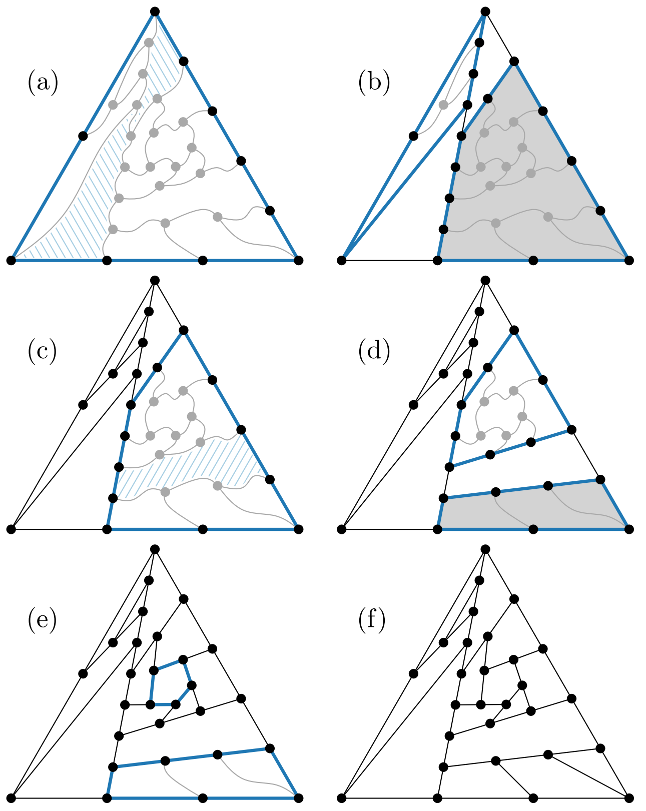


Figure 2: The windmill algorithm. Cycles are drawn thick; unshaded cycles are processed in the next step. (a) Initial call. (b-e) Consecutive states. (f) Final result. Two cycles are processed between (e) and (f). (a,c) Hashures indicate faces revelant for case 2 and 3.

4 Outlook

We presented two algorithms to compute graph layouts for planar cubic 3-connected graphs, that achieve a visual complexity matching the lower bound. The algorithms typically produce different layouts (Fig. 3). As the visual complexity is the same, we need other criteria to further assess the overall quality. We intend to investigate which algorithm performs better in such other criteria, or design another algorithm to explicitly incorporate these. Moreover, user studies are needed to investigate whether this definition of visual complexity correlates to an observer's cognitive load.

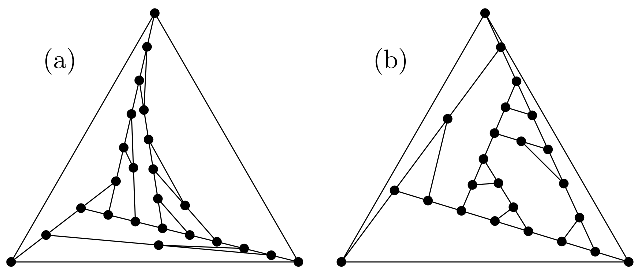


Figure 3: Different layouts for the same graph using uniform weights in the harmonic equations. (a) Deconstruction algorithm. (b) Windmill algorithm.

References

[1] N. Aerts and S. Felsner. Straight line triangle representations. In *Graph Drawing 2013*, pages 119–130, 2013.

[2] V. Dujmović, D. Eppstein, M. Suderman, and D. Wood. Drawings of planar graphs with few slopes and segments. *CGTA*, 38:194–212, 2007.

[3] B. Grünbaum. *Convex Polytopes*. Graduate Texts in Math. Springer-Verlag, New York, 2nd edition, 2003.

[4] D. Mondal, R. Nishat, S. Biswas, and S. Rahman. Minimum-segment convex drawings of 3-connected cubic plane graphs. *J. of Comb. Opt.*, 25:460–480, 2013.

[5] A. Schulz. Drawing graphs with few arcs. In *Graph-Theoretic Concepts in CS*, pages 406–417, 2013.

[6] R. Tamassia. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.

Randomized Incremental Construction for the Hausdorff Voronoi Diagram[*]

Elena Khramtcova[†] Evanthia Papadopoulou[†]

Abstract

We apply the randomized incremental construction framework to the Hausdorff Voronoi diagram of a family of k clusters of total n points, and obtain an algorithm of expected $O((m + n\log k)\log n)$ time and $O(m + n\log k)$ space, where m is the number of crossings between pairs of crossing clusters ($m = O(n^2)$); the complexity of the diagram is $O(n+m)$. For non-crossing clusters ($m = 0$), the algorithm simplifies to expected $O(n\log n + k\log n\log k)$ time and $O(n)$ space.

1 Introduction

The *Hausdorff Voronoi diagram* of a family F of point clusters in $\mathbb{R}^2$ (HVD(F), for brevity HVD) is a subdivision of $\mathbb{R}^2$ into maximal regions such that every point within one region has the same nearest cluster, where the distance from a point t to a cluster P is the maximum distance from t to any point in P. See Fig. 1(a).

We assume that no cluster contains another one inside its convex hull, as such clusters would have no effect to the diagram. We call two clusters *non-crossing*, if their convex hulls admit two supporting segments, and *crossing* otherwise (see Fig. 1(b)). The number of supporting segments (divided by 2 minus 1) reflects the *number of crossings* between the two clusters. Let k be the number of clusters in F, and n be the total number of points in all clusters; let m be the total number of crossings between pairs of crossing clusters ($m = O(n^2)$).

The combinatorial complexity of the HVD is $O(n + m)$ [5, 7], and this is tight. The diagram can be computed in time $O(n^2)$ [5], but faster algorithms exist for non-crossing clusters where the diagram has complexity $O(n)$, see [3] and references therein. For non-crossing clusters, we recently presented a randomized incremental construction (RIC) in expected $O(n\log n\log k)$ time and $O(n)$ space [3]. The HVD of non-crossing clusters is an instance of *abstract Voronoi diagrams* (AVDs) while the HVD of arbitrary clusters is not.

In this abstract, we apply the standard RIC framework to the construction of the HVD in all cases. The algorithm for the HVD of arbitrary clusters requires $O((m+n\log k)\log n)$ expected time and $O(m+n\log k)$ expected space. The simplified algorithm for non-crossing clusters runs in $O(n\log n + k\log n\log k)$ expected time and $O(n)$ expected space. The latter is a complimentary approach to [3] for the same problem, which avoids the use of point location data structures, and follows the RIC framework.

[*]Research supported in part by the Swiss National Science Foundation, project 20GG21-134355, under the ESF EUROCORES program EuroGIGA/VORONOI.

[†]Università della Svizzera italiana (USI), Lugano, Switzerland

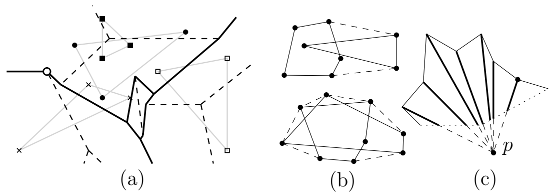


Figure 1: (a) HVD; (b) Non-crossing clusters and clusters with 2 crossings; (c) visibility-based decomposition

2 Overview of RIC and properties of HVD

RIC framework [4]. The RIC paradigm to compute a Voronoi diagram incrementally, inserts sites one by one, each time recomputing the target diagram. The diagram is viewed as a collection of *ranges*, defined and without *conflicts* w.r.t. the set of sites inserted so far. To update the diagram efficiently, a *conflict graph* is maintained. It is a bipartite graph whose nodes correspond to (1) the ranges for the diagram of of the sites inserted so far, and (2) the sites that are not yet inserted. Edges of the graph correspond to conflicts between the respective ranges and objects. The RIC framework has been applied to a number of generalized Voronoi diagrams [2]. However, these diagrams do not include the HVD, which has several features that have not been considered by efficient applications of the RIC paradigm before. These include: (1) sites of non-constant complexity, (2) sites that are not enclosed in their Voronoi regions, and (3) disconnected and empty Voronoi regions. A recent work on AVDs with *disconnected regions* [1] does not apply to the HVD.

HVD. For a cluster $C \in F$, let $\mathrm{hreg}_F(C)$ denote the region of C in HVD(F), and $\mathcal{T}(C)$ denote the graph structure of the *farthest Voronoi diagram* of C.

A *C-mixed vertex* of HVD(F) is the point where a Voronoi edge bounding $\mathrm{hreg}_F(C)$ meets $\mathcal{T}(C)$; it is equidistant to two points of C and one point of another cluster. In Fig. 1(a), a C-mixed vertex of the leftmost cluster is marked by unfilled circle.

Observation 1 ([7]) *Each face of $\mathrm{hreg}_F(C)$ intersects $\mathcal{T}(C)$ in one non-empty connected component. This component is delimited by C-mixed vertices.*

The components are shown dashed in Fig 1(a).

The faces of the HVD may be split into simple subfaces by the *visibility-based* decomposition [7] (while maintaining the $O(n+m)$ complexity of the diagram). In Fig. 1(c), the visibility-based decomposition of (part of) a single face of HVD is shown in bold.

3 The algorithms

We follow the RIC framework. First, we discuss arbitrary clusters, and then non-crossing clusters. The latter case is simpler, as each region of HVD has at most one face.

For both settings we define a *range* to be a face of the visibility-based decomposition of HVD. A *conflict* is defined differently in each case.

RIC for HVD of arbitrary clusters. A conflict is defined as follows. A range τ is *in conflict* with a cluster $C \in F \setminus S$, if $\tau \cap \mathcal{T}(C) \cap \mathrm{hreg}_{S\cup\{C\}}(C) \neq \emptyset$. With each conflict in the conflict graph we store a point in this intersection as a witness and all C-mixed vertices of $\mathrm{HVD}(S \cup \{C\})$ contained in τ.

Suppose that for some $S \subset F$, $\mathrm{HVD}(S)$ and the conflict graph have been computed. To insert a cluster $C \in F \setminus S$, we need to efficiently: (1) compute $\mathrm{HVD}(S \cup \{C\})$; and (2) update the conflict graph. The latter is especially difficult since the update condition on the conflict graph [2] is violated. That is, we need to bound the number of unsuccessful tests of ranges against clusters. Note that one such test does not take constant time as the clusters are of non-constant complexity.

Lemma 1 *(1)* $\mathrm{HVD}(S \cup \{C\})$ *can be computed from* $\mathrm{HVD}(S)$ *in time linear in the number of its structural changes. (2) The conflict graph can be updated in time* $O((A_C + N(C))\log n)$, *where* A_C *is the number of conflicts created and deleted, and* $N(C) = \sum_{Q\in F\setminus S} N_Q$ *where* N_Q *is the number of edges of* $\mathcal{T}(Q)$ *that were conflicting with ranges of* $\mathrm{HVD}(S)$ *and do not conflict with any range of* $\mathrm{HVD}(S \cup \{C\})$.

Proof. *(1).* To obtain $\mathrm{HVD}(S \cup \{C\})$, we trace all faces of $\mathrm{hreg}_{S\cup\{C\}}(C)$ starting from the witness points of conflicts of C. Each face of $\mathrm{hreg}_{S\cup\{C\}}(C)$ has at least one witness of a conflict in it, due to Observation 1 and the fact that one range of $\mathrm{HVD}(S)$ intersects at most one connected component of $\mathcal{T}(C) \cap \mathrm{hreg}_{S\cup\{C\}}(C)$.

(2). For any range τ of $\mathrm{HVD}(S)$, and any cluster $Q \in F \setminus S$, $\tau \cap \mathrm{hreg}_{S\cup\{Q\}}(Q)$ is connected and its boundary inside τ is a convex chain, whose breakpoints are Q-mixed vertices of $\mathrm{HVD}(S\cup\{Q\})$. Thus the new conflicts of Q can be found by *(a)* tracing N_Q edges of $\mathcal{T}(Q)$ to find new Q-mixed vertices; and *(b)* tracing new ranges conflicting with Q. A range τ can be tested against a cluster $Q \in F \setminus S$ by a constant number of segment queries in the *separator decomposition* [3] of $\mathcal{T}(Q)$. □

The total number of Q-mixed vertices in $\mathrm{HVD}(S\cup\{Q\})$ for all clusters $Q \in F$ and all $S \subset F$, is $O(m+n)$. Thus, by Lemma 1 and [2, Theorem 5.2.3], we conclude.

Theorem 2 $\mathrm{HVD}(F)$ *can be computed in* $O((m + n\log k)\log n)$ *expected time and* $O(m + n\log k)$ *expected space.*

RIC for HVD of non-crossing clusters. If all clusters in F are pairwise non-crossing, then all regions of HVD are connected. Thus, we can modify the definition of a conflict. To this aim, for each $C \in F$, let $\mathcal{T}(C)$ be rooted at a point at infinity along an arbitrary unbounded edge. A range τ of $\mathrm{HVD}(S)$ is *in conflict* with a cluster $C \in F\setminus S$ if and only if τ contains the *highest point* of $\mathcal{T}(C) \cap \mathrm{hreg}_{S\cup\{C\}}(C)$ (w.r.t. root of $\mathcal{T}(C)$). This highest point is the witness of the conflict, and a starting point for tracing the region of C, when C is inserted in HVD. Each cluster of $F \setminus S$ has at most one conflict, thus, the conflict graph requires $O(n)$ expected space. One test of a cluster against a range is performed in $O(\log n)$ time (see the *separator decomposition* [3]).

Theorem 3 *If all clusters of* F *are pairwise non-crossing,* $\mathrm{HVD}(F)$ *can be computed in* $O(n\log n + k\log k\log n)$ *expected time and* $O(n)$ *expected space.*

We plan to improve the space complexity of the first algorithm and to investigate an output sensitive variant.

Acknowledgments. We thank Maria Saumell for useful discussions.

References

[1] C. Bohler and R. Klein. Abstract Voronoi diagrams with disconnected regions. In *ISAAC'13*, p. 306–316.

[2] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, 1998.

[3] P. Cheilaris, E. Khramtcova, S. Langerman, and E. Papadopoulou. A randomized incremental approach for the Hausdorff Voronoi diagram of non-crossing clusters. In *11th LATIN'14*, vol. 8392 of *LNCS*, p. 96–107.

[4] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[5] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, 4:311–336, 1989.

[6] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom.*, 3(3):157–184, 1993.

[7] E. Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.

Recognizing Weighted and Seeded Disk Graphs

Boris Klemz* Martin Nöllenburg* Roman Prutkin*

Abstract

A set of disks D in the plane is a *disk intersection representation (DIR)* of a graph $G = (V, E)$ if there is a bijection between V and D such that two disks intersect if and only if the corresponding vertices are adjacent. If the disks are interior-disjoint, D is called *disk contact representation (DCR)*. We show that recognizing graphs realizable as DCRs/DIRs in which the disks' radii coincide with preassigned values and in which disks cover preassigned points, called *seeds*, is NP-hard even if the input graph is a path and even if a uniform radius is assigned to all disks. We also extend a previous reduction and show that the seeded DCR case without preassigned radii is NP-hard even for trees.

1 Introduction

Graphs that have a DIR/DCR are called *disk intersection/contact graphs* (DIG/DCG). Application areas for such disk graphs include modeling physical problems like wireless communication networks, covering problems like geometric facility location, visual representation problems like area cartograms and many more (Clark et al. [4] provide an extensive overview). Often, one is interested in recognizing disk graphs or generating disk representations that do not only realize the input graph, but also satisfy additional requirements. It might be desirable to generate a *weighted* DIR/DCR, that is, a DIR/DCR in which the disks' radii correspond to predefined values. The corresponding recognition problems are NP-hard, even if all vertices are weighted uniformly [2]. One might also be interested in generating a *seeded* DIR/DCR in which each disk is required to cover a point preassigned to its corresponding vertex. Atienza et al. [1] showed that seeded DCG recognition is NP-hard even if G is outerplanar.

We extend the result by Atienza et al. and show that seeded DCG recognition remains NP-hard even for trees. We also combine the two scenarios and show that weighted, seeded DCG and DIG recognition are NP-hard, even if all vertices are weighted uniformly and even if the input graph is a path.

*Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. boris.klemz@student.kit.edu, noellenburg@kit.edu, roman.prutkin@kit.edu

2 Weighted, Seeded Disk Representations

A planar 3SAT (P3SAT) formula φ is a Boolean 3SAT formula with a set $\mathcal{U}$ of variables and a set $\mathcal{C}$ of clauses such that its *variable-clause-graph* $G_\varphi = (\mathcal{U} \cup \mathcal{C}, E_\varphi)$ is planar. The set E_φ contains for each clause $c \in \mathcal{C}$ the edge (c, x) if a literal of variable x occurs in c. Deciding the satisfiability of a P3SAT formula is NP-complete [7] and there exists a planar drawing of G_φ on a grid of polynomial size such that the variable vertices are placed on a horizontal line and the clauses are connected in a comb-shaped rectangular fashion from above or below that line [6]. This drawing can furthermore be slanted such that all angles are multiples of 60 degrees [3]. We call this slanted drawing $\mathcal{G}_\varphi$. In order to show that weighted, seeded DCG recognition is NP-hard, we create a graph $G = (V, E)$ and a seed assignment $s : V \to \mathbb{R}^2$, which encode $\mathcal{G}_\varphi$, such that G can be realized as a uniformly weighted, seeded DCR (USDCR) with respect to s if and only if φ is satisfiable.

If a graph and a seed assignment have an unique USDCR, we call this USDCR and its disks *rigid*. It is easy to construct graphs with rigid USDCRs. Start by placing two touching disks and assign their seeds to the extremal points two unit diameters apart from each other. More disks can be rigidly attached to this construction using a similar approach.

A key idea for our reduction is placing a seed p between two rigid disks such that the disk covering p has exactly two possible locations, see the close-up box in the top part of Fig. 1. Note how the bottom white disk has to be embedded towards the bottom if the top disk is embedded towards the bottom. This idea can be extended to create long *chains* that allow information transfer. Furthermore, if a chain is *closed* it has exactly two possible states, either all disks are embedded clockwise or anti-clockwise, see the gray highlighted section of Fig. 1. This figure depicts a *variable gadget*, one of which is created for each variable vertex of $\mathcal{G}_\varphi$. The two states of the closed chain emulate the truth states of the variable. The *left/right sections* are designed such that adjacent variable gadget can be attached to each other. The required number of *middle sections* depends on the number of incident literal edges. These edges are represented by chains attached to the middle sections such that their disks are pushed away from the gadget if the literal evaluates to false with respect to the gadget state.

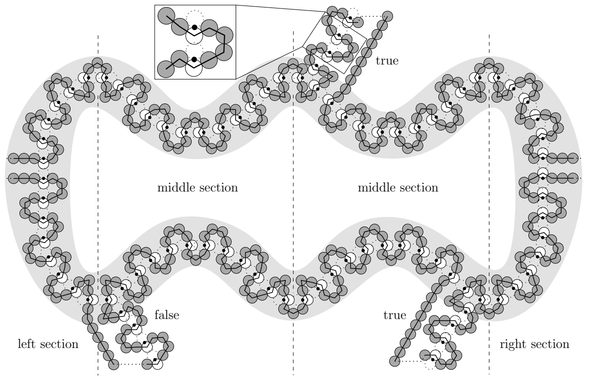


Fig. 1: A variable gadget. Gray disks are rigid.

At *clause gadgets*, three literal chains meet symmetrically at 120° angles, see Fig. 2. These gadgets are designed such that disks of at most two literal chains can be embedded towards it. This is achieved by three *junctions*, see close-up in Fig. 2. A junction pushes disks of at least one of two adjacent chains away if the literal chain connected to the junction is embedded towards it. Recalling that literal chains are pushed towards clauses if they evaluate to false, we can conclude that G can be realized with respect to s if and only if φ is satisfiable. Finally, on the left side of the left-most variable gadget, we add two disks connecting the bottom and the top part so that the constructed graph becomes a path. The size of G is clearly polynomial. Computing precise coordinates for s can take superpolynomial time, however, by shifting the seeds onto a Cartesian grid of appropriate acuteness (which does not depend on the input) we obtain an equivalent instance allowing us to carry out the reduction in polynomial time.

It is straight-forward to adapt our construction for USDIG recognition. We shift seeds to provide a tiny amount of "wiggle room" for disks that had only two possible locations. We also remove each edge $(u, v) \in E$ where u and/or v correspond to a non-rigid disk, and add a vertex w with an appropriately chosen seed to V and edges (u, w) and (w, v) to E. Figure 3 illustrates the information transfer with this adaption. We obtain:

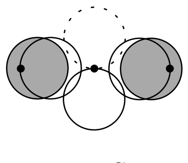


Fig. 3: USDIR.

Theorem 1 *USDCG and USDIG recognition are* NP-*hard, even if the input graph is a path.*

3 Seeded Disk Contact Representations

Atienza et al. [1, Theorem 2.3] show that seeded DCG recognition is NP-hard by creating a graph G and and a seed assignment realizable if and only if a P3SAT φ formula is satisfiable. We note that their graph G is outerplanar and non-connected. The different connected components of G can easily be connected by adding paths between literal chains and variable gadgets and by slightly shifting all singleton seeds and connecting them with edges to their respective chains. At both ends of each literal chain we find a cycle. Each of these cycles contains exactly one edge connecting two degree-2 vertices. We remove these edges and observe that our graph is now a tree which is still realiziable if and only if φ is satisfiable. For more details we refer to [5].

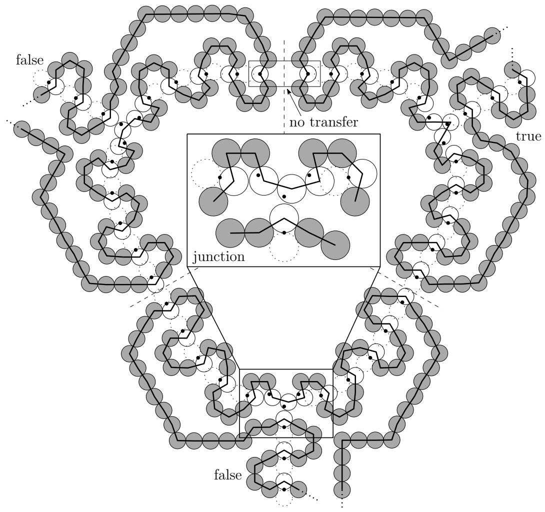


Fig. 2: A clause gadget.

Theorem 2 *Seeded DCG recognition is* NP-*hard even if the input graph is a tree.*

References

[1] N. Atienza, N. de Castro, C. Cortés, M. Á. Garrido, C. I. Grima, G. Hernández, A. Márquez, A. Moreno-González, M. Nöllenburg, J. R. Portillo, P. Reyes, J. Valenzuela, M. T. Villar, and A. Wolff. Cover Contact Graphs. *Journal of Computational Geometry*, 3(1):102–131, 2012.

[2] H. Breu and D. G. Kirkpatrick. Unit Disk Graph Recognition is $\mathcal{NP}$-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

[3] C. Cabello, E. D. Demaine, and G. Rote. Planar Embeddings of Graphs with Specified Edge Lengths. *Journal of Graph Algorithms and Applications*, 11(1):259–276, 2007.

[4] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit Disk Graphs. *Disc. Mathematics*, 86(1–3):165–177, 1990.

[5] B. Klemz. Weighted Disk Contact Graphs. Diploma thesis, Karlsruhe Institute of Technology, 2014.

[6] D. E. Knuth and A. Raghunathan. The Problem of Compatible Representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992.

[7] D. Lichtenstein. Planar Formulae and their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

Diamonds are a Quiver's Best Friend[*]

Clément Maria[†]

Abstract

Motivated by the theory of persistent homology and its algorithmic aspects, new theorems, called *diamond principles*, have been recently introduced to track the changes in the algebraic decomposition of a quiver representation under local transformations. We present in this article a new proof of Gabriel's theorem for A_n-type quivers using these principles. The proof uses an elementary recursive argument and leads to an efficient algorithm for decomposing quiver representations.

1 Context

This work originated from the study of persistent homology in computational topology. In [1], Carlsson and de Silva use the framework of quiver theory to explain and compute zigzag persistent homology. They also introduce an *exact diamond principle* to transform a zigzag module locally. More recently, Maria and Oudot [3] introduced new diamond principles, called *injective and surjective diamond principles*, to compute more complex local changes in a module and deduce a new algorithm for decomposing zigzag persistence modules.

In this article, we use these last principles to give a new proof of Gabriel's theorem for A_n-type quivers, a fundamental decomposition theorem in quiver theory. To do so, we introduce a *dualized* version of the injective and surjective diamond principles. Using these diamonds and their duals, we deduce a constructive proof of Gabriel's theorem for A_n-type quivers. This proof may be turned directly into an efficient algorithm for decomposing A_n-type quiver representations.

This work improves the results from [3] in two directions. Firstly, the use of dualized diamonds simplifies the decomposition algorithm of [3], which requires injective and surjective diamonds as well as *transposition diamonds*, which are diamonds of a different nature. Secondly, this work presents a constructive approach in full generality (*i.e.*, at the level of quiver representations), whereas [3] focuses on zigzag persistence modules. This work in particular illustrates how results in computational topology may be used in a more general context to solve problems in computational mathematics.

2 Quiver Theory

Let $(\mathbb{F}, +, \cdot)$ be an arbitrary field. An *A_n-type quiver $\mathcal{Q}$* is a directed graph:

$$\bullet_1 \longleftrightarrow \bullet_2 \longleftrightarrow \cdots \longleftrightarrow \bullet_{n-1} \longleftrightarrow \bullet_n$$

where, by convention in this article, bidirectional arrows are either forward or backward.

An *$\mathbb{F}$-representation* of $\mathcal{Q}$ is an assignment of a finite dimensional $\mathbb{F}$-vector space V_i for every node $\bullet_i$ and an assignment of a linear map $f_i \colon V_i \to V_{i+1}$ for every forward arrow $\bullet_i \to \bullet_{i+1}$ or $f_i \colon V_{i+1} \to V_i$ for every backward arrow $\bullet_i \leftarrow \bullet_{i+1}$. We denote such a representation by $\mathbb{V} = (V_i, f_i)$.

Let $\mathbb{V} = (V_i, f_i)$ and $\mathbb{W} = (W_i, r_i)$ be two $\mathbb{F}$-representations of a quiver $\mathcal{Q}$. A *morphism of representations* $\phi \colon \mathbb{V} \to \mathbb{W}$ is a set of linear maps $\{\phi_i : V_i \to W_i\}_{i=1\dots n}$ such that, for every arrow, one of the following two diagrams commutes (depending on the orientation of arrow $\bullet_i \leftrightarrow \bullet_{i+1}$):

$$\begin{tikzcd} V_i \arrow[r, "f_i"] \arrow[d, "\phi_i"'] & V_{i+1} \arrow[d, "\phi_{i+1}"] \\ W_i \arrow[r, "r_i"] & W_{i+1} \end{tikzcd} \qquad \begin{tikzcd} V_i \arrow[d, "\phi_i"'] & V_{i+1} \arrow[l, "f_i"'] \arrow[d, "\phi_{i+1}"] \\ W_i & W_{i+1} \arrow[l, "r_i"] \end{tikzcd}$$

The morphism is called an *isomorphism*, denoted by $\cong$, if every ϕ_i is bijective. Finally, the $\mathbb{F}$-representations of $\mathcal{Q}$ admit a *direct sum* denoted by $\oplus$. For any $\mathbb{V} = (V_i, f_i), \mathbb{W} = (W_i, r_i)$, define the $\mathbb{F}$-representation $\mathbb{V} \oplus \mathbb{W}$ to be the representation of $\mathcal{Q}$ with spaces $V_i \oplus W_i$, for all i, and maps $f_i \oplus r_i = \left(\begin{smallmatrix} f_i & 0 \\ 0 & r_i \end{smallmatrix}\right)$ for every arrow. An $\mathbb{F}$-representation $\mathbb{V}$ is *decomposable* if it is isomorphic to the direct sum of two non-trivial representations. It is otherwise *indecomposable*.

Finally, define an *interval representation* $\mathbb{I}[b; d]$, $1 \leq b \leq d \leq n$, of an A_n-type quiver to be the $\mathbb{F}$-representation:

$$\underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[1;b-1]} \overset{0}{\longleftrightarrow} \underbrace{\mathbb{F} \overset{\mathbb{1}}{\longleftrightarrow} \cdots \overset{\mathbb{1}}{\longleftrightarrow} \mathbb{F}}_{[b;d]} \overset{0}{\longleftrightarrow} \underbrace{0 \overset{0}{\longleftrightarrow} \cdots \overset{0}{\longleftrightarrow} 0}_{[d+1;n]}$$

where 0 and $\mathbb{F}$ stand respectively for the 0 and 1-dimensional vector spaces, and the maps 0 and $\mathbb{1}$ stand respectively for the null map and the identity map.

Theorem 1 states that the indecomposable representations of an A_n-type quiver are the interval representations:

[*]This work was supported by the Australian Research Council (project DP140104246).

[†]c.maria@uq.edu.au The University of Queensland

Theorem 1 (Gabriel [2]) *Every $\mathbb{F}$-representation $\mathbb{V}$ of an A_n-type quiver is decomposable as a direct sum of indecomposables:*

$$\mathbb{V} = \mathbb{V}^1 \oplus \mathbb{V}^2 \oplus \cdots \oplus \mathbb{V}^N,$$

where each indecomposable $\mathbb{V}^j$ is isomorphic to some interval representation $\mathbb{I}[b_j; d_j]$.

For an $\mathbb{F}$-representation $\mathbb{V} = (V_i, f_i)_{i=1\ldots n}$ of an A_n-type quiver $\mathcal{Q}$, we define $\mathbb{V}[b; d]$ to be the representation $(V_i, f_i)_{i=b\ldots d}$ of the quiver $\mathcal{Q}[b; d]$ restricted to the vertices (and arrows between them) of indices $b \leq i \leq d$. We call $\mathbb{V}[b; d]$ a *restriction* of the representation $\mathbb{V}$ to the range $[b; d]$. If $b = 1$, we call $\mathbb{V}[1; d]$ a *prefix* of $\mathbb{V}$ and if $d = n$ we call $\mathbb{V}[b; n]$ a *suffix*. The restriction principle [1] states that the interval decomposition of a representation $\mathbb{V}[i; j]$ restricted to indices between i and j behaves naturally, *i.e.*, is equal to the direct sums of the intervals of the decomposition of $\mathbb{V}$ restricted, as intervals, to $[i; j]$.

3 Diamond Principles

Maria and Oudot [3] have recently introduced the *surjective and injective diamond principles*, that relate the change in the interval decomposition when passing from the bottom to top representation in the following diagram:

$$\begin{tikzcd}
\mathbb{W} := & & & & W & & & \\
& V_1 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & V \arrow[ur, "f"] & & V \arrow[ul, "f"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & V_n \\
\mathbb{V} := & & & & V \arrow[ul, "\mathbb{1}"] \arrow[ur, "\mathbb{1}"'] & & &
\end{tikzcd} \tag{1}$$

where f is either surjective of nullity 1 or injective of corank 1. The diagram can be "dualized" into:

$$\begin{tikzcd}
\mathbb{W} := & & & & W \arrow[dl, "g"'] \arrow[dr, "g"] & & & \\
& V_1 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & V \arrow[dr, "\mathbb{1}"'] & & V \arrow[dl, "\mathbb{1}"] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & V_n \\
\mathbb{V} := & & & & V & & &
\end{tikzcd} \tag{2}$$

and a similar diamond principle can be obtained [1]. The principles give an explicit transformation of the interval decompositions.

4 Proofs of Gabriel's Theorem for A_n-type Quivers

Consider an arbitrary representation $\mathbb{V} = \begin{tikzcd} V_1 \arrow[r, leftrightarrow, "f_1"] & \cdots \arrow[r, leftrightarrow, "f_{n-1}"] & V_n \end{tikzcd}$ of an A_n-type quiver. Every map f_i can be decomposed into a bijective map and a sequence of surjective maps of nullity 1 and injective maps of corank 1. Hence we assume w.l.o.g. that each map f_i is of one of these three types.

[1]Proof to appear in a longer version of this article.

The proof is inductive. We first notice that the representation V_1 of the quiver with one node $\mathcal{Q}[1; 1] = \bullet_1$ admits an interval decomposition equal to the direct sum of $(\dim V_1)$ times the interval $\mathbb{I}[1; 1]$.

Suppose that at iteration i, we are given the representation $\mathbb{V}_i :=$

$$\begin{tikzcd}
V_1 \arrow[r, leftrightarrow, "f_1"] & \cdots \arrow[r, leftrightarrow, "f_{i-1}"] & V_i \arrow[r, leftrightarrow, "h_i"] & W_i \arrow[r, leftrightarrow, "h_{i+1}"] & \cdots \arrow[r, leftrightarrow, "h_{m-1}"] & W_m
\end{tikzcd}$$

which is identical in its first i vector spaces and $i-1$ maps to the prefix $\mathbb{V}[1; i]$. Suppose this representation admits an interval decomposition. We distinguish the cases:

(i) If f_i is bijective, the representation $\mathbb{V}_{i+1}$:=

$$\begin{tikzcd}
V_1 \arrow[r, leftrightarrow, "f_1"] & \cdots \arrow[r, leftrightarrow, "f_{i-1}"] & V_i \arrow[r, leftrightarrow, "f_i"] & V_{i+1} \arrow[r, leftrightarrow, "f_i"] & V_i \arrow[dl, leftrightarrow, bend left, "h_i"'] & & \\
& & & W_i \arrow[r, leftrightarrow, "h_{i+1}"'] & \cdots \arrow[r, leftrightarrow, "h_{m-1}"'] & W_m &
\end{tikzcd}$$

where the two arrows labeled f_i have opposite orientations, admits naturally an interval decomposition. Now suppose f_i is not bijective.

(ii) If $\bullet_i \longrightarrow \bullet_{i+1}$ is forward, we apply the diamond principle 1 to:

$$\begin{tikzcd}
\mathbb{V}_{i+1} := & & & & V_{i+1} & & & \\
& V_1 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & V_i \arrow[ur, "f_i"] & & V_i \arrow[ul, "f_i"'] \arrow[r, leftrightarrow, "h_i"] & \cdots \arrow[r, leftrightarrow, "h_{m-1}"] & W_m \\
\widetilde{\mathbb{V}_i} := & & & & V_i \arrow[ul, "\mathbb{1}"] \arrow[ur, "\mathbb{1}"'] & & &
\end{tikzcd}$$

where $\widetilde{\mathbb{V}_i}$ admits an interval decomposition by virtue of *(i)*, as it is equal to $\mathbb{V}_i$ with two additional bijective maps $\mathbb{1}$. Hence $\mathbb{V}_{i+1}$ admits an interval decomposition.

(iii) If $\bullet_i \longleftarrow \bullet_{i+1}$ is backward, we proceed similarly by applying the dual diamond principle

In all three cases, the prefix $\mathbb{V}_{i+1}[1; i+1]$ of the representation $\mathbb{V}_{i+1}$ obtained is identical to $\mathbb{V}[1; i+1]$.

Consequently, at iteration n of the process described above, we find that a representation $\mathbb{V}_n$, whose prefix $\mathbb{V}_n[1; n]$ is exactly $\mathbb{V}$, admits an interval decomposition. By virtue of the restriction principle, truncating a quiver does not change the property of having an interval decomposition. Hence, $\mathbb{V}_n[1; n] = \mathbb{V}$ admits an interval decomposition. $\square$

References

[1] G. E. Carlsson and V. de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[2] P. Gabriel. Unzerlegbare darstellungen. *Manuscripta Mathematica*, 1972.

[3] C. Maria and S. Y. Oudot. Zigzag persistence via reflections and transpositions. In *SODA 2015*, pages 181–199, 2015.

Pose Statistics for Eccentric Parts

Fatemeh Panahi[*] Aviv Adler [†] A. Frank van der Stappen[*]

Abstract

In many contexts, it is very useful to have an estimate of the final orientation, or *pose*, of an object which is dropped onto a flat surface. In this paper, we consider the final orientation of an object which starts with a random orientation, and show how the shape of the object relates to the distribution of its final orientation. We define a notion of *geometric eccentricity* for d-dimensional objects, which takes into account both its shape and its center-of-mass. We show that under quasi-static conditions, the pose into which eccentric objects settle will be with high probability in a cluster of poses which are very close together. Furthermore, the probability of ending up in this range of poses increases, and the size of the range decreases, as the object gets more eccentric.

1 Introduction

Manufactured parts often have to be oriented with part feeders before they are put through assembly lines; the shape of the part is usually known, but its starting orientation or location is not. The basic objective of part feeders is to minimize the uncertainty on the orientation of the part before the next stage of the manufacturing process. There are various part feeding systems which have been thoroughly studied. For such systems it is important to consider how parts will arrive on the supporting surface when their initial pose is unknown. When a part arrives on the surface, it settles into an orientation such that it does not topple over under the influence of gravity; this is called a *stable pose.* Evaluating the stable poses of an object allows estimation of the likelihood of what orientation it will arrive in, which can result in faster and more effective design of part feeders as well as many other automated tasks. There are a number of works estimating the probability distribution over a part's stable poses when the part falls on a flat worksurface in presence of gravity [1], [2].

For some parts, it is possible to predict that the pose which they settle into will (with high probability) be in a cluster of poses which are very close together. Considering these poses as the most probable initial orientations of the part for feeding allows faster design for part feeding tasks. Observing the usefulness of bias in pose distribution for part feeding, we consider a variant of the notion of *geometric eccentricity* for objects and show that there is a high probability for an eccentric object being dropped on a flat surface to come to the rest in a small range of stable orientations. Broadly, eccentricity is the degree to which a part deviates from being uniformly wide; generally, eccentric 3D objects are long and thin like a pencil (longer in one dimension) or wide and flat like a coin (longer in two dimensions). It is easy to see that a pencil is more likely to rest at one of its long sides and a coin will rest at one of its two larger sides. Considering the set of final orientations where these objects end up with high probability, they lie on a single plane (for the pencil) or on a single line (for the coin).

Fatness is a shape-related notion that has led to improved bounds or better solutions to many problems in the field of computational geometry (a fairly recent survey of the results can be found in [3].) We note that eccentricity can be regarded as the opposite (or lack) of fatness.

2 Geometric eccentricity

The definition of eccentricity that we propose applies to d-dimensional objects and captures the property that an object is a considerable factor, say k, bigger in b of its dimensions than in any of the remaining $d-b$ dimensions. It generalizes an earlier strictly 2D notion based on the aspect ratio of a bounding box [4] and allows us to distinguish between 3D objects similar to the pencil and similar to the coin.

Definition 1 ((b)-Eccentricity) *Let $P \subset \mathbb{R}^d$ be a bounded convex set with its center of mass at the origin O. For any $1 \leq b < d$, the set P is said to be k-(b)-*eccentric *for some $k \geq 1$ if there exists a scaled and rotated copy of P such that*

- *the projection of P onto the subspace spanned by b of its dimensions contains the b-dimensional sphere of radius k centered at O, and*
- *the projection of P onto the subspace spanned by the remaining $d-b$ dimensions is contained in the $(d-b)$-dimensional sphere of unit radius centered at O.*

[*]Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands. `f.panahi@uu.nl`, `a.f.vanderstappen@uu.nl`

[†]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA. `adlera@mit.edu`

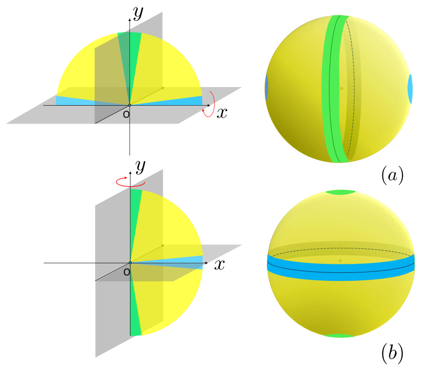


Figure 1: Three types of orientations for **(a)** a k-(1)-eccentric 3D object and **(b)** for a k-eccentric 3D object.

Using our definition for eccentricity, we show that for 3D objects similar to a pencil, there is a plane which the final orientation is, with high probability, going to end up near; for objects similar to a coin there is a line which the final orientation is going to end up near.

3 Pose and Probabilities

We focus on 3D objects with eccentricity $k > 2\sqrt{2}$. We assume that the only force acting on the object is gravity and for simplicity we do not consider dynamics. When an object is dropped onto a flat horizontal surface, it moves downward until it contacts the surface. If the center-of-mass is not directly over a point where the object contacts the surface then it will rotate so the center-of-mass descends as quickly as possible. We assume without loss of generality that the longer dimention of a 3D k-(1)-eccentric object is aligned with the x-axis and the two longer dimensions of a 3D k-(2)-eccentric object are aligned with the (x, z)-plane.

We use $\mathbb{S}^{d-1}$ to refer to the set of *orientations* in d-dimensional space. We define three types of orientations that decompose $\mathbb{S}^1$ and $\mathbb{S}^2$. Let $\theta_k = \arctan((k + \sqrt{k^2-8})/2)$ and $\phi_k = \arctan((k - \sqrt{k^2-8})/2)$. We first decompose $\mathbb{S}^1$. For a direction $u \in \mathbb{S}^1$ let $\alpha \leq \pi/2$ be the angle between u and the (positive or negative) y-axis. Then u is a *type-1* orientation, if $\alpha \geq \phi_k$, u is a *type-2* orientation, if $\theta_k < \alpha < \phi_k$, and u is a *type-3* orientation, if $\phi_k \leq \alpha$. The orientation types in S^2 associated with (1)-eccentric parts are obtained by rotating the decomposition of S^1 about the x-axis. The orientation types in S^2 associated with (2)-eccentric parts are obtained by rotating the decomposition of S^1 about the y-axis. The type-1, type-2, and type-3 orientations are displayed in *blue*, *yellow*, and *green* (respectively) in Fig. 1.

Considering the object's eccentricity, it can be shown that a 3D k-(1)-eccentric or k-(2)-eccentric object that is initially in a type-2 or type-3 pose always ends up in a type-3 pose. The idea to prove this is as follows. We first show that there is no type-2 pose is stable which means that the final orientation cannot be a type-2 pose. We define L to be the x-axis in the k-(1)-eccentric case and the (x, z)-plane in the k-(2)-eccentric case. It can be proven that as the part rotates, the minimum angle between the current orientation and L cannot decrease; this means that if it starts in a type-2 or type-3 orientation, it cannot enter the set of type-1 orientations. Thus, it must settle in a type-3 pose.

Theorem 1 *Let $P \subset \mathbb{R}^3$ be a k-(1)-eccentric or k-(2)-eccentric object ($k > 2\sqrt{2}$) in a uniformly random initial orientation. If P is k-(1)-eccentric then it settles*

- *with probability at least $\sin\theta_k$ in a spherical segment of orientations symmetrically surrounding a plane that covers a fraction $\sin\phi_k$ of $\mathbb{S}^2$,*
- *and in one of two antipodal spherical caps of orientations surrounding a line that jointly cover a fraction $1 - \sin\theta_k$ of $\mathbb{S}^2$ otherwise.*

If P is k-(2)-eccentric then it settles

- *with probability at least $1 - \cos\theta_k$ in one of two antipodal spherical caps of orientations surrounding a line that jointly cover a fraction $1 - \cos\phi_k$ of $\mathbb{S}^2$*
- *and in one spherical segment of orientations symmetrically surrounding a plane that covers a fraction $\cos\theta_k$ of $\mathbb{S}^2$ otherwise.*

The tight bounds of the theorem shows that there is a smaller range of poses that the object always can settle in and there is a higher probability for the object to settle in again smaller range of poses.

References

[1] J. Wiegley, A. Rao, and K. Goldberg, Computing a statistical distribution of stable poses for a polyhedron, Proc. of Allerton Conference Communications, Control and Computing, 1992.

[2] K. Goldberg, B. Mirtich, Y. Zhuang, J. Craig, B. Carlisle, and J. Canny. Part Pose Statistics: Estimators and Experiments. IEEE Transactions on Robotics and Automation. 15, 1999, pp. 849-857.

[3] C. M. Gray, Algorithms for Fat Objects: Decompositions and Applications. Ph. D. Dissertation, Technische Universiteit Eindhoven, Netherlands, 2008.

[4] A. F. van der Stappen, K. Goldberg and M. H. Overmars, Geometric eccentricity and the complexity of manipulation plans, Algorithmica 26, 2000, pp. 494-514.

Subsampling in Smoothed Range Spaces*

Jeff M. Phillips
University of Utah, SLC, USA
jeffp@cs.utah.edu

Yan Zheng
University of Utah, SLC, USA
yanzheng@cs.utah.edu

Abstract

We consider smoothed versions of geometric range spaces, so an element of the ground set (e.g. a point) can be contained in a range with a non-binary value in $[0, 1]$. Similar notions have been considered for kernels; we extend them to more general types of ranges. We then consider approximation of these range spaces through ε-nets and ε-samples (aka ε-approximations). We characterize when size bounds for ε-samples on kernels can be extended to these more general smoothed range spaces. We also describe new generalizations for ε-nets to these range spaces and show when results from binary range spaces can carry over to the smoothed ones.

1 Introduction

Combinatorial range spaces play a central role in geometry and have important connections to many areas, notably learning theory [6, 3], data structures, and recently differential privacy. We will focus on geometric range spaces where the ground set P is a point set in $\mathbb{R}^d$. The family of ranges $\mathcal{A}$ are typically defined by sets of subsets contained in some geometric objects, e.g., a disk, or a halfspace. The pair $(P, \mathcal{A})$ is called a *range space*.

An important consideration is how well we can approximate these objects through a subset $Q \subset P$, formalized as an ε-sample (aka ε-approximation, which preserves density) and an ε-net (which perverse the existence of large subsets). Formally, an *ε-sample* for a range space $(P, \mathcal{A})$ is a subset $Q \subset P$ s.t.

$$\max_{A\in\mathcal{A}} \left| \frac{|A \cap P|}{|P|} - \frac{|Q \cap A|}{|Q|} \right| \leq \varepsilon.$$

An *ε-net* of a range space $(P, \mathcal{A})$ is a subset $Q \subset P$ s.t. for all $A \in \mathcal{A}$ such that $\frac{|P\cap A|}{|P|} \geq \varepsilon$ then $A \cap Q \neq \emptyset$.

Through techniques ranging from discrepancy theory to Fourier analysis to basic combinatorics, we now largely understand these relationship of these bounds to the size of the subsets Q, for geometrically described ranges and with constructions; see a pair of great books [4, 1]. However, at least at a high-level, many of these size lower bounds are constructed with sets P so that problematic subsets $A \in \mathcal{A}$ have many elements near the boundary. This leads to the question, *what if we smoothed out this boundary?*

Background on Kernels and Kernel Range Spaces. This question was studied in the context of ε-samples for statistical kernels (e.g. Gaussians). A *kernel* is a bivariate similarity function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, which can be normalized so $K(x, x) = 1$ (which we assume through this paper). We focus on symmetric, shift invariant kernels which can be written as a single parameter function $K(x, p) = k(\|x - p\|)$, so it usually decreases as $\|x-p\|$ increases; these can be parameterized by a single bandwidth (or just width) parameter w so $K_w(x, p) = k_w(\|x - p\|) = k(\|x - p\|/w)$. Most commonly used kernels are Gaussian, Laplace, Triangular, Epanechnikov, and Ball kernels.

A *kernel range space* [2, 5] $(P, \mathcal{K})$ is an extension of the combinatorial concept of a range space $(P, \mathcal{A})$ (or to distinguish it we refer to the classic notion as a *binary range space*). It is defined by a point set $P \subset \mathbb{R}^d$ and a set of kernels $\mathcal{K}$. An element of $\mathcal{K}$ is a kernel $K(x, \cdot)$ applied at point $x \in \mathbb{R}^d$; it assigns a value in $[0, 1]$ to each point $p \in P$ as $K(x, p)$.

Given a point set P of size n and a kernel K, a *kernel density estimate* KDE_P is the convolution of that point set with K. For any $x \in \mathbb{R}^d$ we define $\text{KDE}_P(x) = \frac{1}{n}\sum_{p\in P} K(x, p)$. The notion of ε-kernel sample [2] extends the definition of ε-sample. It is a subset $Q \subset P$ such that $\max_{x\in\mathbb{R}^d} |\text{KDE}_P(x) - \text{KDE}_Q(x)| \leq \varepsilon$.

A binary range space $(P, \mathcal{A})$ is *linked* to a kernel range space $(P, \mathcal{K})$ if the set $\{p \in P \mid K(x, p) \geq \tau\}$ is equal to $P \cap A$ for some $A \in \mathcal{A}$, for any threshold value τ.

Two main observations have been made in the kernel range spaces. (1) An ε-sample for a (linked) range space defined by balls, is also an ε-sample for kernels [2]. (2) Using a careful discrepancy-based approach, smaller ε-samples (sometimes significantly smaller) can be constructed for kernels than for balls [5]. In this article we extend this line of work in a few interesting directions.

Contributions.

- We define a general class of *smoothed range spaces*, with application to density estimation.
- We define a notion of an (ε, τ)-net for a smoothed range space. We show how this can inherit sam-

*Thanks to support by NSF CCF-1350888, IIS-1251019, and ACI-1443046.

pling complexity bounds from *linked* non-smooth range spaces. We also relate this concept to a smoothed hitting set problem.

- We provide discrepancy-based bounds and constructions for ε-samples on smooth range spaces requiring significantly fewer points than uniform sampling approaches and discrepancy-based approaches on the linked binary range spaces.

2 Smoothed Range Spaces

Let $\mathcal{H}_w$ denote the family of *smoothed halfspaces with width parameter* w, and let $(P, \mathcal{H}_w)$ be the associated smoothed range space where $P \subset \mathbb{R}^d$. Given a point $p \in P$, the smoothed halfspace $h \in \mathcal{H}_w$ maps p to a value $v_h(p) \in [0,1]$ (rather than the traditional $\{0,1\}$ in a binary range space).

We first describe a specific mapping to the function value $v_h(p)$. Let F be the $(d-1)$-flat defining the boundary of halfspace h. Given a point $p \in \mathbb{R}^d$, let $p_F = \arg\min_{q \in F} \|p - q\|$ describe a point on F closest to p. We make the definition more general using a shift-invariant kernel $k_w(\|p - x\|) = k(\|p - x\|/w)$ such that we define $v_{h,w}(p)$ as follows.

$$v_{h,w}(p) = \begin{cases} \frac{1}{2} + \frac{1}{2}k_w(\|p - p_F\|) & p \in h \\ \frac{1}{2} - \frac{1}{2}k_w(\|p - p_F\|) & p \notin h. \end{cases}$$

For brevity, we will omit the w and just use $v_h(p)$ when clear. We can also further generalize this by replacing the flat F at the boundary of h with a polynomial surface G. The point $p_G = \arg\min_{q \in G} \|p - q\|$ replaces p_F in the above definitions. Then the slab of width $2w$ is replaced with a more curved volume in $\mathbb{R}^d$; see Figure 1. For concreteness and simplicity, the remainder of this note will focus on halfspaces.

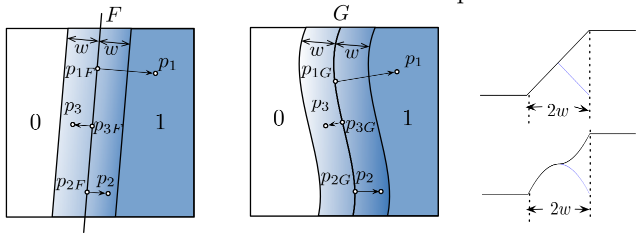


Figure 1: Illustration of the smoothed halfspace F (left), and smoothed polynomial surface G (middle).

We extend the notion of a kernel density estimate to these smoothed range spaces. A *smoothed density estimate* SDE_P is defined for any $h \in \mathcal{H}_w$ as

$$\mathrm{SDE}_P(h) = \frac{1}{|P|}\sum_{p \in P} v_h(p).$$

Then an ε-sample Q of a smoothed range space $(P, \mathcal{H}_w)$ is a subset $Q \subset P$ such that

$$\max_{h \in \mathcal{H}_w} |\mathrm{SDE}_P(h) - \mathrm{SDE}_Q(h)| \le \varepsilon.$$

(ε, τ)-Net for smoothed range spaces. We introduce two new definitions to generalize the definition of hitting and ε-net. A subset $Q \subset P$ is an (ε, τ)-*net of smoothed range space* $(P, \mathcal{H}_w)$ if for any $h \in \mathcal{H}_w$ such that $\mathrm{SDE}_P(h) \ge \varepsilon$, there exists a point $q \in Q$ such that $v_h(q) \ge \tau$. A subset $Q \subset P$ is an (ε, τ)-*hitting set of smoothed range space* $(P, \mathcal{H}_w)$ if for any $h \in \mathcal{H}_w$ such that $\mathrm{SDE}_P(h) \ge \varepsilon$, then $\mathrm{SDE}_Q(h) \ge \tau$. We can show that both of these notions are implied by an $(\varepsilon - \tau)$-sample.

Theorem 1 *An $(\varepsilon - \tau)$-sample Q in a smoothed range space $(P, \mathcal{H}_w)$ is an (ε, τ)-hitting set in $(P, \mathcal{H}_w)$, and thus also an (ε, τ)-net of $(P, \mathcal{H}_w)$.*

Consider a smoothed range space $(P, \mathcal{H}_w)$, a linked binary range space $(P, \mathcal{A})$, and an ε-sample Q of $(P, \mathcal{A})$. Prior results for kernels [2] can be generalized to show Q is an ε-sample of $(P, \mathcal{H}_w)$. We can further extend this relation for (ε, τ)-nets; thus they can require significantly smaller size sets Q to satisfy.

Theorem 2 *Consider a smoothed range space $(P, \mathcal{H}_w)$, a linked binary range space $(P, \mathcal{A})$, and an $(\varepsilon - \tau)$-net Q of $(P, \mathcal{A})$. Then Q is an (ε, τ)-net of $(P, \mathcal{H}_w)$.*

Discrepancy-based approaches. We improve on random sample bounds using discrepancy [4, 1]. These results are restricted to when points P are contained in a d-dimensional cube $\mathsf{C}_{\ell,d}$ of side length ℓ.

Theorem 3 *In $\mathbb{R}^2$, for any $P \subset C_{\ell,2}$, we can construct an ε-sample of $(P, \mathcal{H}_w)$ of size $O(\frac{1}{\varepsilon}\sqrt{\frac{\ell}{w}\log\frac{\ell}{w\varepsilon\delta}})$ with probability at least $1 - \delta$.*

Theorem 4 *In $\mathbb{R}^d$, for any $P \subset \mathsf{C}_{\ell,d}$ with d is constant, we can construct an ε-sample of $(P, \mathcal{H}_w)$ of size $O\Big((\ell/w)^{2(d-1)/(d+2)} \cdot \Big(\frac{1}{\varepsilon}\sqrt{\log\frac{\ell}{w\varepsilon\delta}}\Big)^{2d/(d+2)}\Big)$ with probability at least $1 - \delta$,*

We can improve some results if the data is "well-clustered" under other specific conditions.

References

[1] B. Chazelle. *The Discrepancy Method.* Camb., 2000.

[2] S. Joshi, R. V. Kommaraju, J. M. Phillips, and S. Venkatasubramanian. Comparing distributions and shapes using the kernel distance. *SOCG*, 2011.

[3] Y. Li, P. M. Long, and A. Srinivasan. Improved bounds on the samples complexity of learning. *J. Comp. and Sys. Sci.*, 62:516–527, 2001.

[4] J. Matoušek. *Geometric Discrepancy.* Sprgr., 1999.

[5] J. M. Phillips. eps-samples for kernels. *SODA*, 2013.

[6] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Th of Prob., Apps.*, 16:264–280, 1971.

Approximate Diameter with Inexact Input Data

Masood Seddighin* Hamid Homapour* Mohammad Ghodsi*

Abstract

In this paper, we study the minimum diameter problem with inexact data. First, we focus on data in the indecisive model and present an $O(2^{\frac{1}{\epsilon^d}} \cdot \epsilon^{-2d} \cdot n^2)$ time approximation algorithm of factor $(1+O(\epsilon))$. Next, we consider the problem in the imprecise model. In d-dimensional space, we propose a polynomial time $\sqrt{d}$-approximation algorithm for constant d. In addition, for $d = 2$, we define the notion of α-separability and then use our result in the indecisive model to obtain an $(1 + O(\epsilon))$ approximation algorithm for α-separable regions. Time complexity of our algorithm is $O(2^{\frac{1}{\epsilon^2}} \cdot \frac{n^2}{\epsilon^8 (\sin \frac{\alpha}{2})^2})$.

1 Introduction

In computational geometry, there are various optimization problems that measure the properties of given data or compute structures on these data. In the design and analysis of computer algorithms often, the assumption is that the given data is precise. Thus, these algorithms work based on the input data without any imprecision. However, in most cases, the input data are imprecise. This imprecision in input leads to inaccurate output.

Several models have been defined for dealing with inexact inputs. For example, in the indecisive model for each point as an input data, there exist a finite number of copies that show possible locations for that point. Another example is the imprecise model in which for an input point potential locations are restricted by a region.

In this paper, we study the problem of finding the minimum diameter in both above-mentioned models. The formal definitions of these two problems are as follows.

MinDCS problem. Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in d dimensional Euclidean space colored with m colors (meaning that P is a set of indecisive points). Select one point of each color so that the diameter of these points is smallest among all options.

MinDiam problem. Given a set $R = \{R_1, R_2, \ldots, R_n\}$, where each R_i is a region. Select one point from each region so that the diameter of these points is smallest among all options.

We discuss on $MinDiam$, for the case when the regions are convex with polynomial complexity.

Fleischer and Xu [3, 2] demonstrated that $MinDCS$ can be solved in polynomial time for the L_1 and L_∞ metrics, while it is NP-hard for all other L_p metrics, even in two dimensions. They also gave an efficient algorithm to compute a constant factor approximation. Consuegra et al. [1] by extending the definition of ϵ-kernels to avatar ϵ-kernels (the notion of avatar is the same as the indecisive points) in d-dimensional Euclidean space proposed an $(1+\epsilon)$ approximation algorithm with running time $O((n)^{(2d+3)} \cdot \frac{m}{\delta^d} \cdot (2d)^2 \cdot 2^{\frac{1}{\delta^d}} (\frac{2}{\delta^{d-1}})^{\lfloor \frac{d}{2} \rfloor} + (\frac{2}{\delta^{d-1}})^a)$. δ is the side length of the grid cells for constructing core-set.

For the special case, when the shapes of regions are square, Loffler and van Kreveld proposed an $O(n \log n)$ time algorithm for computing $MinDiam$. When the shapes of regions are disk, they also proposed an $(1+\epsilon)$ approximation algorithm with running time $O(n^{c\epsilon^{-\frac{1}{2}}})$, where $c = 6.66$ [4].

2 MinDCS

In this section, we present an $O(2^{\frac{1}{\epsilon^d}} \cdot \epsilon^{-2d} \cdot n^2)$ time approximation algorithm of factor $(1+O(\epsilon))$ for $MinDCS$.

Definition 1 (C-legal) *Given a set of colored points P and a set C of colors. P is C-legal iff for each $c \in C$ there exists a point $p \in P$ with color c.*

Definition 2 (Possible Area) *Consider two points p and q. Draw two balls of radius $|pq| + \epsilon$ that one of them centered at p and the other centered at q. We name the intersection area of these two balls as possible area (C_{pq}).*

Note that, if p and q are the two points that define the diameter of a point set, then all the points must lie in C_{pq}.

Approximate minimum diameter. Let $S = \{S_1, S_2, \ldots\}$ be the set of all pairs of points with different colors in P. For each $S_i = \{p, q\}$, let P_i be the set of points in C_{pq}. For each C-legal P_i, according to $|pq|$ we can compute an approximation of minimum diameter of P_i (based on finding an ϵ-coreset for the points in C_{pq}, as analogous to that of the manner in [1]). Then between all computation we choose the minimum of them (D_{alg}).

*Sharif University of Technology

Note that, since for some P_i we have $D_{min} = |pq|$ then D_{alg} is an ϵ-approximation of D_{min}.

Theorem 1 *The MinDCS problem can be approximated in time $O(2^{\frac{1}{\epsilon^d}} \cdot \epsilon^{-2d} \cdot n^2)$ of factor $(1 + O(\epsilon))$ for d-dimensions.*

3 MinDiam

We discuss on *MinDiam*, for the case when the regions are convex with polynomial complexity.

3.1 $\sqrt{d}$-approximation

Our $\sqrt{d}$-approximation construction is based on linear programming, which uses the rectilinear distances of the points. Consider *LP* 3.1. In this *LP*, we want to select the points $S_1, S_2, \ldots, S_n$, such that $S_i \in R_i$ and the rectilinear diameter of these points is minimized.

LP 3.1

$$\begin{array}{lll} \text{minimize} & \ell & \\ \text{subject to} & & \\ & d_{l_1}(S_i,\ S_j) \leq \ell & \forall i, j \\ & S_i \in R_i \ \leq \ell & \forall i, j \end{array}$$

We argue that the expression $d_{l_1}(S_i,\ S_j)$ can be described using 2^d linear constraints (note that $d_{l_1}(S_i,\ S_j) = |S_{i,1} - S_{j,1}| + \ldots + |S_{i,d} - S_{j,d}|$).

Because of the convexity of regions, the constraint $S_j \in R_i$ can be described using $|R_i|$ linear inequalities, where $|R_i|$ is the complexity of region R_i.

Theorem 2 *LP 3.1 yields to a $\sqrt{d}$-approximation algorithm for the MinDiam problem.*

3.2 ϵ-approximation

In this section, we propose an ϵ-approximation algorithm for *MinDiam*, when $d = 2$ and the regions in R admit a special notion of separability.

Definition 3 *Two intersecting lines, divide the plane into four areas. Two regions A and B are separated by these lines, if they completely belong to opposite areas. Two regions X and Y are α-separable, if there exists two intersecting lines separating these regions, with the degree of separation equals to α. For example, see Fig. 1. We also say a set R of regions is α-separable, if there exists two regions X and Y in R, which are α-separable.*

Theorem 3 *Let R be an α-separable set of regions. Then there exists an ϵ-approximation algorithm for computing MinDiam for R with running time $O(2^{\frac{1}{\epsilon^2}} \cdot \frac{n^2}{\epsilon^8 (\sin\frac{\alpha}{2})^2})$.*

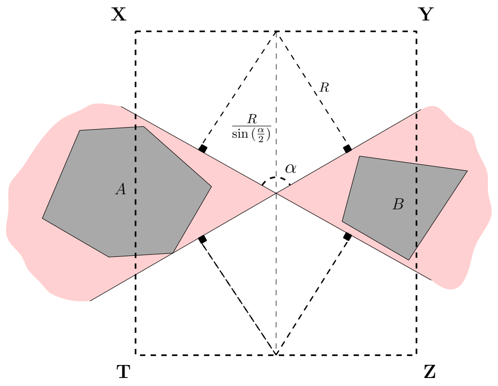


Figure 1: Regions are separated by two intersecting lines and the optimal solution S_{min}, must be entirely within the rectangle $XYZT$.

Proof. (sketch) Suppose we have a constant factor approximation of D_{min} based on LP 3.1 (say R). By considering the notion of separability of a set of regions, we know that the points in the optimal solution, must be entirely within the rectangle $XYZT$ (see Fig. 1). Taking this into account, we construct a uniform grid $\mathbb{G}$ on the rectangle $XYZT$. For each incidence between the grid points and a region, we create a point colored with a distinct color for each region. By applying the algorithm described in sectoin 2, we can get an $(1 + O(\epsilon))$ approximation of optimal solution. $\square$

Remark. The notion of α-separability is not well defined in the case, when all the regions have a nonempty intersection. We can address this issue as follows. Let R be the set of regions that can't be α-separated for any α. Then either the answer of *MinDiam* is zero, or we can divide the problem into a constant number of separable subproblems such that the best solution of these subproblems equals to *MinDiam* of S.

References

[1] M. E. Consuegra, G. Narasimhan, and S.-i. Tanigawa. Geometric avatar problems. In *FSTTCS*, 2013.

[2] R. Fleischer and X. Xu. Computing minimum diameter color-spanning sets. In *Frontiers in Algorithmics*. Springer, 2010.

[3] R. Fleischer and X. Xu. Computing minimum diameter color-spanning sets is hard. *IPL*, 2011.

[4] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 2010.

A hom-tree lower bound for the Reeb graph interleaving distance

Vin de Silva* Elizabeth Munch† Anastasios Stefanou‡

Abstract

The Reeb graph has become an increasingly common tool in applied topology. Recently, several definitions of a metric on Reeb graphs have been proposed, including the interleaving distance (de Silva et al.). Here we give a lower bound for the Reeb graph interleaving distance, by the related join tree interleaving distance (Morozov et al.) through the newly defined hom-tree construction.

1 Introduction

The Reeb graph is a construction which originated in Morse theory to study a real valued function defined on a topological space. Given a function $\tilde{f} : \mathbb{X} \to \mathbb{R}$ defined on a space $\mathbb{X}$, we construct the Reeb graph by collapsing path connected components of level sets of $\mathbb{X}$ and denote the resulting quotient space by $\Gamma_{\tilde{f}}$ or simply by Γ. Because the Reeb graph inherits a function from the original function, we denote this by $f : \Gamma \to \mathbb{R}$, or (Γ, f). The Reeb graph has been used widely in applications; see [2] for a survey. Because real data has noise, we are interested in methods for comparison of Reeb graphs which provide stability results.

There are several methods that have already been developed for defining a measure of similarity between these structures, including the functional distortion distance [1] and the combinatorial edit distance [7]. In this paper, we focus on the interleaving distance [6], which is inspired by the persistence module interleaving distance [5] and its equivalent definition in terms of category theory [3]. Moreover, we use a version of the interleaving distance for join trees (also called merge trees) [8], which we view as a subcategory of Reeb graphs, to construct a lower bound for the Reeb graph interleaving distance.

2 Related work

Reeb graphs can be identified with a particular kind of category theory construction called a cosheaf. These cosheaves may be compared using an interleaving distance of the kind studied in [3], which works by constructing almost-isomorphisms between the cosheaves and measuring distance based on the parameter necessary for this construction. This metric on cosheaves can be pulled back to a metric on the original topological constructions, which is what we call the interleaving distance [6]. Via a similar process, there is also an interleaving distance on join trees [8], which is a construction representing connected components of sublevel sets instead of the level sets used by the Reeb graph [4]. Both of these interleaving metrics come with bottleneck and L_∞ type stability results, making them especially useful for data analysis.

3 Interleaving Distances

Category theory generalizes set theory by studying morphisms between objects rather than just the objects themselves. Reeb graphs form a category which is in fact equivalent to the category of constructible cosheaves [6]. In this case, the cosheaves can be thought of as functors $\mathbf{Int} \to \mathbf{Set}$ from the category of open intervals to the category of sets. Given the Reeb graph of the function $f : \mathbb{X} \to \mathbb{R}$, this functor sends an interval I to $\pi_0(f^{-1}(I))$, the set of path connected components of the inverse image of the function. Likewise, we can store the information of the join tree in a functor $\mathbb{R} \to \mathbf{Set}$ which sends a real number a to $\pi_0(f^{-1}(-\infty, a])$, the set of path connected components of the sublevel sets of the function.

The idea of an interleaving metric between two functors $\mathsf{F}, \mathsf{G} : \mathcal{P} \to \mathbf{Set}$ (here, $\mathcal{P}$ is either $\mathbb{R}_{\geq 0}$ or $\mathbf{Int}$), is to measure how far F and G are from being equivalent. This involves finding a pair of natural transformations which use the shift functor and commute as much as possible; this is called an ε-interleaving where ε is the required amount of shift. Then one can define an interleaving distance on cosheaves, as follows:

$$d_I(\mathsf{F}, \mathsf{G}) = \inf\{\varepsilon \geq 0 \mid \text{there exists an } \varepsilon\text{-interleaving}\}.$$

4 The hom-tree construction

Given a Reeb graph (Γ, f), we consider its corresponding constructible cosheaf, $\mathsf{F} : \mathbf{Int} \to \mathbf{Set}$. The hom-tree construction defines a join tree for F through the following process. Fix a test Reeb graph $h : \mathrm{E} \to \mathbb{R}$ with associated cosheaf $\mathsf{H} : \mathbf{Int} \to \mathbf{Set}$. This could be, for example, a single line with a monotone function supported on a finite range. Define for each $\delta \geq 0$ the shift functor $\mathsf{T}_\delta : \mathbf{Int} \to \mathbf{Int}$ given by $(a, b) \mapsto (a - \delta, b + \delta)$. The hom-tree is defined to be the functor $\mathcal{H}(\mathsf{F}) : \mathbb{R}_{\geq 0} \to \mathbf{Set}$,

*Mathematics Department, Pomona College
†Mathematics Department, University at Albany, SUNY
‡Mathematics Department, University at Albany, SUNY

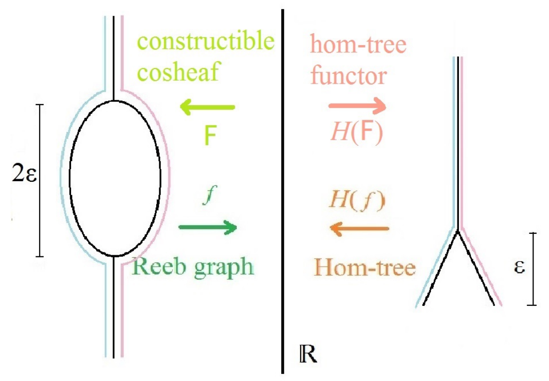


Figure 1: The hom-tree construction

$\delta \mapsto \mathrm{Hom}(\mathsf{H}, \mathsf{FT}_\delta)$, where $\mathrm{Hom}(\mathsf{H}, \mathsf{FT}_\delta)$ is the set of natural transformations from the functor H to FT_δ. There is an equivalent geometric construction of this functor-defined hom-tree which has the structure of a join tree; we denote the functor hom-tree as $\mathcal{H}(\mathsf{F})$ and the corresponding geometric object as $\mathcal{H}(f)$.

The motivation for studying hom-trees is to give a lower bound for the Reeb graph interleaving distance by the interleaving distance between the associated hom-trees. For the comparison of hom-trees, we pull back the interleaving distance of join trees, as studied in [8].

5 Example

We present the hom-tree construction, by a simple example given in Figure 1. Let (Γ, f) be a Reeb graph having a single hole of height 2ε as in the left of the figure, and let $\mathsf{F} : \mathbf{Int} \to \mathbf{Set}$, $I \mapsto \pi_0(f^{-1}(I))$ be the corresponding constructible cosheaf. In this example, we use the test space (E, h) which is the graph with two vertices and one vertical edge of height equal to the range of f. Let H be its associated cosheaf.

Consider the hom-tree $\mathcal{H}(\mathsf{F}) : \delta \to \mathrm{Hom}(\mathsf{H}, \mathsf{FT}_\delta)$. If $\delta < \epsilon$, there are exactly two natural transformations from H to FT_δ: the ones that transform H to the blue and purple colored curves after an δ-smoothing, respectively. This is represented by the two legs of the hom-tree, shown in the figure at right. Else, if $\delta \geq \varepsilon$, the hole in the Reeb graph shrinks enough to disappear because of the smoothing process, and the images of the blue and purple curves coincide. Hence, we get only one natural transformation between H and FT_δ, which is the identity transformation. This is represented by the straight line above the legs.

6 Comparing Reeb graphs with hom-trees

Our main result is the following theorem, proved using the machinery of category theory and the definitions of ε-interleavings on Reeb graphs and on join trees.

Theorem 6.1 *The interleaving distance for a pair of Reeb graphs $f : \Gamma \to \mathbb{R}$ and $g : \Delta \to \mathbb{R}$, is bounded below by the interleaving distance of the corresponding hom-trees, i.e.*

$$\hat{d}_I(\mathcal{H}(f), \mathcal{H}(g)) \leq d_I(f, g)$$

where $\hat{d}_I$ denotes the join tree interleaving, d_I denotes Reeb graph interleaving, and both $\mathcal{H}(f)$ and $\mathcal{H}(g)$ use the same test space $h : \mathrm{E} \to \mathbb{R}$ for the hom-tree construction.

7 Conclusion

We have defined a construction, the hom-tree, which can be used to define a lower bound on the Reeb graph interleaving distance by the join tree interleaving distance with respect to a fixed test space. We expect that this new result will allow for improved understanding of the Reeb graph interleaving distance.

References

[1] U. Bauer, X. Ge, and Y. Wang. Measuring distance between Reeb graphs. In *Annual Symposium on Computational Geometry*, page 464. ACM, 2014.

[2] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science: Computational Algebraic Geometry and Applications*, 392(13):5 – 22, 2008.

[3] P. Bubenik and J. Scott. Categorification of persistent homology. *Discrete & Computational Geometry*, 51(3):600–627, 2014.

[4] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[5] F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Y. Oudot. Proximity of persistence modules and their diagrams. In *Annual Symposium on Computational Geometry*, pages 237–246. ACM, 2009.

[6] V. de Silva, E. Munch, and A. Patel. Categorification of Reeb graphs. *arXiv:1501.04147*, 2015.

[7] B. di Fabio and C. Landi. The edit distance for Reeb graphs of surfaces. *arXiv: 1411.1544*, 2014.

[8] D. Morozov, K. Beketayev, and G. Weber. Interleaving distance between merge trees. In *Proceedings of TopoInVis*, 2013.

On the Hardness of Unlabeled Multi-Robot Motion Planning

Kiril Solovey* Dan Halperin*

Abstract

In *unlabeled* multi-robot motion planning several interchangeable robots operate in a common workspace. The goal is to move the robots to a set of target positions such that each position will be occupied by *some* robot. In this paper, we study this problem for the specific case of unit-square robots moving amidst polygonal obstacles and show that it is PSPACE-hard. We also consider three additional variants of this problem and show that they are all PSPACE-hard as well. To the best of our knowledge, this is the first hardness proof for the unlabeled case. Furthermore, our proofs can be used to show that the *labeled* variant (where each robot is assigned with a specific target position), again, for unit-square robots, is PSPACE-hard as well, which sets another precedence, as previous hardness results require the robots to be of different shapes.

1 Introduction

Hopcroft et al. [3] were the first to consider the problem of multi-robot motion planning with an arbitrary number of robots. Specifically, they studied the standard (*labeled*) case. They showed that this problem is PSPACE-hard for the setting of rectangular robots bound to translate in a rectangular workspace. Spirakis and Yap [6] showed that the labeled problem is NP-hard for disc robots in a simple polygon. Hearn and Demaine [2] improved the result of Hopcroft et al. by showing that the robots can be restricted to only two types—2×1 and 1×2 rectangles. Their work is more general: They introduced the *nondeterministic constraint logic* (NCL) model of computation, for which they describe several PSPACE-hard problems, and from which they derive hardness proofs for a variety of puzzle-like problems that consist of sliding game pieces. Hardness results for multi-robot motion planning are summarized in Table 1.

Recently, three polynomial-time algorithms for the unlabeled problem have been introduced. However, in one technique it is assumed that a certain portion of the free space, surrounding each start or target position, is *star-shaped* [7], while in the other two each pair of

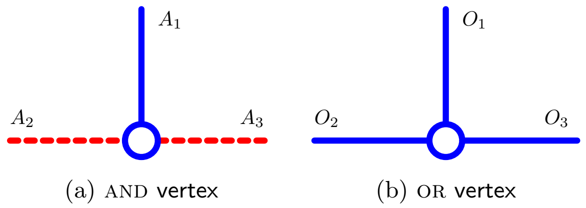


Figure 1: AND and OR vertices used in the NCL model. Red (dashed) edges have a weight of 1 and blue (solid) edges have a weight of 2. The blue (solid) vertex (circle) represents a minimum flow constraint of 2. In (a) A_1 can be directed outward if and only if A_2 and A_3 are both directed inward. In (b) O_1 can be directed outward if and only if any of the other two is directed inward.

start or target positions need to be *well separated* [1, 5]. A crucial question that follows from the latter work is whether the efficient solution of the problem is possible due to the separation constraints or the fact that the robots are unlabeled.

Contribution. In this paper we study the problem of unlabeled multi-robot motion planning for unit-square robots translating amidst polygonal obstacles. We show that deciding whether a solution exists is PSPACE-hard. To the best of our knowledge, this is the first hardness proof for the unlabeled case. Our construction also implies that three other variants of the unlabeled problem are PSPACE-hard, as well. A full version of this text, including all four results, can be found in [4].

2 Preliminaries

We are given a collection of polygonal obstacles and two sets of start and target configurations, $S = \{s_1, \dots, s_m\}$ and $T = \{t_1, \dots, t_m\}$, respectively. The unlabeled multi-robot motion-planning problem consists of translating unit-square robots from S to T such that each robot starts its motion in some $s \in S$ and ends it in some $t \in T$. We require that throughout the motion the robots will not collide with the obstacles[1], nor with each other. (Notice that by definition at the end of a successful motion each target is occupied by some robot.)

A key ingredient in our hardness proof for this problem is the NCL machine [2], which is defined by a *constraint* undirected graph $G = (V, E)$, a *weight* function, and a *minimum-flow* constraint. A machine configuration assigns orientations to all the edges of G, such that

*Blavatnic School of Computer Science, Tel-Aviv University, Israel. email: {kirilsol,danha}@post.tau.ac.il. This work has been supported in part by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski-Minerva Center for Geometry at Tel Aviv University.

[1]Robots may touch, but not penetrate, obstacles or each other.

Contributor	Problem	Complexity	Robots	Workspace
Hopcroft et al. [3]	labeled	PSPACE-hard	rectangles	rectangle
Spirakis, Yap [6]	labeled	strongly NP-hard	discs	simple polygon
Hearn, Demaine [2]	labeled	PSPACE-complete	1×2 rectangles	rectangle
this paper, [4]	unlabeled, labeled	PSPACE-hard	unit squares	polygonal obstacles

Table 1: Hardness results related to the multi-robot problem.

minimum-flow constraint of every vertex is satisfied. Hearn and Demaine [2] prove that deciding whether one machine configuration can be transformed into another, by a sequence of moves[2], is PSPACE-hard. Moreover, they show that this holds even if G consists only of AND and OR vertices (Figure 1).

3 From NCL to multi-robot motion planning

Given a constraint graph $G = (V, E)$ we construct a corresponding unlabeled scenario, which consists of unit-square robots and polygonal obstacles. We use a grid layout in which each cell functions as a placeholder for a gadget that represents and emulates a specific vertex of G. Between every two adjacent cells there is a doorway so that an interaction between adjacent gadgets can take place. When two vertices of G share an edge, the corresponding gadgets share a robot (see Figure 2).

For the vertices of V we create AND and OR gadgets, each having three exits through which they connect to other gadgets. Every gadget accommodates several robots and contains polygonal obstacles; the robots are drawn in orange, purple or green and the obstacles are drawn in gray. Robots are placed such that they neither overlap with the obstacles nor with each other. AND gadgets also have a point obstacle, illustrated in red. The following theorem summarizes our main result.

Theorem 1 *Given a set of start and target configurations S, T, respectively, where $|S| = |T| = m$, and a workspace $\mathcal{W}$ cluttered with polygonal obstacles, deciding whether there exists a set of m collision-free paths from S to T for m unlabeled unit-square robots is PSPACE-hard.*

4 Additional results

Our construction allows to show that three more variants of the unlabeled problem are PSPACE-hard [4]. In particular, even the following seemingly simpler problem is PSPACE-hard: The robots are initially placed in S and given some $t \in \mathcal{F}$ the problem is to decided whether *some* robot can reach t.

The construction is so crafted that it can be used for showing that the *labeled* problem for unit squares is PSPACE-hard as well. Specifically, given S and T a robot that is initially placed in $s \in S$ can reach at most one target position $t \in T$.

[2]A *move* consists of a reversal of the orientation of a single edge, while maintaining the minimum-flow constrains.

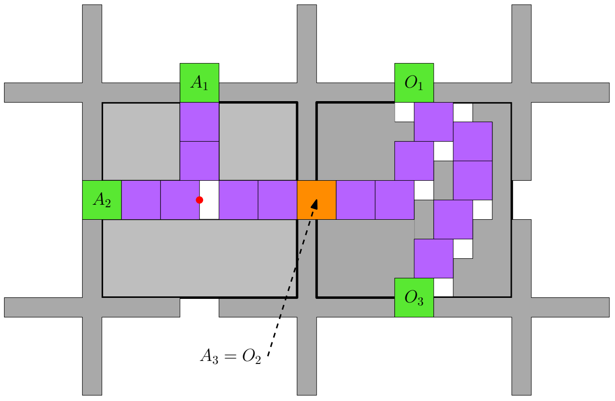


Figure 2: An example of an AND gadget (left) and OR gadget (right) that are connected by sharing a robot $A_3 = O_2$ (drawn in orange). In the AND gadget, A_1 can move a half-step down only if A_2 moves a half-step to the left, *and* A_3 stays put. In the OR gadget O_1 can move a half-step down only if O_2 moves a half-step to the left, *or* O_3 moves a half- step down.

References

[1] A. Adler, M. de Berg, D. Halperin, and K. Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. In *WAFR*, 2014.

[2] R. A. Hearn and E. D. Demaine. *Games, puzzles and computation.* A K Peters, 2009.

[3] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's problem". *IJRR*, 3(4):76–88, 1984.

[4] K. Solovey and D. Halperin. On the hardness of unlabeled multi-robot motion planning. In *RSS*, 2015. To appear.

[5] K. Solovey, J. Yu, O. Zamir, and D. Halperin. Motion planning for unlabeled discs with optimality guarantees. In *RSS*, 2015. To appear.

[6] P. G. Spirakis and C.-K. Yap. Strong NP-hardness of moving many discs. *Information Processing Letters*, 19(1):55–59, 1984.

[7] M. Turpin, N. Michael, and V. Kumar. Concurrent assignment and planning of trajectories for large teams of interchangeable robots. In *ICRA*, pages 842–848, 2013.

An Optimal Algorithm for Tiling the Plane with a Translated Polyomino

Andrew Winslow [*]

Abstract

We give a $O(n)$-time algorithm for determining whether translations of a polyomino with n edges can tile the plane. The algorithm is also a $O(n)$-time algorithm for enumerating all such tilings that are also regular, and we prove that at most $\Theta(n)$ such tilings exist.

1 Introduction

Motivated by applications in parallel computing, Shapiro [9] asked whether it could be decided if translations of a given polyomino could tile the plane. Beauquier and Nivat [1] proved that the problem was not only decidable, but solvable in polynomial time by testing a simple criterion called the *BN criterion*. Informally, a tile satisfies the BN criterion if it can be surrounded by instances of itself (see Figure 1). Such a surrounding corresponds to a *regular* or *isohedral* tiling where all tiles share an identical neighborhood.

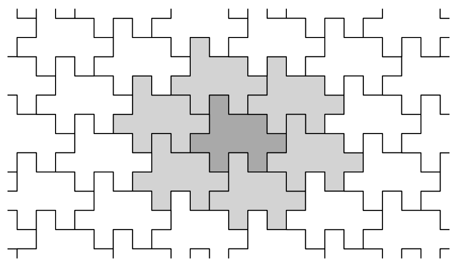


Figure 1: A polyomino tile (dark gray), a surrounding of the tile (gray), and the induced regular tiling (white).

Using a naive algorithm, the BN criterion can be applied to a polyomino with n edges in $O(n^4)$ time. Gambini and Vuillon [4] gave an improved $O(n^2)$-time algorithm and around the same time, Brlek, Provençal, and Fédou [2] achieved $O(n)$-time algorithms for two special cases: (1) the boundary contains no consecutive repeated sections larger than $O(\sqrt{n})$, and (2) testing a restricted version of the BN criterion (surroundable by just four instances). Provençal [8] further improved on the algorithm of Gambini and Vuillon for the general case, obtaining $O(n \log^3(n))$ running time, and in a recent survey, Blondin Massé, Brlek, and Labbé [7] conjecture that a $O(n)$-time algorithm exists. In this work, we confirm their conjecture by giving such an algorithm (Theorem 5).

Our algorithm also doubles as an algorithm for enumerating all surroundings (regular tilings) of the polyomino. As part of the proof of the running time of the algorithm, we prove a claim of Provençal [8] that the number of surroundings of a tile with itself is $O(n)$ (Theorem 3), complementing other tight bounds by Blondin Massé et al. [6] on a special class of surroundings.

2 Definitions

A *letter* is a symbol $x \in \{\mathbf{u}, \mathbf{d}, \mathbf{l}, \mathbf{r}\}$. The *complement* of a letter x, written $\bar{x}$, is defined by the bijection $\bar{\mathbf{u}} = \mathbf{d}$, $\bar{\mathbf{r}} = \mathbf{l}$, $\bar{\mathbf{d}} = \mathbf{u}$, and $\bar{\mathbf{l}} = \mathbf{r}$. A *word* W is a sequence of letters, and the ith letter of W is denoted $W[i]$. A *boundary word* is a word corresponding to the sequence of edge directions encountered during a clockwise traversal of a polyomino's boundary, e.g. the polyomino in Figure 1 has (circular) boundary word $\mathbf{uru^2rdr^2dr(dl)^2uldlul}$.

For a word W, $\widehat{W}$ denotes the sequence of complemented letters of W traversed in reverse order. A *factor of* W is a contiguous sequence X of letters in W. A factor X is *admissible* provided $W = XU\widehat{X}V$ with $|U| = |V|$, non-complementary first and last letters of U, and similarly of V. A *factorization* of W is a partition of W into consecutive (possibly length-0) factors F_1 through F_k, written $W = F_1F_2 \ldots F_k$.

3 BN Factorizations

Definition 1 *A factorization of a boundary word W is a* BN factorization *provided it is of the form $W = ABC\widehat{A}\widehat{B}\widehat{C}$.*

Lemma 1 *Let P be a polyomino with boundary word W. There exists a regular tiling of P with clockwise neighbor-shared boundary word factors $F_1, F_2, \ldots, F_k$ if and only if $F_1F_2 \ldots F_k$ is a BN factorization.*

Observe that this mapping between between tilings and factorizations is a bijection.

Lemma 2 *A boundary word W has $O(|W|)$ BN factorizations.*

[*]Université Libre de Bruxelles, 1050 Bruxelles, Belgium, `andrew.winslow@ulb.ac.be`

Theorem 3 *A polyomino with n sides has $O(n)$ regular tilings.*

Provençal [8] observed that polyominoes with $\Omega(n)$ tilings exist, e.g. $\mathbf{ur}^i\mathbf{dl}^i$ with $i \geq 1$ has i regular tilings.

4 An Algorithm for Enumerating Factorizations

Lemma 4 *Let W be a polyomino boundary word. The BN factorizations of W can be enumerated in $O(|W|)$ time.*

Proof. Corollary 5 of [2] states that all factors of a BN factorization are admissible. The algorithm first computes all admissible factors, then enumerates factorizations consisting of them.

Computing admissible factors. Corollary 5 of [2] implies that there are at most $2|W|$ admissible factors, since admissible factor has a distinct center. For each center $W[i]$ or $W[i]W[i+1]$, the admissible factor with this center is LR, where R is the longest common prefix of WW starting at $WW[i+1]$ and $\widehat{W}\widehat{W}$ starting at $\widehat{W}\widehat{W}[|W|/2-(i+1)]$. Use a suffix-tree-based approach (see Theorem 9.1.1 of [5]) to preprocess these words in $O(|W|)$ time so that the longest common prefixes can be computed in $O(1)$ time each and $O(|W|)$ total time. The word L is defined and computed similarly.

Enumerating factorizations. Let $W = AY\widehat{A}Z$ with A an admissible factor and $|Y| = |Z|$. Let $B_1, B_2, \ldots, B_l$ be the admissible prefix factors of Y, with $|B_1| < |B_2| < \cdots < |B_l|$. Similarly, let $C_1, \ldots, C_m$ be the suffix factors with $|C_1| < \cdots < |C_m|$. A variation of Lemma C4 of [3] (omitted due to space) implies that for fixed A, there exist intervals $[b, l]$, $[c, m]$ such that the BN factorizations $AB_iC_j\widehat{A}\widehat{B_i}\widehat{C_j}$ are exactly those with $i \in [b, l]$ or $j \in [c, m]$.

First, construct a list of all admissible factors starting at each $W[k]$, sorted by length in $O(|W|)$ time using counting sort. Repeat for factors ending at each $W[k]$.

Next, use a two-finger scan to find, for each factor A that ends at $W[k]$, the longest factor B_l starting at $W[k+1]$ such that $|A| + |B_l| \leq |W|/2$. Then check whether C_j, the factor following B_l such that $|AB_lC_j| = |W|/2$, is admissible and report the factorization $AB_lC_j\widehat{A}\widehat{B_l}\widehat{C_j}$ if so. Checking whether C_j is admissible takes $O(1)$ time using an array mapping each center to the unique admissible factor with this center.

Enumerated additional BN factorizations containing A by checking factors B_i with $i = l-1, l-2, \ldots$ for an admissible following factor C_j. If C_j is admissible, report the factorization, otherwise stop the iteration, since $i = b-1$.

Finally, use a similar two-finger scan to find, for each factor A that starts at $W[k]$, the longest factor C_m that ends at $W[k+|W|/2-1]$ such that $|A| + |C_m| \leq |W|/2$, check whether B_i preceeding C_m such that $|AB_iC_m| = |W|/2$ is admissible, and report the possible BN factorization. Then check and report similar factorizations with C_j for $j = m-1, m-2, \ldots$ until $j = c-1$.

In total, the two-finger scans take $O(|W|)$ time plus $O(1)$ time to report each factorization. Each factorization is reported once per choice of A. Remove duplicate factorizations with a canonical factor labeling (A contains $W[0]$) and radix sorting the six-tuples of the first letter indices of the factors. Then by Lemma 2, reporting factorizations also takes $O(|W|)$ time. $\square$

Theorem 5 *Let P be a polyomino with n sides. The regular tilings of P can be enumerated in $O(n)$ time.*

Acknowledgments

The author would like to thank Stefan Langerman for fruitful discussions that greatly improved the paper.

References

[1] D. Beauquier and M. Nivat. On translating one polyomino to tile the plane. *Discrete & Computational Geometry*, 6:575–592, 1991.

[2] S. Brlek, J.-M. X. Provençal, and Fédou. On the tiling by translation problem. *Discrete Applied Mathematics*, 157:464–475, 2009.

[3] Z. Galil and J. Seiferas. A linear-time on-line recognition algorithm for "Palstar". *Journal of the ACM*, 25(1):102–111, 1978.

[4] L. Gambini and L. Vuillon. An algorithm for deciding if a polyomino tiles the plane by translations. *RAIRO - Theoretical Informatics and Applications*, 41(2):147–155, 2007.

[5] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[6] A. B. Massé, S. Brlek, A. Garon, and S. Labbé. Every polyomino yields at most two square tilings. In *7th International Conference on Lattice Paths and Applications*, pages 57–61, 2010.

[7] A. B. Masseé, S. Brlek, and S. Labbé. Combinatorial aspects of Escher tilings. In *22nd International Conference on Formal Power Series and Algebraic Combinatorics*, pages 533–544, 2010.

[8] X. Provençal. *Combinatoire des mots, géométrie discrète et pavages.* PhD thesis, Université du Québec à Montréal, 2008.

[9] H. D. Shapiro. Theoretical limitations on the efficient use of parallel memories. *IEEE Transactions on Computers*, 27(5):421–428, 1978.