Randomness and Hardness in Geometric Computing Problems

by

Sachio TERAMOTO

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Dr. Tetsuo Asano

School of Information Science
Japan Advanced Institute of Science and Technology

March 2007

Abstract

In this dissertation, geometric optimization problems ensuring a kind of randomness (such as uniformity, irregularity, or random generation) as its objective and designing practical strategies are widely studied. We mainly consider how to define a good uniformity in geometric dispersion to be as irregularly as possible, how to generate efficiently geometric structures (e.g., simple polygons, plane triangulations, etc.) at random, and how difficult to achieve randomness underlying uniformity criterion. For each question, the occasion of investigations has been derived from both of theoretical and engineering aspects.

With wide-ranging advances in discrete and computational geometry, geometric computing, especially, which manipulates spatial data being structured combinatorially and/or geometrically, has become a pervasive and increasingly critical aspect in every corner of science and engineering. A lot of applications have been requiring practical algorithms for geometric optimization problems. When we deal with geometric optimization problems which are NP-hard, we often prefer to efficiently pick a good solution instead of the optimal from a solution space spread by relaxing the objective function. It is important to present a better relaxation to provid an appropriate solution space, rather to analyze an approximative solution space to show that any cost of the relaxed objective function is not so inferior, no matter what we pick up any from the space.

There exist various approximated and randomized paradigms in a comprehensive manner, such as concepts of random sampling. However, sampling based approaches may not be applied to practical applications directly, since eventual approaches require impractical large sampling set. Therefore, we have to reconsider geometric configurations more carefully, so that we can present useful and helpful frameworks, getting insight into essential difficulties. In fact, we think it is worth noting that we investigate for good relaxation criteria and techniques for randomly generating geometric structures, in order to sophisticate sampling techniques. In addition, analytical study for the existence of equilibrium/disequilibrium configuration on spatial competitions is also important. Considering a game model for competitive facility location, we show a case that each player will be competing to locate his/her facilities at equilibrium positions, when there exists a winning strategy.

Acknowledgments

First of all, I would like to express my sincere gratitude to my principal adviser Professor Tetsuo Asano of Japan Advanced Institute of Science and Technology for his academic advice and kind guidance during this work. His persistent encouragement and support were really helpful, and his way of looking at problems, way of presenting materials, and everything were very exciting to me. He has had a profound influence throughout my academic career. At the most basic level, he introduced me to the exciting subject of computational geometry, and provided key insights and direction on the research side; problem-solving techniques, publications, collaborations, and academic politics. Especially, he provided me with experience of meeting to many advanced research topics and great researchers who work worldwide and actively in the field of theoretical computer science. He also gave me some jobs as assistant and the pay was helpful. Again, I show my gratitude to my supervisor.

I would like to thank my adviser Associate Professor Ryuhei Uehara of Japan Advanced Institute of Science and Technology for his helpful suggestions, encouragements, and friendliness. He always allowed me to make remarks somewhat puerile or nonsense idea, and made some of them into interesting research themes with fruitful and conscientious discussions.

I would like to express my gratitude to Professor Mineo Kaneko who kindly admitted to be minor-research adviser, for helpful suggestions and encouragements.

I am no less grateful to the following people for their excellent comments and substantial supports: Professor Koji Nakano of Hiroshima University, Associate Professor Koji Obokata of Ichinoseki National College of Technology, Assistant Professor Arijit Bishnu of Indian Institute of Technology, and Associate Mitsuo Motoki & Masashi Kiyomi of Japan Advanced Institute of Science and Technology.

Some of chapters in the thesis are based on joint papers with the following collaborators: Erik D. Demaine of Massachusetts Institute of Technology, Benjamin Doerr of Max Planck Institute für Informatik, and Naoki Katoh of Kyoto University. I would like to thank them for many enlightening discussions.

I am grateful to all who have affected or suggested my areas of research, and thanks some of my fellow and colleagues: Eishi Chiba, Yasuyuki Kawamura, Shinji Sasahara,

Taisuke Shimamoto, and Xuefeng Liang. They each helped make my time in the PhD program more fun and interesting.

Finally, I deeply thank my family for their love, patience, and encouragement, and for all that they have done for my sake; this work is dedicated to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this dissertation, geometric optimization problems ensuring a kind of randomness (such as uniformity, irregularity, or random generation) as its objective and designing practical strategies are widely studied. We mainly consider how to define a good uniformity in geometric dispersion to be as irregularly as possible, how to generate efficiently geometric structures (e.g., simple polygons, plane triangulations, etc.) at random, and how difficult to achieve randomness underlying uniformity criterion. For each question, the occasion of investigations has been derived from both of theoretical and engineering aspects.

The technical term "*Randomness*" in the thesis to be distinguished from which is used in randomized algorithm [93], or statistical optimization [79].

With wide-ranging advances in discrete and computational geometry, geometric computing, especially, which manipulates spatial data being structured combinatorially and/or geometrically, has become a pervasive and increasingly critical aspect in every corner of science and engineering. A lot of applications have been requiring practical algorithms for geometric optimization problems. When we deal with geometric optimization problems which are NP-hard, we often prefer to efficiently pick a good solution instead of the optimal from a solution space spread by relaxing the objective function. It is important to present a better relaxation to provid an appropriate solution space, rather to analyze a approximative solution space to show that any cost of the relaxed objective function is not so inferior, no matter what we pick up any from the space.

There exist various approximated and randomized paradigms in a comprehensive manner, such as concepts of random sampling. However, sampling based approaches may not be applied to practical applications directly, since eventual approaches require impractical large sampling set. Therefore, we have to reconsider geometric configurations more carefully, so that we can present useful and helpful frameworks, getting insight into essential difficulties. In fact, we think it is worth noting that we investigate for good

relaxation criteria and techniques for randomly generating geometric structures, in order to sophisticate sampling techniques. In addition, analytical study for the existence of equilibrium/disequilibrium configuration on spatial competitions is also important. Considering a game model for competitive facility location, we show a case that each player will be competing to locate his/her facilities at equilibrium positions, when there exists a winning strategy.

1.1 Summary and Organization

This dissertation tries to identify some nice structural randomness for a number of combinatorial or geometric objects. It consists of three rather independent parts.

1.1.1 Part I

In Part I, we consider the problem in which a generalization of the point arranging problem, namely its on-line version. We want to insert n points one by one in such a way that uniformity is achieved at every insertion of a point. Since the solutions for the off-line point arranging problems are different for different values of n, it would be impossible to derive good point sequences from such optimal solutions even if they were available. It should also be noted that a subsequence of an optimal point sequence is not optimal. Therefore, we cannot hope for an incremental algorithm constructing optimal point sequences.

It is not straightforward to define uniformity of points. The minimum pairwise distance is not good to measure the uniformity of points as it does not reflect large empty areas. We could also borrow a measure from discrepancy theory [26, 86]. Here we take a simple geometric shape R and count how many points are contained in R while moving R all over the unit cube. The uniformity is measured by the difference between the largest and smallest counts for all possible sizes of the shape. A serious disadvantage of the measure is computational hardness. Also, it is not clear what shape R to use.

We define uniformity of point distribution using not only closest point pairs but also largest empty circles. Our criterion is to minimize the gap ratio, which is the maximum gap (diameter of a largest empty circle) over the minimum gap (the minimum pairwise distance). Note that this definition is extendible to higher dimensions since those gaps can be defined in any dimension.

We present a linear time algorithm for finding an optimal n-point sequence with the maximum gap ratio bounded by $2^{\lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1)}$ in the 1-dimensional case. We describe

how hard analytically the same problem is for a point set in the plane and propose a local search heuristics for finding a good solution.

1.1.2 Part II

In Part II, we consider geometric random generation problems for generating simple polygons and plane triangulations at random, which are well-known outstanding open problems. Firstly, we study the simple polygonalization problem; given a set S of points in the plane, randomly generate a simple polygon with n sides using the points of S as its vertices, or compute a simple polygonalization of S. More precisely, we would like polygonalize a set S of points so that each member of all polygonalizations of S is generated uniformly at random.

However, examining the set of all simple polygonalizations is quite difficult even in the counting problem which asks how many simple polygonalizations are there in S. The currently known approximate upper and lower bounds are $O(86.81^n)$ and $\Omega(4.642^n)$, respectively. Due to the high upper bound, heuristic approaches have been adopted to generate a simple polygonalization during the past decade.

We propose a triangulation-base heuristic algorithm which generates each possible simple polygon with a positive probability in $O(n^2)$ time. This improves the time complexity $O(n^4 \log n)$ of the best known heuristic algorithm. Our algorithm consists of three phases: first, it generates a triangulation T of given point set S at random; next computes a random maximal polygon tree on the dual $\mathcal{D}(T)$. A polygon tree $\mathcal{T}$ on a dual $\mathcal{D}(T)$ of a triangulation T is a tree such that $\cup_{v \in \mathcal{T}} g^{-1}(v)$ is a simple polygon, where g is bijection from a face in T to a vertex in $\mathcal{D}(T)$; and finally constructs a simple polygon by traversing on the maximal polygon tree. In addition, we experiment our heuristic algorithm for showing efficiency and usefulness.

1.1.3 Part III

In Part III, we consider examining the existence of equilibrium configurations in a spatial competitions on a finite graph. Therefore, we introduce a two-person game model, say Voronoi game, for the competitive facility location, and investigate actual facility locations obtained by resulting a game in which both players do their best, whether the configuration have some equilibrium/disequilibrium property, or not. Furthermore, the computational difficulties also are described.

The Voronoi game is played on continuous domain, and only two special cases (the 1-dimensional cases and the 1-round case) are well investigated. We introduce the discrete

Voronoi game in which the game arena is given as a graph. We first show the best strategy when the game arena is a large complete k-ary tree. Next we show that the discrete Voronoi game is intractable in general. Even in the 1-round case in which place occupied by the fist player is fixed, the game is NP-complete in general. We also show that the game is PSPACE-complete in general case.

Chapter 2

Randomness and Hardness in Geometric Dispersion

2.1 Introduction

The circle packing problem is to place n equal and non-overlapping circles in a unit square. It is one of the most important geometric optimization problems with a number of applications and has been intensively investigated [33, 96, 97, 98]. It is well known that the circle packing problem is equivalent to arranging n points in a unit square in such a way that the minimum pairwise distance is maximized. This problem seems to be computationally hard. In fact, no optimal solution is known for relatively large values of n, say $n > 100$.

The problem considered in this paper is a generalization of the point arranging problem, namely its online version. We want to insert n points one by one in such a way that uniformity is achieved at every insertion of a point. Since the off-line point arranging problem has different solutions for different values of n, it would be impossible to derive good point sequences from such optimal solutions even if they were available. It should also be noted that a subsequence of an optimal point sequence is not optimal. Therefore, we cannot hope for an incremental algorithm constructing optimal point sequences.

It is not straightforward to define uniformity of points. The minimum pairwise distance is not good to measure the uniformity of points as it does not reflect large empty areas. We could also borrow a measure from discrepancy theory [26, 86]. Here we take a simple geometric shape R and count how many points are contained in R while moving R all over the unit cube. The uniformity is measured by the difference between the largest and smallest counts for all possible sizes of the shape. A serious disadvantage of the measure is computational hardness. Also, it is not clear what shape R to use.

We define uniformity of point distribution using not only closest point pairs but also largest empty circles. Our criterion is to minimize the gap ratio, which is the maximum gap (diameter of a largest empty circle) over the minimum gap (the minimum pairwise distance). Note that this definition is extendible to higher dimensions since those gaps can be defined in any dimension.

This problem is closely related to an industrial application on digital halftoning, which is a technique to convert continuous-tone images into binary images for printing. One of the most popular methods for halftoning is Dithering, which binarizes an images using a threshold matrix called the dither matrix. The quality of output images heavily depends on this matrix. A target is an $n \times n$ matrix containing integers from 0 through n^2-1 in such a way that elements up to i are uniformly distributed for each $i = 1, 2, \ldots, n^2-1$. Such a matrix is similar to the dither matrix called the blue-noise mask [95, 123]. Combinatorial approaches are also found for the problem, see e.g., [10, 12, 13, 14, 44, 66, 112].

2.1.1 Problem Statement

Let $\mathbb{S}^d = [0,1]^d$ be the unit cube in the d-dimensional space $\mathbb{R}^d$ and $P = (p_1, \ldots, p_n)$ be an n-point sequence contained in $\mathbb{S}^d$. We insert $p_1, p_2, \ldots, p_n$ in this order. For each $i = 1, 2, \ldots, n$, we define a subsequence P_i of P by its first i points, i.e., $P_i = (p_1, \ldots, p_i)$. With P_i, we associate a point set $S_i := \{p_1, \ldots, p_i\} \cup S_0$, where S_0 is the set of the 2^d corner points of $\mathbb{S}^d$. The smallest among all pairwise distances in S_i is *the minimum gap*

$$g_i \;:=\; \min_{p,q \in S_i,\ p \neq q} d(p,\ q),$$

where $d(p, q)$ is the Euclidean distance between two points p and q.

The maximum gap is defined via the largest empty circle. An empty circle is a circle whose center is located within the unit cube and contains no point of the set S_i. The diameter of the largest empty circle for the set S_i is *the maximum gap* G_i

$$G_i \;:=\; \max_{p \in \mathbb{S}^d} \min_{q \in S_i} 2d(p,\ q)$$

Note that the point p in the definition above is an arbitrary point in the unit cube. Now we define *the i-th gap ratio* by

$$r_i \;:=\; G_i / g_i.$$

For a point sequence P, we define *the maximum gap ratio* as $R_P := \max_{i=1,\ldots,n} r_i$. For a fixed integer n, we denote R_n the optimal gap ratio for any n-point sequence:

$$R_n := \min\ \{\ R_P \mid P \text{ is an } n\text{-point sequence in } \mathbb{S}^d\}.$$

Given d and n, we want to find an n-point sequence P in $\mathbb{S}^d$ that achieves the optimal (=minimal) gap ratio R_n. More formally, our problem is described as follows.

Problem 1:
Input: Integers d and n.
Ensured: Compute an optimal n-point sequence P in $\mathbb{S}^d$ that achieves the optimal gap ratio R_n for n points.

2.1.2 Our contribution

We start with a simple greedy algorithm called *incremental Voronoi insertion* for points in the plane in Section 2.2. The Voronoi insertion generates a point sequence P with $R_P \leq 2$. It is also easy to extend this algorithm to higher dimensions while keeping the gap ratio 2.

In Section 2.3, we give a linear time algorithm that constructs an n-point sequence with maximum gap ratio bounded by $2^{\lfloor n/2 \rfloor/(\lfloor n/2 \rfloor+1)}$ in the 1-dimensional unit cube $\mathbb{S}^1$. We also show that the bound is optimal, which fully shows the 1-dimensional case.

Section 2.4 deals with the 2-dimensional case again. It looks quite challenging to find an optimal point sequence even for rather small values of n. Therefore, we give two simple heuristic algorithms finding a point sequence with maximum gap ratio smaller than that of the point sequence generated by the incremental Voronoi insertion. Since the Voronoi insertion gives an upper bound, our next goal is the following:

Problem 2:
Input: Integers d and n.
Output: An n-point sequence P in $\mathbb{S}^d$ such that $R_P < 2$.

We have implemented our heuristic algorithm to find point sequences whose maximum gap ratios are strictly less than 2, which is achieved by the Voronoi insertion. Some such sequences are given together with related statistics on our experiments.

2.2 Simple Greedy Algorithm – Voronoi insertion

We start with a simple greedy algorithm for inserting points uniformly. In this algorithm, we maintain a Voronoi diagram for a set of points which have already been inserted and its intersection with the boundary of the unit cube. Voronoi vertices and the intersections between Voronoi edges and cube surface are candidates for the next point to be inserted. We evaluate each such vertex by the distance to its nearest point (site) and choose the

one of the largest such distance as the next point to be inserted. This is why we call it incremental Voronoi insertion.

Define a point set $S_i^d = \{(x_1, x_2, \ldots, x_d) \mid$ exactly i coordinates are either 0 or 1 and the remaining coordinates are $1/2\}$ for $i < d$. For example, we have

$$\begin{aligned} S_0^3 &= \{(1/2, 1/2, 1/2)\}, \\ S_1^3 &= \{(*, 1/2, 1/2), (1/2, *, 1/2), (1/2, 1/2, *)\}, \\ S_2^3 &= \{(*, *, 1/2), (*, 1/2, *), (1/2, *, *)\}, \end{aligned}$$

where $*$ indicates 0 or 1, that is, $(*, 1/2, 1/2)$ represents $(0, 1/2, 1/2), (1, 1/2, 1/2)$.

The first point to be inserted must be the unique element of S_0^d, i.e., $(1/2, 1/2, \ldots, 1/2)$. Then, we insert points in the set S_1^d one by one, and continue to points in $S_2^d, S_3^d, \ldots, S_{d-1}^d$. Suppose we have inserted all the points in $S_0^d, S_1^d, \ldots, S_{d-2}^d$ and we are now going to insert $p_j = (0, 0, \ldots, 0, 1/2)$, the first point in the set S_{d-1}^d. The point p_j is the mid-point of a cube edge by the definition. Thus, the minimum pairwise distance is $1/2$, that is, the minimum gap is $1/2$. Since this is the first point located on a cube edge, the empty ball centered at the next point $(0, 0, \ldots, 0, 1, 1/2)$ that passes through the two points $(0, 0, \ldots, 0, 0)$ and $(0, 0, \ldots, 0, 1)$ remains empty. In fact, this ball is the largest empty ball. Its diameter is obviously 1. Therefore, the ratio is exactly 2 after the point.

We can also show that the maximum ratio before inserting this point is less than 2 and it remains so until the very last point of S_{d-1}^d. When we have inserted all the points in $S_0^d, S_1^d, \ldots, S_{d-1}^d$, we can continue the same process again for 2^d sub-cubes in a recursive fashion. Thus, we can conclude that the above-mentioned approximation algorithm achieves the maximum ratio 2.

2.3 1-dimensional problem

Our domain here is a unit interval $[0, 1]$. The two extremal points 0 and 1 are assumed to be placed in advance. We present a simple linear time strategy better than the incremental Voronoi insertion. Moreover, we show that the strategy is in fact optimal.

2.3.1 Lower bound on R_n

We first estimate the lower bound of R_n for an n-point sequence. Let $P = (p_1, p_2, \ldots, p_n)$ be a finite sequence of n points in the unit interval $[0, 1]$ such that $p_i \neq p_j$ whenever $i \neq j$. For $i = 0, \ldots, n$, the points $p_1, \ldots, p_i$ partition the unit interval into $i + 1$ intervals of lengths $m_1^i, m_2^i, \ldots, m_{i+1}^i$. Without loss of generality we may assume that $m_j^i \geq m_{j+1}^i$ for

all i, $0 \leq i \leq n$ and $j, 1 \leq j \leq i$. Then, the maximum and minimum gaps are given by m_1^i and m_{i+1}^i, respectively. Hence, the ratio R_P for the sequence P is

$$R_P := \max_{1 \leq i \leq n} \frac{m_1^i}{m_{i+1}^i} \tag{2.1}$$

Put $M^i = \{m_1^i, \ldots, m_{i+1}^i\}$ and regard it as a multi-set (i.e., it may contain elements more than once). Clearly, M^{i+1} is obtained from M^i by replacing one element from M^i by two which add up to the first one. The following lemma states that if $R_P \leq 2$, then this replaced element is always the largest.

Lemma 2.3.1 *If $R_P \leq 2$, then for each $i = 0, \ldots, n-1$ there are $a, b \in [0,1]$ such that $m_1^i = a+b$ and $M^{i+1} = \{m_2^i, \ldots, m_{i+1}^i, a, b\}$ (as multi-set) and one of a and b is a smallest element of M^{i+1}.*

Proof Assume that $M^{i+1} = M^i \setminus \{m_j^i\} \cup \{a, b\}$ for some $j, 1 \leq j \leq i+1$ such that $m_j^i < m_1^i$ and $a + b = m_j^i$. W.l.o.g., let $b \leq a$. Then, $b \leq \frac{1}{2} m_j^i < \frac{1}{2} m_1^i$ and hence $R_P \geq m_1^{i+1}/b = m_1^i/b > 2$. If both a and b are greater than m_{i+1}^i, then again $R_P \geq m_1^i/m_{i+1}^i = (a+b)/m_{i+1}^i > 2m_{i+1}^i/m_{i+1}^i = 2$. □

Note, however, that a priori we do not know that both a and b are not larger than m_{i+1}^i.

Lemma 2.3.2 *Given an integer $n \geq 1$, the lower bound of R_n is $2^{\lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1)}$.*

Proof Assume that $R_P \leq 2$ for an n-point sequence. Let first n be even. Let $j, 1 \leq j \leq \frac{n}{2} + 1$ be such that $m_j^{n/2} \in M^n$. Such a j exists, since at most $n/2$ of the elements in $M^{n/2}$ are replaced in the sequel from $M^{n/2}$ to M^n. We have

$$\frac{m_1^{n/2}}{m_j^{n/2}} \leq \frac{m_1^{n/2}}{m_{n/2+1}^{n/2}} \leq R_P.$$

Also, for each $n/2 \leq i \leq n-1$, we have $R_P \geq m_1^{i+1}/m_{i+2}^{i+1} \geq m_1^{i+1}/m_{j/2}^i$ by Lemma 2.3.1. Since $m_j^{n/2} \in M^n$,

$$R_P \geq \frac{m_1^{n/2}}{m_j^{n/2}} \geq \frac{m_1^{n/2}}{m_1^n} = \prod_{i=n/2}^{n-1} \frac{m_1^i}{m_1^{i+1}} = \prod_{i=n/2}^{n-1} \frac{m_1^i}{m_{i+2}^{i+1}} \Big/ \frac{m_1^{i+1}}{m_{i+2}^{i+1}} \geq \left(\frac{2}{R_P}\right)^{n/2}.$$

We conclude $R_P \geq 2^{(n/2)/(n/2+1)}$. For n odd, let $P' = (p_1, \ldots, p_{n-1})$. Then, $R_P \geq R_{P'}$ by definition and $R_{P'} \geq 2^{\lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1)}$ by the above. So, $R_n \leq 2^{\lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1)}$. This completes the proof of Lemma 2.3.2. □

So, we have obtained the lower bound of R_n for n-point sequences. Now, what remains is to give an algorithm for computing an optimal point sequence P^*.

Algorithm 1: A naive strategy

Calculate $r = 2^{\lfloor n/2 \rfloor/(\lfloor n/2 \rfloor+1)}$;

$p_1 = 1/(1+r)$;

for $i = 1$ *to* $n-2$ **do**

  Let m_1^i and m_2^i be the current longest and second longest intervals, respectively;

  Put a point p_{i+1} into m_1^i to partition it into two subintervals a and b so that $m_2^i / \min\{a, b\} = r$;

Put the last point p_n so as to partition the current longest interval into two intervals of the same lengths;

First, consider the following algorithm (Algorithm 1) suggested in the lower bound proof.

This strategy always puts a point p_i so that the gap ratio is equal to $2^{\lfloor n/2 \rfloor/(\lfloor n/2 \rfloor+1)}$ for each i. If it is possible then the sequence obtained is optimal since its bound coincides with the lower bound. The strategy implicitly assumes that the smaller one of the new subintervals has the minimum length among current intervals. Unfortunately, it is impossible to keep the ratio. The reason is as follows. Let a_i and b_i $(a_i > b_i)$ be new subintervals resulting after the i-th insertion. Then, $M^1 = \{a_1, b_1\} = \{\frac{r}{r+1}, \frac{1}{r+1}\}$ and $M^2 = \{b_1, a_2, b_2\} = \{\frac{1}{r+1}, \frac{r-1}{r}, \frac{1}{r(r+1)}\}$. We insert p_3 into M^2. Note that the maximum interval length in M^2 depends on the number of points to be inserted. If $b_1 \geq a_2$, (the case of $r \leq \frac{1+\sqrt{5}}{2}$), then $b_3 = \frac{a_2}{r} = \frac{r-1}{r^2}$ and $a_3 = \frac{1}{r+1} - b_3 = \frac{1}{r^2(r+1)}$. Since $a_3 < b_3$ for $r > \sqrt{2}$, $r_3 = a_2/a_3 = r(r^2-1) > R_n$. This suggests that if n is large enough, say $n > 3$, the assumption of above strategy does not hold. On the other hand, if $b_1 < a_2$ (the case of $r > \frac{1+\sqrt{5}}{2}$), then $r_2 = \frac{a_2}{b_2} = r^2 - 1$, and $2 < R_n^2 - 1$ for $n \geq 8$. Therefore, we cannot obtain an optimal point sequence P^* by the above strategy.

Observation 2.3.3 *Gap ratios for the first $n-1$ points should be strictly less than $2^{\lfloor n/2 \rfloor/(\lfloor n/2 \rfloor+1)}$, and moreover, these ratios are never determined until the last interval is fixed.*

This Observation 2.3.3 suggests that an optimal point sequence of length n should be determined in a bottom-up fashion, that is, from the last interval to the first one.

2.3.2 An optimal point insertion strategy

A rough sketch of our strategy is as follows. Let $(p_1, p_2, \ldots, p_n)$ be a point sequence to be inserted in the unit interval $x_1 = [0, 1]$. We maintain all intervals generated during n insertions, and we denote by x_j the interval induced by the p_{j-1}. Hereafter, we denote

the j-th interval by x_j, and unify x_j and its length $|x_j|$. Each point $p_i, i = 1, \ldots, n$, is inserted into the current largest interval x_i to split it into two new subintervals x_{2i} and x_{2i+1} with $x_{2i} + x_{2i+1} = x_i$. An important observation here is that we can determine the point p_i so that it results in a sorted sequence $(x_{i+1}, x_{i+2}, \ldots, x_{2i}, x_{2i+1})$ of intervals in the non-increasing order of their lengths. The process is terminated when the last point p_n is inserted to have a sequence $(x_{n+1}, x_{n+2}, \ldots, x_{2n+1})$.

Now, let us describe how to determine the point sequence. It is divided into two subsequences at $k = \lfloor n/2 \rfloor$. For the first half $(p_1, \ldots, p_k)$, the current longest interval x_i is unevenly partitioned into the new two subintervals x_{2i} and x_{2i+1}, so that $x_{2i} > x_{2i+1}$ and $x_i = x_{2i} + x_{2i+1}$. Since we are trying to achieve a ratio strictly less than 2, the ratio x_{i+1}/x_{2i+1} must be strictly less than 2. For the remaining points $(p_{k+1}, \ldots, p_n)$, the current longest interval x_i is partitioned evenly into two new subintervals x_{2j} and x_{2j+1} so that $x_{2j} = x_{2j+1} = x_j/2$ and x_{j+1}/x_{2j+1} is strictly less than 2, or equal to R_n. This is because the intervals x_{2i} and $x_{2i+1}, i = k+1, \ldots n$, will never be subdivided during the remaining insertion. Since minimum gaps are maximized by evenly partitioning, it minimizes the maximum gap ratios.

More concretely, we first compute the target ratio $R_n = 2^{k/(k+1)}$ where $k = \lfloor n/2 \rfloor$, and a magic number $y_1 = (2^{l-k} + 2\sum_{i=2}^{k+1} \frac{R_n^{i-1}}{2^{i-1}})^{-1}$, where $l = \lceil n/2 \rceil$. Then, we fix the last $2k+2$ intervals;

$$
\begin{aligned}
x_{2l} &= x_{2l+1} = y_1 && \text{if } n \text{ is odd,} \\
&\quad x_{2l+1} = y_1 && \text{if } n \text{ is even,} \\
x_{2(l+1)} &= x_{2(l+1)+1} = \frac{R_n}{2} y_1, \\
x_{2(l+2)} &= x_{2(l+2)+1} = \left(\frac{R_n}{2}\right)^2 y_1, \\
&\vdots \\
x_{2(l+k)} &= x_{2(l+k)+1} = \left(\frac{R_n}{2}\right)^k y_1.
\end{aligned}
$$

The remaining intervals can be determined so that $x_i = x_{2i} + x_{2i+1},\ i = k, k-1, \ldots, 2, 1$. This strategy can be summarized in the following pseudo code.

2.3.3 Configuration tree

Before showing the optimality and correctness of our strategy, we introduce a *configuration tree* to simplify the arguments for the proof. The tree describes how intervals are generated. Initially it consists of a root corresponding to the unit segment (or interval) x_1.

Algorithm 2: An optimal strategy

input : An integer $n > 0$.

output: An optimal point sequence P, i.e., $R_P = R_n$.

1 $R_n = 2^{\lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1)}$;

2 $y_1 = \left(2^{\lceil n/2 \rceil - \lfloor n/2 \rfloor} + 2\sum_{i=2}^{\lfloor n/2 \rfloor + 1} \frac{{R_n}^{i-1}}{2^{i-1}}\right)^{-1}$;

3 **if** *n is odd* **then** $x_{2\lceil n/2 \rceil} = x_{2\lceil n/2 \rceil + 1} = y_1$;

4 **else** $x_{2\lceil n/2 \rceil + 1} = y_1$;

5 **for** $i = 1$ **to** $\lfloor n/2 \rfloor$ **do** $x_{2(\lceil n/2 \rceil + i)} = x_{2(\lceil n/2 \rceil + i} = \left(\frac{R_n}{2}\right)^i \cdot y_1$;

6 **for** $i = \lfloor n/2 \rfloor$ **downto** 1 **do** $x_i = x_{2i} + x_{2i+1}$;

7 Compute a point sequence P from the interval sequence $(x_{i+1}, x_{i+2}, \ldots, x_{2i}, x_{2i+1})$;

When an interval x_i is partitioned into two subintervals x_{2i} and x_{2i+1}, two corresponding nodes are created as children of the node for the interval x_i. Then, a set of internal nodes are those for $x_1, x_2, \ldots, x_i$ and the remaining nodes for $x_{i+1}, \ldots, x_{2i+1}$ are leaf nodes of the tree, which form an interval sequence $(x_{i+1}, \ldots, x_{2i+1})$ in the order of their appearance. Figure 2.1 shows an example of the configuration tree for $n = 4$. The shaded nodes are leaf nodes. We can see how the intervals corresponding to leaf nodes subdivide the unit segment.

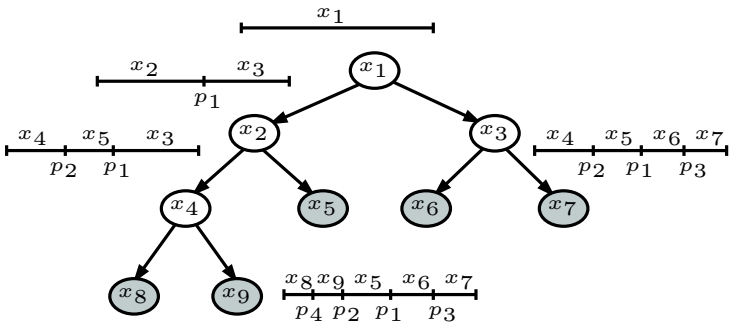


Figure 2.1: Configuration tree for a point sequence $P = (p_1, p_2, p_3, p_4)$.

The above strategy constructs a configuration tree, and a partition of the unit segment is obtained. So, each interval length corresponding a leaf node is calculated using the magic number y_1. Since each internal node has exactly two children and both intervals are known, all interval lengths are determined successively from bottom to top (root).

As an exercise, let us consider the case when we insert 5 points. Unlike the incremental Voronoi insertion, we put the first point p_1 so that the unit interval is split unevenly. Then, we put the second point p_2 to split the longer interval. Now we split the current largest interval into two by putting the third point p_3. This process is represented by a binary tree

rooted at the unit interval x_1. It is followed by two intervals x_2 and x_3, where $x_2 + x_3 = 1$ with $x_2 > x_3$. Then, x_2 has branches to x_4 and x_5 with $x_4 + x_5 = x_2$ and $x_4 \geq x_5$. The node x_3 is also followed by two node x_6 and x_7 such that $x_6 + x_7 = x_3$ and $x_6 \geq x_7$, and so on (see Figure 2.2).

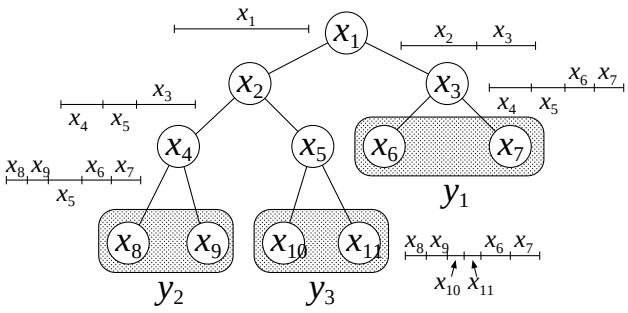


Figure 2.2: An example to be shown our strategy.

Then, the ratios are $r_1 = x_2/x_3, r_2 = x_3/x_5, r_3 = x_4/x_7, r_4 = x_5/x_9$ and $r_5 = x_6/x_{11}$. Since the intervals $x_6, \ldots, x_{11}$ are not split anymore, the partition of x_3 into x_6 and x_7 and that of x_4 into x_8 and x_9 and that of x_5 into x_{10} and x_{11} should be bisections at their center points to minimize the ratios, that is, $x_6 = x_7, x_8 = x_9$, and $x_{10} = x_{11}$. Now, let us denote x_7, x_9 and x_{11} by y_1, y_2 and y_3, respectively. Then, $x_3 = 2y_1, x_4 = 2y_2$ and $x_5 = 2y_3$. Therefore, $r_3 = x_4/x_7 = 2y_2/y_1, r_4 = x_5/x_9 = 2y_3/y_2$, and $r_5 = x_6/x_{11} = y_1/y_3$. Thus, the maximum ratio R_5 is minimized when r_3, r_4 and r_5 are equal, and it is given by

$$R_5 = (r_3 \cdot r_4 \cdot r_5)^{1/3} = \left(\frac{2y_2}{y_1}\frac{2y_3}{y_2}\frac{y_1}{y_3}\right)^{1/3} = 2^{2/3}. \tag{2.2}$$

2.3.4 Optimality and correctness

Finally, we prove that the maximum gap ratio R_P of the point sequence P computed by our strategy is equal to R_n. The magic number y_1 plays a very important role to optimize R_P. Lemma 2.3.4 determines the value of y_1 and guarantees the optimality of the resulting point sequence. The correctness of the strategy is proved in Lemma 2.3.6.

Lemma 2.3.4 *If any set of intervals $\{x_{i+1}, \ldots, x_{2i+2}, x_{2i+3}\}$ are sorted in non-increasing order with respect to their lengths, then the above strategy achieves the maximum gap ratio $R_P = 2^{\lfloor n/2 \rfloor/(\lfloor n/2 \rfloor+1)}$.*

Proof Let y_i denote the length of $x_{2(l+i)+1}$ for $i = 0, 1, \ldots, k$, where $k = \lfloor n/2 \rfloor$ and $l = \lceil n/2 \rceil$. Note that the node x_l has interval y_1 as one of the children in the tree configuration. Now, we assume the gap ratio r_i is defined by $\frac{x_{i+1}}{x_{2i+1}} = \frac{x_{i+1}}{y_i}$ for the $(l+i)$-th

insertion. By Lemma 2.3.6, this definition of r_i does not cause any inconsistency. From this fact, the minimum interval is y_i and the maximum interval is $x_{2(l+i)}+x_{2(l+i)+1} = 2y_{i+1}$, for $(l+i)$-th insertion $(1 \le i < k)$. At the last insertion, the minimum interval is y_{k+1} and the maximum interval is y_1. Therefore, the gap ratios r_i for $i = l, l+1, \ldots, l+k$, are given as follow,

$$\begin{aligned}
r_l &= \frac{x_{l+1}}{x_{2l+1}} = \frac{x_{2l+2}+x_{2l+3}}{x_{2l+1}} = \frac{2y_2}{y_1}, \\
r_{l+1} &= \frac{x_{l+2}}{x_{2l+3}} = \frac{x_{2l+4}+x_{2l+5}}{x_{2l+3}} = \frac{2y_3}{y_2}, \\
&\vdots \\
r_{n-1} &= r_{l+k-1} = \frac{x_{l+k}}{x_{2(l+k-1)+1}} = \frac{x_{2(l+k)}+x_{2(l+k)+1}}{x_{2(l+k-1)+1}} = \frac{2y_{k+1}}{y_k}, \\
r_n &= r_{l+k} = \frac{x_{l+k+1}}{x_{2(l+k)+1}} = \frac{y_1}{y_{k+1}} .
\end{aligned}$$

Since $x_{2i+3} > x_{4i+2}$ for $i \le l-1$, $r_i = \frac{x_{i+1}}{x_{2i+1}} = \frac{x_{2i+2}+x_{2i+3}}{x_{4i+2}+x_{4i+3}} \le \frac{x_{2i+2}}{x_{4i+3}} = r_{2i+1}$. This implies $R_n = \max\{r_l, r_{l+1}, \ldots, r_n\} \ge \max\{r_1, r_2, \ldots, r_{l-1}\}$. Thus, R_n is minimized when

$$\begin{aligned}
R_n &= (r_l \cdot r_{l+1} \cdot \cdots \cdot r_{l+k})^{\frac{1}{k+1}} = \left(\frac{2y_2}{y_1}\frac{2y_3}{y_2}\cdots\frac{2y_{k+1}}{y_k}\frac{y_1}{y_{k+1}}\right)^{\frac{1}{k+1}} \\
&= 2^{\frac{k}{k+1}} = 2^{\lfloor \frac{n}{2} \rfloor / (\lfloor \frac{n}{2} \rfloor + 1)}.
\end{aligned}$$

□

Since every $r_i = R_n$, we have $y_i = \frac{2}{R_n}\, y_{i+1}$ for $i = 1, \ldots, k$, and $y_{k+1} = \frac{y_1}{R_n}$. Moreover, $y_1 = \left(\frac{2}{R_n}\right)^{i-1} y_{i+1}$ for $i = 2, \ldots, k+1$. Thus, if y_1 is determined then so is every y_i. When n is odd, $1 = \sum_{i=n+1}^{2n+1} x_i = 2\sum_{j=1}^{k+1} y_j = 2\sum_{j=1}^{k+1} \left(\frac{R_n}{2}\right)^{j-1}$ leads to $y_1 = \frac{1}{2\sum_{j=1}^{k+1}\left(\frac{R_n}{2}\right)^{j-1}}$. Similarly, when n is even, $1 = \sum_{i=n+1}^{2n+1} x_i = y_1 + 2\sum_{j=2}^{k+1} y_j$ gives $y_1 = \frac{1}{1+2\sum_{j=2}^{k+1}\left(\frac{R_n}{2}\right)^{j-1}}$.

Observation 2.3.5 $y_1 \ge y_2 \ge \cdots \ge y_{k+1}$.

The observation follows from the facts that $y_i = \frac{2}{R_n}\, y_{i+1}$ and $\frac{2}{R_n}$ is greater than 1.

To show the correctness of this strategy and the optimality of the sequence obtained, we have to prove that the sequence $(x_{i+1}, \ldots, x_{2i}, x_{2i+1})$ generated by p_i is a sorted sequence in the non-increasing order of their lengths for every $1 \le i \le n$.

Lemma 2.3.6 *Whenever our strategy partitions the interval x_i for every $1 \le i \le n$, the resulting intervals $x_{i+1}, x_{i+2}, \ldots, x_{2i+1}$ are sorted in the non-increasing order, that is, we have $x_{i+1} \ge x_{i+2} \ge \cdots \ge x_{2i} \ge x_{2i+1}$.*

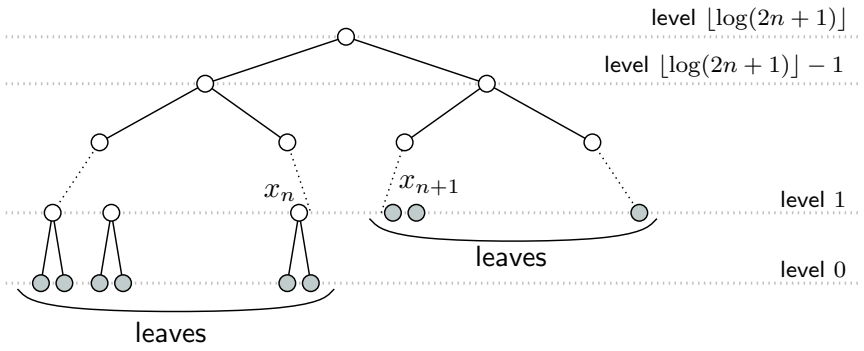


Figure 2.3: The levels of tree corresponding the behavior of our strategy.

Proof Proof is by induction on the level of a tree configuration of size $2n+1$. The level of a node v is defined as $\lceil \log(2n+1) \rceil -$ the height of v. So, all leaf nodes may be in the level 0 or 1, and the level of root x_1 is $\lfloor \log(2n+1) \rfloor$, (see Figure 2.5).

When $2^h = n+1$, where $h = \lfloor \log(2n+1) \rfloor$, all leaf nodes are in the level 0. In this case, from Observation 2.3.5, the statement $x_{n+1} \geq \ldots \geq x_{2n+1}$ holds. When $n+1 \neq 2^h$, the intervals x_n and x_{n+1} are both in the same level 1. Then, we have

$$\begin{aligned} x_n &= 2y_{k+1} = 2\left(\frac{R_n}{2}\right)^k y_1 = \frac{R_n^{k+1}}{R_n\ 2^{k-1}}\ y_1 = \frac{2^k}{R_n\ 2^{k-1}}\ y_1 \\ &= \frac{2}{R_n}\ y_1 > y_1 = x_{n+1}. \end{aligned}$$

On the remaining nodes in level 1, both children of a node are the intervals of the same length $2y_j$. Hence, for two intervals x_i and x_{i+1}, we have $x_i \geq x_{i+1}$ by Observation 2.3.5.

Let $I^i = (z_1^i, \ldots, z_{2^i}^i)$ be the intervals in the level i, where z_1^i and $z_{2^i}^i$ are the leftmost and rightmost intervals in the level i, respectively. Now, we assume that the statement holds up to the level i, that is, $z_1^i \geq z_2^i \geq \cdots \geq z_{2^i}^i \geq \cdots \geq x_{2n+1}$. By the induction hypothesis, we have $z_{2^{i-1}}^{i-1} = z_{2^i-1}^i + z_{2^i}^i \geq z_1^{i+1} + z_2^{i+1} = z_1$. We also have $z_j^{i-1} \geq z_{j+1}^{i-1}$ by a similar argument. □

Thus, we have a conclusion on 1-dimensional dispersion problem.

Theorem 2.3.7 *Given an integer n, our strategy gives an optimal solution with the maximum gap ratio being $2^{\frac{\lfloor n/2 \rfloor}{\lceil n/2 \rceil + 1}}$ on the 1-dimensional dispersion problem in $O(n)$ time.*

2.4 2-dimensional problem

2.4.1 Notations

Let $s_1 = (0,0)$, $s_2 = (1,0)$, $s_3 = (1,1)$, and $s_4 = (0,1)$ be the four corner points of $\mathbb{S}^2$. For each point set S_i after inserting i points in P, we define two empty circles C_i and c_i: The

diameter of C_i is G_i and the center p of C_i satisfies $\min_{s \in S_i} d(s, p) = G_i/2$. The diameter of c_i is g_i and its center is the midpoint of the closest pair of points. Note that the two empty circles are not unique, since the maximum gap and the minimum one may be defined by some of the triples or pairs. We break ties arbitrarily to choose C_i and c_i. For any three different points p_1, p_2, and p_3, not on a line, let $C(p_1, p_2, p_3)$ be the circle passing through the three points. The interior of a circle C is denoted by int C and the diameter of C is denoted by $diam(C)$. The gap ratio r_i is defined by $r_i = G_i/g_i = diam(C_i)/diam(c_i)$.

2.4.2 Analytical results

The shape of a region for the first point

We start to investigate the shape of a region which we can locate the first point p_1 preserving $r_1 \leq 2$. First we formulate the boundary of the region A_1 with $r_1 \leq 2$ inside of the triangle (say I_1) defined by the lines $y = \frac{1}{2}$, $y = x$, and $x = \frac{1}{2}$. If the equation can be formulated, we can construct the whole region by rotation and transformation (*cf.* Figure 2.4).

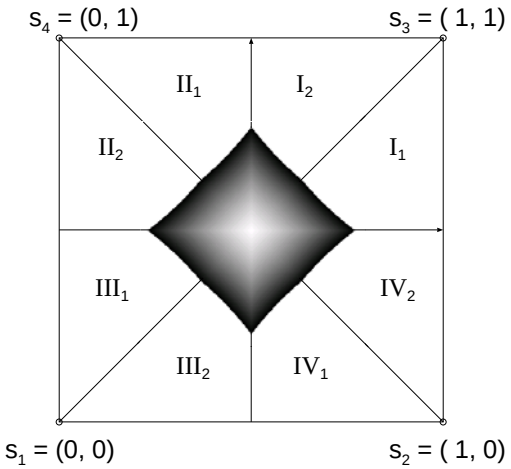


Figure 2.4: The shape of the region in which we can locate p_1.

Since p_1 will be located in sub-quadrant I_1, the maximum gap G_1 is the radius of a circle passing through p, s_1 and s_4, and the minimum gap g_1 is the half of $d(p, s_3)$. We let denote $f(x)$ a function for the boundary of A_1, that is, $r_1 = G_1/g_1 = 2$.

We first determine the boundary condition of the function f. When p_1 lie on the x-axis, since G_1 is the diameter of the circle $C(p_1, s_1, s_4)$ and g_1 is the distance from p_1 to s_3, the x-coordinate holds following equation,

$$\frac{x^2 + 2x + 2}{(1 + x)\sqrt{1 + (1 - x)^2}} = 2.$$

Therefore, $p_1 = (\frac{\sqrt{7}-1}{6}, 0.5)$. When p_1 lie on $y = x$, the x-coordinates holds following equation,

$$x^2 - 4x + 1 = (x - 2 - \sqrt{3})(x - 2 + \sqrt{3}).$$

Therefore, $p_1 = (1 - \frac{\sqrt{3}}{2}, 1 - \frac{\sqrt{3}}{2})$.

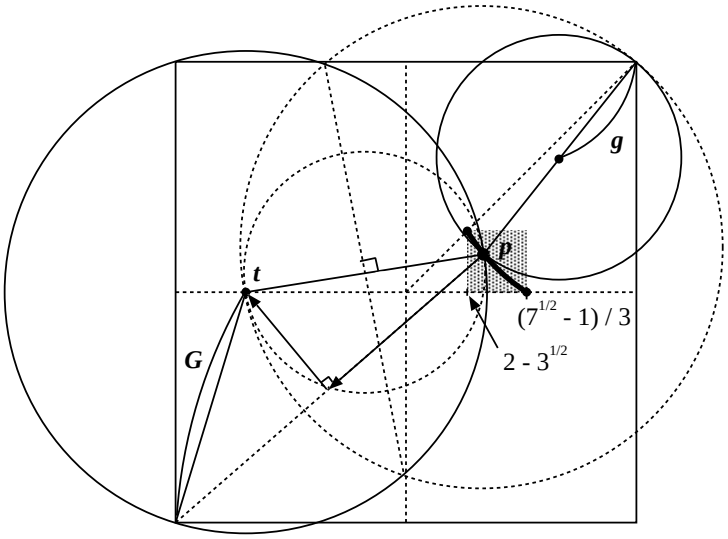


Figure 2.5: The boundary of A_1.

We assume that first point p_1 is in $[1 - \frac{\sqrt{3}}{2}, \frac{\sqrt{7}-1}{6}] \times [\frac{1}{2}, 1 - \frac{\sqrt{3}}{2}]$, with $r_1 = 2$. Let C be a circle centered at p_1 passing through s_3. Note that the radius of C is equal to the minimum gap g_1. Since the center of the largest empty circle C_1 always lie on the line $y = \frac{1}{2}$, the intersection between C and $y = \frac{1}{2}$ being in $\mathbb{S}^2$ is the center of C_1. Let $(t, \frac{1}{2})$ be the coordinates of center of C_1, then

$$(t - x)^2 + y^2 = g_1^2. \tag{2.3}$$

And t can be represented as

$$t = \frac{-1 + x}{2} + \frac{-1 + y^2}{2(1 + x)} = \frac{-2 + x^2 + y^2}{2(1 + x)}, \tag{2.4}$$

since t is the center of circumcircle of s_1, s_4 and p (see Figure 2.5). By substituting Equation (2.4) to Equation (2.3), we can obtain the function f as follow

$$\begin{aligned} \left(\frac{1 + x}{2} + \frac{-1 + y^2}{2(1 + x)}\right)^2 + y^2 - (1 - x)^2 - (1 - y)^2 &= 0 \\ 4 - 12x^2 - 4x^3 - 8y - 16xy - 8x^2y + 4y^2 + 4xy^2 + 2x^2y^2 - y^4 &= 0. \end{aligned} \tag{2.5}$$

Figure 2.6 shows the graph of equation (2.6).

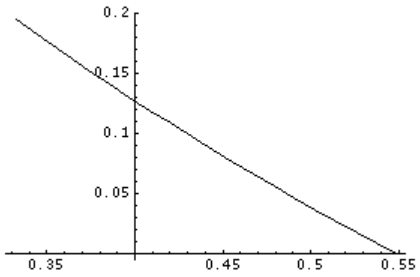


Figure 2.6: The graph of equation (2.6) expressing the boundary of A_1.

A proof for optimality in two points problem

Lemma 2.4.1 *For $i = 1, 2, \ldots, n-1$, if $\max_{1 \le j \le i} r_j < 2$, then p_{i+1} must be inserted in int C_i, to keep $r_{i+1} < 2$.*

Proof If p_{i+1} does not lie in int C_i, then $G_{i+1} = G_i$. We have $g_{i+1} \le G_i/2$ since there is no empty circle whose diameter is greater than that of the largest empty circle. Hence $r_{i+1} \ge 2$. □

Fact 2.4.2 *For two acute triangles $\triangle ABC$ and $\triangle DEF$, if $\angle ABC \le \angle DEF$ and $\angle BCA \ge \angle EFD$, then*

$$\frac{|AB|}{|CA|} \ge \frac{|DE|}{|FD|}.$$

We have equality if $\angle ABC = \angle DEF$ and $\angle BCA = \angle EFD$. See Figure 2.7 for an example.

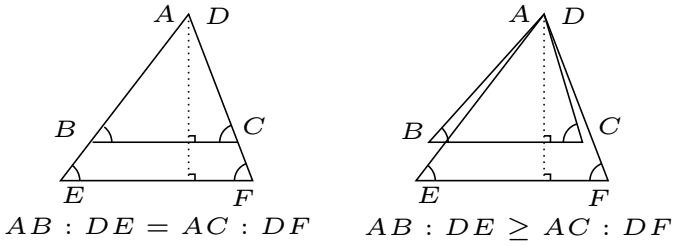


Figure 2.7: Illustration for Fact 2.4.2

Lemma 2.4.3 *The last point p_n must lie at the center of C_{n-1} to minimize the maximum gap ratio.*

Proof By Lemma 2.4.1, we assume that p_n is inserted into the interior of C_{n-1}. If C_{n-1} is not unique, then this lemma immediately holds, since it must maximize the minimum gap g_n.

We assume that C_{n-1} passes through three points a, b and c in the counter-clockwise order. Let C' be the other empty circle (not C_{n-1}) passing through a and b.

We move p_n along the perpendicular bisector of a and b so as to decrease g_n. Note that we may also have to consider motions between b and c, and between c and a. But similar arguments can be appropriately applied.

In this situation, a pair of points defining g_n is never changed, i.e., $g_n = d(p_n, a) = d(p_n, b)$. However, a triple (or pair) of points defining G_n may change. There are two kinds of meaningful circles which may define G_n; the first one is C' defined above and the second one is the empty circle C'' that passes through a and p_n. The other circles may lead to $r_n \geq 2$, or may not lead to a better r_n than that of C' or C''. Note that p_n has to be inserted at the center of C_{n-1} to maximize g_n, when $C_n = C'$. On the other hand, when $C_n \neq C'$, p_n lies in int C'.

Now, we assume $C_n = C''$. Let o be the center of C_{n-1}, o_1 be the center of C' and o_2 be the center of C''. Consider two triangles $\triangle_1 = \triangle(a, o_2, p_n)$ and $\triangle_2 = \triangle(a, o_1, o)$. Since p_n is in C', it can be seen that $\angle a o_2 p_n < \angle a o_1 o$ and $\angle a p_n o_2 > \angle a o o_1$ by simple calculations. Hence, we have $d(a, o_2)/d(a, p_n) > d(a, o_1)/d(a, o)$ from Fact 2.4.2. This concludes the proof. □

By Lemma 2.4.3, we can find an optimal 2-point sequence. Figure2.8 shows the notations for an instance of $n = 2$. In this figure, we consider that p_1 does not lie on the line $y = \frac{1}{2}$. The three circles C', C'' and C''' are shown, where C''' is the largest empty circle passing through p_1 when p_2 is in the largest empty circle passing through the symmetric point p_1' of p_1 with respect to $y = \frac{1}{2}$, but the meaningful circles are just C' and C''. Since $\theta_1 - \theta_2 > 0$ and $\theta_3 - \theta_2$, p_2 is put at the center of C_1, from Fact 2.4.2 and Lemma 2.4.3.

We can assume that p_1 lies on the line $y = \frac{1}{2}$, to maximize g_1. Since we can specify p_2 once p_1 is determined, we only examine the x-coordinate of p_1. The maximum gap ratio R_P is minimized when $g_1 = g_2$, and then an optimal point pair satisfies $G_1^2 = 2 g_1 G_2$, by simple observations. Hence, for example, these gaps G_1, g_1 and G_2 are given by

$$
\begin{aligned}
G_1 &= \frac{4x_1^2 - 8x_1 + 5}{4 - 4x_1}, \quad g_1 = \sqrt{x_1^2 + \frac{1}{4}}, \text{ and} \\
G_2 &= 2\sqrt{\frac{1}{4} + \left(x_1 - \frac{1}{2}\right)^4}.
\end{aligned}
$$

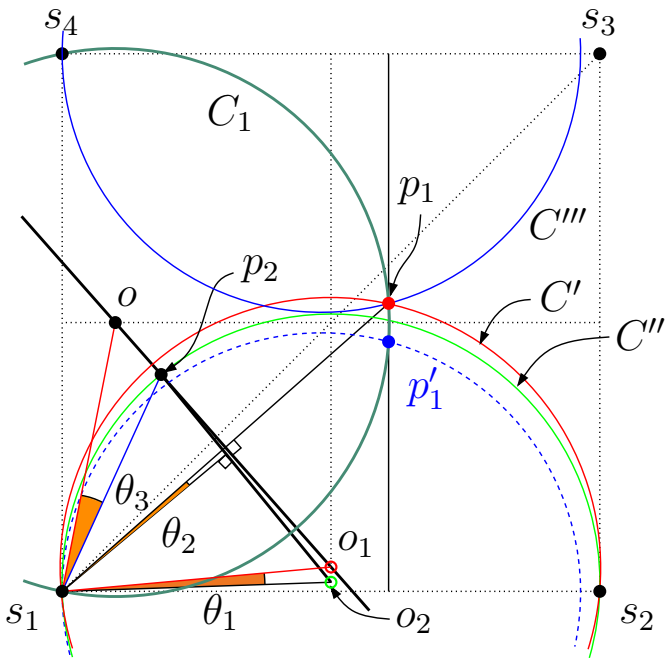


Figure 2.8: Notations of Lemma 2.4.3 for an instance of $n = 2$.

Solving this simultaneous equations, we obtain the coordinates of optimal points;

$$\begin{aligned} p_1^* &= (0.273704, 0.5), \text{ and} \\ p_2^* &= (0.808958, 0.5). \end{aligned}$$

Next, we consider the cases of $n = 3$ and larger n. They are more complicated and may not be solvable in an analytical sense.

A proof for optimality in three points problem

Lemma 2.4.4 *If $n \geq 3$ and the first point p_1 lies on the line $y = \frac{1}{2}$ or the line $x = \frac{1}{2}$, then the maximum gap ratio is greater than or equal to 2.*

Proof We assume that the first point p_1 lies on the line $y = \frac{1}{2}$. When p_2 is inserted into int $C_1 \cap$ int $C(p_1, s_1, s_2)$, p_3 should be inserted at the center of $C(p_1, s_3, s_4)$ from Lemma 2.4.3, and then C_3 is defined by $C(p_2, s_1, s_2)$. Since $diam(C_3) \leq 1$ and $diam(C(p_1, s_3, s_4)) \geq 1$, we have $r_3 \geq 2$. So, p_2 has to be inserted anywhere in int $C_1 \setminus$ int $C(p_1, s_1, s_2) \cap \mathbb{S}^2$. Hence, C_2 is defined as the circle passing through p_1, s_1 and s_2. By Lemma 2.4.1, p_3 is inserted in the interior of C_2. Therefore, the third gap ratio r_3 is at least 2, since $diam(C_2) = diam(C_3)$ and $diam(c_2) \leq \frac{1}{2} diam(C_2)$.

By the symmetry, a similar argument can be applied when p_1 lies on the line $x = \frac{1}{2}$. □

Lemma 2.4.5 *When $n = 3$, the second point p_2 should be inserted at the center of C_1.*

Proof Let C_1' and C_1'' be the second and third largest empty circles of S_1. Consider the case in which there exist exactly two circles, C_1 and C_1', with their diameters greater than 1, at the end of the first insertion. Then, p_2 must be in int $C_1' \cap \{p \in S \mid d(p_1, p) > \frac{1}{2}\} \cap \{p \in S \mid d(s_i, p) > \frac{1}{2}\}$, where s_i is the nearest corner point of S, since C_2 passes through p_2, $diam(C_2) > 1$ and $g_2 = \max\{d(p_1, p_2), d(s_i, p_2)\} < \frac{1}{2}$. If such intersection does not exist, then we can see $R_P \geq 2$. Let x and x' be the centers of C_1 and C_1', respectively. Since p_2 is inserted in that intersection, $C_2 = C(p_2, s_i, s_j)$, where s_j is the second nearest corner point of $\mathbb{S}^2$ from p_2. Let x'' be the center of C_2. Now, consider two triangles, $\triangle s_i x'' p_2$ and $\triangle s_i x' x$. From Fact 2.4.2, p_2 is inserted at the center of C_1, to minimize r_2.

Next, consider the next case that there are three empty circles, C_1, C_1', and C_1'', with diameters greater than 1, at the end of the first insertion. This case occurs when p_1 is contained in exactly one circle $C(o, s_i, s_j)$, where o is the center $(\frac{1}{2}, \frac{1}{2})$, and s_i and s_j are corner points of S. We assume that p_2 is inserted in int $C_1' \cap \{p \in S \mid d(p_1, p) > \frac{1}{2} diam(C_1'')\} \cap \{p \in S \mid d(s_i, p) > \frac{1}{2} diam(C_1'')\}$, where s_i is the nearest corner point of S from p_2, the maximum gap ratio may be less than 2. If the intersection does not exist, then $R_P \geq 2$. However, the same argument as above applies. Therefore, p_2 should be inserted at the center of C_1, if $n = 3$. □

Lemma 2.4.6 *When $n = 3$, all largest empty circles C_1, C_2 and C_3 pass through p_1.*

Proof It is obvious that C_1 and C_2 pass through p_1 from Lemmas 2.4.4 and 2.4.5. If p_2 lies on the line $y = \frac{1}{2}$, then p_3 lies on the line $x = \frac{1}{2}$, and vice versa. We can assume that p_2 and p_3 are located on the boundary of the cube $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$ from Lemmas 2.3.6 and 2.4.5. Then, p_1 lies in the (open) square $(\frac{1}{2}, 1) \times (\frac{1}{2}, 1)$. Hence, C_3 passes through p_1, since the open half plane $y > \frac{1}{2}$ contains p_1 but not p_2 or p_3. □

Since p_2 and p_3 are inserted at $center(C_1)$ and $center(C_2)$, respectively, we obtain $g_2 = \frac{1}{2} G_1$ and $g_3 = \frac{1}{2} G_2$. In order to minimize R_P, we take geometric average among r_1, r_2 and r_3;

$$\begin{aligned} R_P &= \sqrt[3]{r_1 \cdot r_2 \cdot r_3} = \sqrt[3]{\frac{G_1}{g_1} \frac{G_2}{g_2} \frac{G_3}{g_3}} = \sqrt[3]{\frac{G_1}{g_1} \frac{G_2}{\frac{1}{2} G_1} \frac{G_3}{\frac{1}{2} G_2}} \\ &= \sqrt[3]{2^2 \frac{G_3}{g_1}}. \end{aligned} \tag{2.6}$$

Hence, if we can show $\frac{G_3}{g_1} < 2$, then $R_p < 2$ is obtained when $n = 3$. The problem is to find a point p_1 which minimizes the value of equation (2.6). We can formulate it as a

non-linear programming problem. However, it seems to be difficult to specify an optimal position of p_1 satisfying the above conditions, and analytically solving the exact positions in an optimal point sequence is too complicated even if n is rather small, say $n = 3$. So, we propose a heuristic algorithm for finding a good point sequence.

2.4.3 Heuristic Algorithms

We present a simple heuristic algorithm based on local search. First, we describe a procedure to compute the maximum gap ratio, for a given n-point sequence P. Then, we show a main procedure which treats n-point sequence $(p_1, \ldots, p_n)$ as a point $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ in the $2n$-dimensional space $\mathbb{R}^{2n}$ to find a best point by examining its neighborhood in $\mathbb{R}^{2n}$. This technique is similar to the lifting technique common in computational geometry.

Algorithm 3: ComputeMaxGapRatio(P)

input : A point sequence $P = (p_1, p_2, \ldots, p_n)$
output: The maximum gap ratio R_P

```
1 Let S_0 be the corner points of S^2;
2 S ← S_0;    r ← 0;
3 for i = 1, ..., n do
4 |   S ← S ∪ {p_i};
5 |   Compute the maximum gap G_i and the minimum gap g_i;
6 |   if r < G_i/g_i then r ← G_i/g_i;
7 return r;
```

Algorithm 3 computes the maximum gap ratio for a given n-point sequence. It runs in $O(n^2)$ time. In particular, we maintain a planar subdivision by a Delaunay triangulation [100] for each S_i. The planar subdivision by Delaunay triangulation is a planar graph. So, each face contributes to an empty circle and each edge represents the neighborhood relation between two connecting vertices (see [106]). Hence, we obtain the gaps G_i and g_i in linear time, since the reconstruction of the subdivision is the crucial part.

Algorithm 4 is a main procedure of our heuristics. Given three parameters n, m and k, the algorithm iterates local search k times starting from a randomly generated point sequence. In each iteration we compute a local optima of an n-point sequence. The parameter m is used to specify a termination condition to guarantee the accuracy of solutions obtained.

Algorithm 4: A simple local search heuristic algorithm

input : Integers n, m, and k.

output: A good n-point sequence.

```
1  Let e_1 and e_2 be the base unit vectors;
2  r_opt ← ∞;   threshold ← 2^{-m};
3  for i = 1 to k do
4      Initialize P by a randomly generated n-point sequence;
5      ε ← 1/2;
6      r_min ← ComputeMaxGapRatio(P);
7      repeat
8          foreach p'_1 ∈ {p_1 ± εe_1, p_1 ± εe_2} do
9              foreach p'_2 ∈ {p_2 ± εe_1, p_2 ± εe_2} do
10                 ⋮
11                 foreach p'_n ∈ {p_n ± εe_1, p_n ± εe_2} do
12                     P' ← (p'_1, p'_2, ..., p'_n);
13                     r ← ComputeMaxGapRatio(P');
14                     if r < r_min then r_min ← r;   P ← P';
15         if r_min is updated then P'' ← P;
16         else ε ← (1/2)ε;
17     until ε < threshold ;
18     if r_opt > r_min then r_opt ← r_min;   P* ← P'';
19 return P*;
```

2.4.4 Experimental Results

We have implemented Algorithm 4 to evaluate the accuracy of the solutions obtained by the heuristic algorithm. Table 2.1 describes our environment of the experiment. We designed the algorithm using the exact computation in LEDA [94] for the sake of accuracy and for robustness.

Table 2.2 shows the best R_p values obtained by Algorithm 4. For each $n = 2, 3, 4, 5$, we executed the algorithm more than 1000 times with the threshold less than 10^{-8}. For each of $n = 6, 7, 8$, we executed it with 500, 100, and 20 trials with the same accuracy.

As mentioned above, we have an exact value of R_2 and a property for R_3. The computed value R_2 shown in the table finds to be close enough to the exact value of R_2. The 3-point sequence achieving the computed value of R_3 shown in the table satisfies

Table 2.1: The environment of experiment

Workstation	CPU
Dell PowerEdge SC1425 Server	Intel ® Xeon^{TM} 3.6GHz
Main memory	OS
8GB	RedHat Enterprise Linux 3
Compiler	External library
g++-3.4.2	LEDA-5.0.1

Table 2.2: The best solutions obtained by Algorithm 4

n	2	3	4	5
R_p	1.87804	1.92716	1.927164	1.92716
n	6	7	8	9
R_p	1.927203	1.99312	2.008371	–

the property $\frac{G_3}{g_1} < 2$, and we conjecture that the point sequence is optimal. In addition to this, the obtained point sequences for $n = 4, 5, 6$ may also be optimal, since these maximum gap ratios are roughly the same.

There is a gap between the results for $n = 6$ and $n = 7$. We have obtained a better sequence than that of Voronoi insertion. However, in the case of $n = 8$, we did not obtain a sequence with the maximum gap ratio less than 2. In our environment of experiment, we gave up to apply the algorithm for $n \geq 8$, since it is too slow. In fact, it took one day per one trial.

We could use those point sequences obtained above as seed point sequences and perform the incremental Voronoi insertion afterwards. This is our second heuristic algorithm.

We have implemented the above-stated strategy using the 7-point sequence shown in Table 2.3 as a starting seed point sequence. The initial maximum gap ratio is 1.993124. Figure2.9 indicates the resulting point distribution. The maximum gap ratio of this point sequence is actually 1.99921.

Furthermore, we consider the irregularity of the final point distribution for each of our results and Voronoi insertion. In order to enhance the difference between them, we use a Delaunay triangulation shown in Figure 2.10. Our distribution is pretty irregular, compared with that obtained by the Voronoi insertion.

One of the notable remarks is that Voronoi insertion easily gives a uniform point sequence in our criterion, but the final distribution is globally non-uniform and locally

Table 2.3: A good seed point sequence and the initial maximum gap ratio.

p_1	p_2
$(0.769146, 0.501913)$	$(0.263398, 0.508807)$
p_3	p_4
$(0.499994, 0.0637435)$	$(0.477718, 0.891089)$
p_5	p_6
$(2.0687e-05, 0.317322)$	$(8.21674e-06, 0.662797)$
p_7	R_P
$(0.999993, 0.304037)$	1.993124

regular.

2.5 Conclusions and Futher researches

In this part, we have presented a preliminary result on generalized dispersion problems. One of the most important future works is to extend the result to higher dimensions. We showed some results on lower and upper bounds of the maximum gap ratio for the planar case, but none in the higher dimensions.

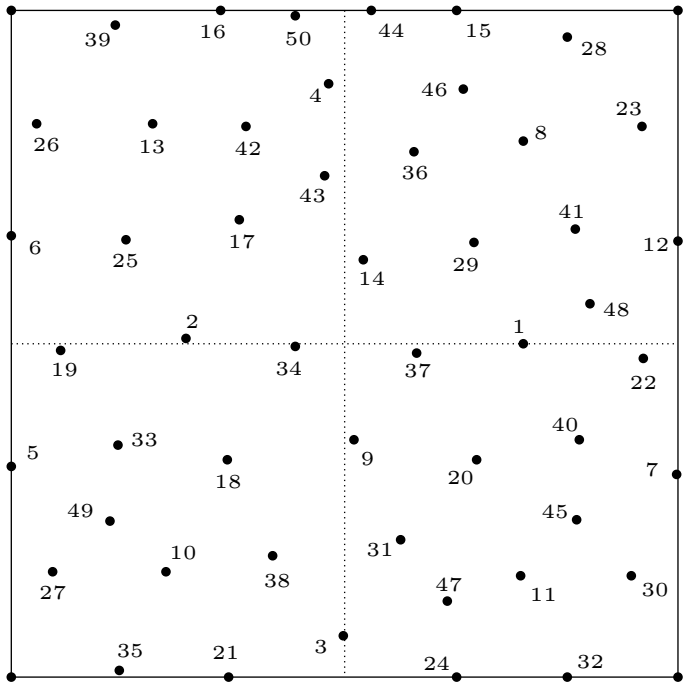


Figure 2.9: A good 50-point sequence with the maximum gap ratio bounded by 1.99921.

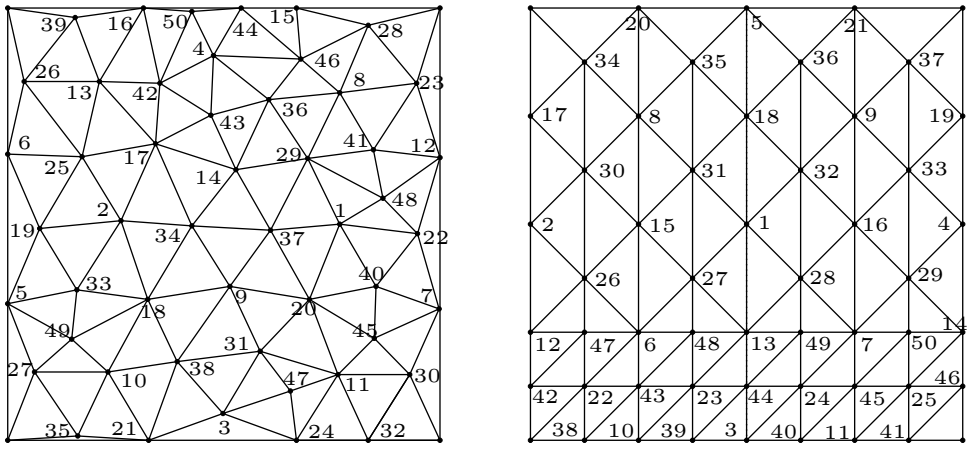


Figure 2.10: The Delaunay triangulations for the resulting point distributions. In the triangulation for our 50-point sequence the maximum gap ratio is bounded by 1.99921, which is shown in the left. The triangulation for the incremental Voronoi insertion is given to the right.

Chapter 3

Random Generation for Geometric Objects

3.1 Introduction

A large class of computational problems can be viewed as the seeking of partial information about a relation which associates *problem instances* with a set of *feasible solutions.* In computer science, mathematics, and in other fields, it is often necessary to examine all of a finite number of possibilities in a combinatorial/geometric structure, in order to solve a problem or gain insight into the solution of a problem.

Generation problems are less well studied but have a number of computational applications. For example, uniform generation can be seen as a way of exploring a large set of combinatorial structures and constructing typical representatives of it. Random generations may be used to formulate conjectures about the set, or perhaps as test data for the empirical analysis of some heuristic algorithm which takes inputs from the set. Non-uniform generation occurs in the mathematical modelling of physical systems where the structures are valid system configurations each of which has a weight which depends on its energy. Many important properties of the model can be deduced from estimates of the expectation, under the weighted distribution, of certain operators on configurations.

From a computational point of view, the problems of *approximate counting* and *almost uniform generation* are very closely related. More precisely, Jerrum, Valiant & Vazirani [77] (see also [117]) have shown that for most natural structures a polynomial time procedure for approximate counting can be used to construct a polynomial time almost uniform generation algorithm, and *vice versa.* The only assumption we need to make is that the structures are *self-reducible,* which essentially means that they possess a simple inductive construction in terms of similar structures of a smaller size.

In this chapter, we study problems for randomly generating simple polygons or *simple polygonalizations* with respect to a given point set in the plane. A simple polygonalization of S is defined as one of the crossing-free Hamiltonian cycles in S. In general, there exist a lot of such cycles or possibilities in a fixed point set. This problem has enormous significance in development better random sampling techniques, since the set of all possibilities in simple polygonalizations gives a good solution space of Euclidean Traveling Salesman problems, rather than choosing randomly some of $n!$ possibilities in simply permuting point-sequence.

3.2 Heuristics for Generating Simple Polygonalizations

Geometric Enumeration deals with problem of listing all geometric objects that satisfy a specified property. Typical objects to be enumerated are triangulations of a set of planar points. Triangulation is one of the most important geometric structures in Computational Geometry [21, 106], so that a number of literature have proposed good enumerating algorithms [18, 20, 3].

In addition to theoretical interest, generation of random geometric objects has applications that include testing and verification of time-complexity of computational geometric algorithms for practical rather than worst-case behavior. In that sense, generating a simple polygon seems to be more practical importance than a triangulation, since there are many applications which treat a simple polygon as an instance in geometric optimization, such as Art Gallery Problems [101, 116], Polygon Partitioning [82], Geometric Shortest Paths [73], and so on [111, 65]. This chapter considers how to generate efficiently simple polygonalization at random.

Problem statement

Given a set S of n points in general position in the plane, randomly generate a simple polygon whose polygon vertices are precisely of S, or compute a random *simple polygonalization* of S.

More precisely, we would like simple polygonalize a set S of points so that each member of all simple polygonalizations of S is generated uniformly at random. The problem of maximizing the number of simple polygonalizations over all n point sets plays a role in random generation of simple polygons (in maximizing the probability that a random permutation of a given set of points defines a simple polygon). However, examining the set

of all simple polygonalizations of S is quite difficult even in the counting problem which asks how many simple polygonalizations are there in S, see [91]. A brief history of the asymptotic bounds on this number are summarized by Demaine [41]. The currently known approximate upper and lower bounds are $O^*(86.81^n)$ due to Sharir and Welzl [114] and $\Omega^*(4.642^n)$ due to García, Noy and Tejel [64], respectively. Note that the notations $O^*(\cdot)$ and $\Omega^*(\cdot)$ are similar with O- and Ω-notations but these neglect the factor of polynomials: $f(n) = O^*(g(n))$, or $f(n) = \Omega^*(g(n))$ if and only if there exists a polynomial $p(n)$ such that $f(n) = O(g(n)p(n))$, or $f(n) = \Omega(g(n)p(n))$, respectively. Due to the high upper bound heuristic approaches have been adopted to generate a simple polygonalization during the past decade. Hence purpose in this chapter can be formulated as follows.

Problem: Random Generation for Simple Polygonalizations	
Input:	A set S of n points in the plane.
Output:	A simple polygonalization $\mathcal{P}$ of S.
Ensurement:	Every simple polygonalization for S has possibility to be generated with positive probability.

Related works

Simple polygonalizations are also called simple polygonizations, or crossing-free Hamiltonian cycles. There are a few related optimization problems such as the Traveling salesman problem. In fact, the problems for computing a simple polygonalization with the minimum total polygon edge length [124] or with the minimum or maximum area [55] are NP-complete.

As we mentioned, it is an outstanding open problem whether the number of simple polygonalizations of S can be computed in polynomial time. There are two different approaches in the literature: one is to investigate a sub-class of simple polygons such as x-monotone polygons [90, 126], and star-shaped polygons [15, 16, 118, 122]; the other is to design an efficient heuristics [15, 16, 37, 103, 126]. Auer and Held [15], and Zhu, Sundaram, Snoeyink and Mitchel [126] independently proposed a practically useful heuristic, so called 2-opt Moves. It can generate every possible simple polygonalization with a *positive probability* (this implies there is no possible simple polygons which cannot be generated by the heuristic). Any other heuristics cannot generate all possible simple polygonalizations, or are impractical by the experimental results of Auer [16].

2-opt Moves is experimentally good, but, it requires $\Theta(n^3)$ times "untangling 2-opt" moves before convergence to a simple polygon in the worst case, as is proved by Van Leeuwen and Schoone [85]. Hence, the time complexity of 2-opt Moves is $O(n^4 \log n)$ with

sweep-line technique (see e.g., [21]) in the worst case. In particular, an implementation of 2-opt Moves has been included in CGAL [25]. Unfortunately it takes too much time to implement 2-opt Moves for a larger instance, and any algorithms proposed so far seem to be impractical. We propose a simple heuristic algorithm which runs fast enough even for a large instance.

Our contributions

We propose a triangulation-base heuristic algorithm which generates each possible simple polygon with a positive probability in $O(n \log n + f)$ time, where f is the number of edge-flipping operations which may have effect on the randomness. This may improve the time complexity $O(n^4 \log n)$ of 2-opt Moves.

Our algorithm consists of three phases: first, it generates a triangulation T of given point set at random; next computes a random maximal *polygon tree* on the dual $\mathcal{D}(T)$ of T. A polygon tree $\mathcal{T}$ on the dual $\mathcal{D}(T)$ of a triangulation T is a tree such that $\cup_{v \in \mathcal{T}} g^{-1}(v)$ is a simple polygon, where g is bijection from a face in T to a vertex in $\mathcal{D}(T)$; and finally constructs a simple polygon by traversing on the maximal polygon tree with depth-first search.

In section 3.4, we describe our heuristic algorithms in detail. We show the efficiency and the usefulness of algorithms by computer experiments in section 3.5. Our approach may applicable to the generalized simple polygonalization problems. We discuss it in Section 3.6. Finally, we show future works in Section 3.7.

3.3 Preliminaries & Fundamental results

A *polygon* is a (closed) region of the two-dimensional plane bounded by a finite collection of line segments forming a closed curve. A polygon P is said to be *simple* if points of the plane belonging to two polygon edges of P are limited to the polygon vertices of P. Hence, there are no self-intersections or no holes (see Figure 3.5), and then a simple polygon is topologically homeomorphic image of a disk. Figure 3.5 (a) and (b) depict a simple polygon with 64 vertices, and a nonsimple polygon with 5 self-intersections and 3 holes, respectively. In this paper, polygons mean simple polygons unless it is stated. We sometimes treat a polygon P with k vertices, or k-gon, as a circular list of k polygon vertices $(v_1, v_2, \ldots, v_k, v_{k+1} = v_1)$ in clockwise-order and denote the number of polygon vertices or polygon edges by $|P|$. We also make sure that the visibility between two points in a simple polygon. We say that point x can *see* point y (or y is *visible* to x) if and only if the closed segment $\overline{xy}$ is nowhere exterior to the polygon P : $\overline{xy} \subseteq P$.

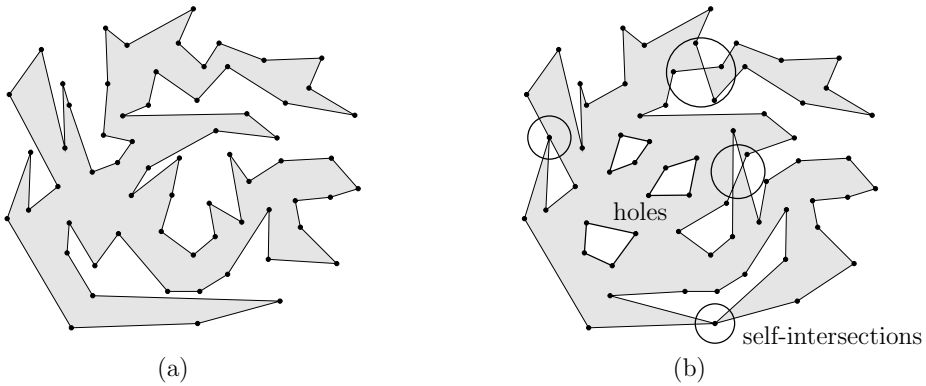


Figure 3.1: Simple and nonsimple polygons

Throughout this paper we let S stand for a finite set of n points in the plane. We assume that S is in general position, i.e., no three points are collinear and no four points are cocircular.

A *triangulation* of planar points S, denoted by $T(S)$, is a simplicial decomposition of its convex hull $\mathcal{CH}(S)$ whose vertices are precisely the points in S. In other words, a triangulation is a maximal crossing-free geometric graph on S (in a geometric graph the edges are realized by straight line segments). To distinguish the terminologies of 'vertex' and 'edge' between polygon and graph, we explicitly specify the modifier "polygon" for vertex and edge of polygon.

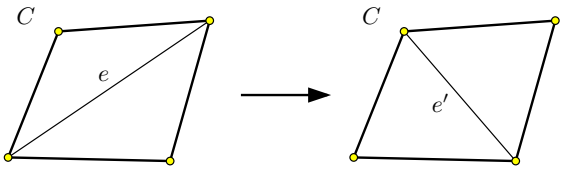


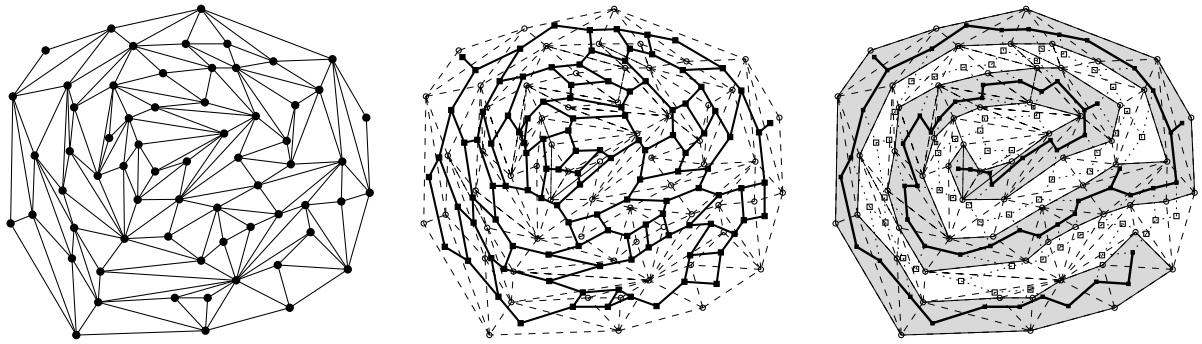
Figure 3.2: Flipping edge e within a convex quadrilateral C in plane triangulations.

In a triangulation $T(S)$, an edge e of $T(S)$ is *flippable* if it is adjacent to two triangles whose union is a convex quadrilateral C. By *flipping* e we mean an operation of removing e for $T(S)$ and replacing it by the other diagonal of C (see Figure 3.2). In this way we obtain a new triangulation $T'(S)$, and we say that $T'(S)$ has been obtained from $T(S)$ by means of a *flip*. Lawson [84] showed that any two triangulations of a planar point set can be transformed into each other by flipping edges. Fortune [59] showed that at most $\binom{n}{2}$ flips are sufficient to compute Delaunay triangulation. This implies there exists a sequence of $O(n^2)$ flips which transforms a triangulation to any other. More precisely, Hurtado, Noy and Urrutia [75] showed that if a set of n points has k *convex layers*[1], then

[1]convex layers in a set of planar points are obtained from removing the convex hull (the first layer)

one triangulation can be transformed into the other triangulation using $O(kn)$ flips.

We denote a set of vertices, edges and faces of $T(S)$ by V, E and F, respectively. A triangulation $T(S)$ is always associated with a *dual graph*. Let $\mathcal{D}(T) = (V_{\mathcal{D}}, E_{\mathcal{D}})$ be the dual graph of $T(S)$. We can define a bijection $g : F \hookrightarrow V_{\mathcal{D}}$, and then $(v, w) \in E_{\mathcal{D}}$ for any distinct $v, w \in V_{\mathcal{D}}$ if and only if the triangles, or the reverse images, $g^{-1}(v)$ and $g^{-1}(w)$ share a common edge in E. Figure 3.3 (a) and (b) depict a triangulation of planar points and its dual graph, respectively.



(a) (b) (c)

Figure 3.3: Triangulation $T(S)$ of planar points, its dual graph $\mathcal{D}(T)$, and a polygon tree on $\mathcal{D}(T)$.

We neglect the unbounded face in a triangulation and its dual. So, we will refer to each face in a triangulation as triangle, and the degree of each vertex in the dual is at most 3. Since a triangulation $T(S)$ is a planar graph, the size of a triangulation and its dual can be derived from the Euler's formula[2]. Hence we have that $|V| = n$, $|E| = 3n - 3 - k$, $|F| = |V_{\mathcal{D}}| = 2n - 2 - k$, and $|E_{\mathcal{D}}| = 3n - 3 - 2k$, where $k = |\mathcal{CH}(S)|$.

We also consider a *polygon triangulation* of a simple polygon P, denoted by $T(P)$, that is, a simplicial decomposition of P whose vertices are precisely the polygon vertices of P, or the subdivision of P into non-overlapping triangles using diagonals only. The following theorems are important to design our heuristic algorithm.

Theorem 3.3.1 (Triangulation Theorem, see e.g., [101]) *Every simple polygon admits a polygon triangulation. A simple polygon of n vertices may be partitioned into $n-2$ triangles by additional $n-3$ internal diagonals.*

Theorem 3.3.2 (see e.g., [101]) *The dual graph of a triangulation of a simple polygon forms a tree (see Figure 3.4).*

and repeating the operation with the remaining point set until no point is left

[2]For any planar graph $G = (V, E)$, we always have $|V| - |E| + |F| = 1$.

Theorem 3.3.3 (Meisters' Two Ears Theorem [89]) *Every simple polygon with $n \geq 4$ vertices has at least two non-overlapping ears. An* ear *of simple polygon P is a triangle such that one of its edges is a diagonal of P and the remaining two edges are edges of P.*

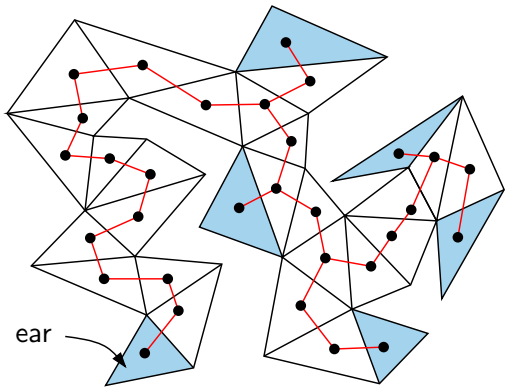


Figure 3.4: An example for Triangulation theorem, Theorem 3.3.2 and Meisters' Two Ears Theorem.

We define a simple polygonalization $\mathcal{P}$ for S, so that $\mathcal{P}$ is a simple polygon whose polygon vertices are precisely of S, or is a crossing-free Hamiltonian cycle of S. Now we can show somewhat a trivial fact that S always admits at least one simple polygonalization.

Lemma 3.3.4 *Let S be a set of planar points, where all points does not lie on a line. Then S always admits at least one simple polygonalization.*

Proof It is shown by induction with respect to the number of convex layers of a given point set S. Let k be the number of convex layers for S. Let $c_i = \{p_1^i, p_2^i, \ldots\}$ be the (closed) polygonal chain layered at level i in the convex layer of S, where $p_j^i,\ j = 1, 2, \ldots,$ are points counterclockwise ordered.

If $k = 1$, it is a trivial. When $k = 2$, there are a few cases for the number of points in c_1. If $|c_1| \leq 2$, then the statement follows since the polygonal chain $\{p_1^2, p_1^1, p_2^2, (p_2^1,) \ldots\}$ makes a simple polygonalization. We consider the case $|c_1| > 2$. If there exists a pair of line segment $s_1 = \overline{p_i^1 p_{i+1}^1}$ and $s_2 = \overline{p_j^2 p_{j+1}^2}$ such that s_1 and s_2 are mutually visible, then the polygonal chain $\{\ldots, p_j^2, p_i^1, p_{i-1}^1, \ldots, p_{i+2}^1, p_{i+1}^1, p_{j+1}^2, \ldots\}$ makes a simple polygonalization. Otherwise, there is no such mutually visible line segment pair. We pick any point p_i^1 in c_1, then the nearest line segment $\overline{p_j^2 p_{j+1}^2}$ is uniquely determined. Since $|c_1| \geq 3$ and $|c_2| \geq 3$, and then $\overline{p_{i-1}^1 p_{i+1}^1}$ and $\overline{p_j^2 p_{j+1}^2}$ are mutually visible, the polygonal chain $\{\ldots, p_j^2, p_i^1, p_{i-1}^1, \ldots, p_{i+2}^1, p_{i+1}^1, p_{j+1}^2, \ldots\}$ makes a simple polygonalization.

Now, we assume that the statement follows for any point set having $k-1$ convex layers. In addition, we also assume that the simple polygonalization is built by the above manner. Hence, the $k-1$th convex layer is considered as an opened convex polygonal chain c'_{k-1} consisting of (at least three) points in c_{k-1}. Let s^{k-1} be the line segment deleted from c_{k-1}. Without loss of generality, we can apply the above strategy to the kth layer, choosing a line segment in c_k which has mutually visible edge of c'_{k-1} if it exists; otherwise a line segment in c_k whose nearest point p_j^{k-1} in c'_{k-1} is neither of endpoints of s^{k-1}. This completes the proof. □

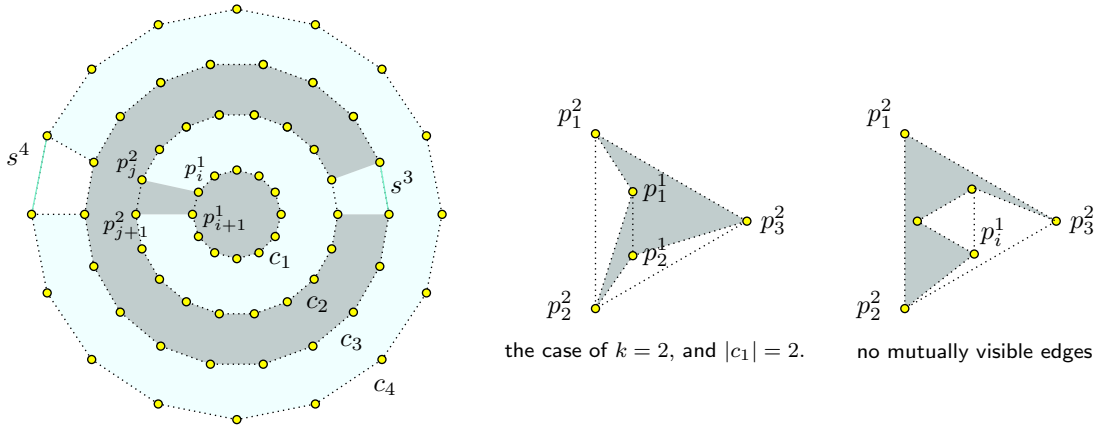


Figure 3.5: Examples for the proof of Lemma 3.3.4

Finally, we define a *polygon tree* on the dual of a triangulation $T(S)$ of planar points. This notation is conceptually basic idea in our heuristic algorithm. A polygon tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ on the dual $\mathcal{D}(T) = (V_\mathcal{D}, E_\mathcal{D})$ of a triangulation T is a tree such that $\cup_{v\in\mathcal{T}}\, g^{-1}(v)$ is a simple polygon, where g is bijection from a face in T to a vertex in $\mathcal{D}(T)$. Hence, we say a polygon tree $\mathcal{T}$ *maximal* if and only if appending any edge $e \in E_\mathcal{D} \setminus E_\mathcal{T}$ into $E_\mathcal{T}$ makes $\cup_{v\in\mathcal{T}}\, g^{-1}(v)$ nonsimple polygon. Figure 3.3 (c) depicts a maximal polygon tree and the dual polygon.

3.4 Heuristic Algorithm

In this section, we describe a triangulation-base heuristic algorithm for computing a simple polygonalization of S. We assume that a triangulation $T(S)$ is stored in a canonical data structure for maintaining *Planar Straight Line Graphs* such as *Halfedge* data structure [30] or doubly connected edge list [21]. Halfedge supports efficient local modifications with constant time such as edge-flipping operation. Note that we do not explicitly maintain $\mathcal{D}(T)$, since Halfedge provides efficient functions for the bijection between $T(S)$ with

$\mathcal{D}(T)$. Algorithm 5 shows an outline of our heuristic algorithm for computing a simple polygonalization.

Algorithm 5: Heuristic for computing a simple polygonalization

Input : A set S of points in general position in the plane

Output: A random simple polygonalization

1 Let initialize $T(S)$ with a randomly generated triangulation of S;

2 Construct the dual graph $\mathcal{D}(T)$ of $T(S)$;

3 Compute a random maximal polygon tree $\mathcal{T}$ on $\mathcal{D}(T)$;

4 Construct a simple polygon by traversing the tree $\mathcal{T}$ with depth-first search;

3.4.1 On generating a random triangulation

In the first line of Algorithm 5, we generate a random triangulation $T(S)$. Recently, considering a random triangulation has received various attentions. For instance, Sharir and Welzl [115] show several results on the numbers of planar triangulations by using some properties of random triangulations. However, it does not seem that many have been known about generating a random triangulation, although Epstein and Sack [53] propose efficient $O(n^3)$ and $O(n^4)$ time algorithms for counting triangulations and generating a random one of a given simple polygon with n vertices, respectively. Aichholzer [3, Section 4.3] suggests an idea for generating a random triangulation: first, enumerate all possible triangulations, and number them in generating order; next, generate a random number $r \in \{i\}_{i=1}^{t(S)}$, where $t(S)$ is the number of triangulations of S; finally, report the r-th triangulation. It seems to be expensive to compute all triangulations for larger instances.

Our idea of generating a random triangulation is to perform random edge-flipping operations. However, it is a folklore open problem to determine the mixing rate of the Markov process that starts at some triangulation and keeps flipping a random flippable edge; see [87, 92] where this is treated for points in convex position. In fact, the mixing rate of triangulation of planar n points in convex position is bounded by $O(n^4 \log n)$.

This leads that our $O(n \log n + f)$ heuristic algorithm has the same time complexity as that of 2-opt Moves even in the special case for which all planar points are in a convex position. However, we can generate any triangulation of S with a positive probability by performing random $O(n^2)$ edge-flipping operations in the worst case. Since the expected number of convex layers for n uniformly and independently distributed points is $\Theta(n^{2/3})$, due to Dalal [39], random $O(n^{5/3})$ edge-flipping operations may be required in the average case. Therefore, when we stand for the sense that we want to generate a polygon as an

instance, or do not require fairness of generated polygons, we can consider our heuristic algorithm has advantage to 2-opt Moves.

3.4.2 Computing a random polygon tree

We describe a procedure for computing a random polygon tree $\mathcal{T}$ on the dual $\mathcal{D}(T) = (V_{\mathcal{D}}, E_{\mathcal{D}})$ of triangulation T of S. Algorithm 6 is a pseudo-code for computing a random polygon tree $\mathcal{T}$. Throughout the Algorithm 6, we maintain two sets X and Y: X is a set of vertices in $V_{\mathcal{D}}$ visited so far; Y is a set of current candidate edges in $E_{\mathcal{D}}$ which connect between vertices $v^* \in X$ and $w^* \in V_{\mathcal{D}} \setminus X$. In the line 6 – 14, the procedure augments the current polygon tree by adding an appropriate edge one at a time. An edge $(v^*, w^*) \in E_{\mathcal{D}}$ is admitted as of polygon tree $\mathcal{T}$, where $v^* \in X$ and $w^* \notin X$, if a triangle $g^{-1}(w^*)$ has a vertex marked unvisited. Otherwise, the union of triangles $\cup_{v^* \in X \cup \{w^*\}} g^{-1}(v^*)$ induces a nonsimple polygon.

We can show a few properties of polygon tree generated by Algorithm 6.

Lemma 3.4.1 *A subgraph $\mathcal{T}$ of $\mathcal{D}(T)$ which is generated by Algorithm 6 is a maximal polygon tree.*

Proof It is obviously that $\mathcal{T}$ is connected, since the procedure grows the current polygon tree by adding an edge incident to a vertex in X with the other in $V_{\mathcal{D}} \setminus X$ one by one.

It can be completed by showing there is no cycle in $\mathcal{T}$. When we assume that $\mathcal{T}$ contains a cycle, intermediately we have a contradiction. To construct a cycle, we have to violate at least two times for the condition of line 8 in Algorithm 6.

When the while-loop, i.e. at the line 14, ends, the maximality is satisfied since the procedure has tried each adjacent triangles to check whether or not it is admitted. □

Unfortunately, there exist undesirable situations for which this procedure goes into a deadlock: the simple polygonalization induced by the reported maximal polygon tree of Algorithm 6 cannot cover all given points. Figure3.6 shows the undesirable situation. Therefore, Algorithm 5 is a Monte-Carlo algorithm [93] which does not guarantee always to compute a feasible simple polygonalization of S.

However, our heuristics can generate each possible simple polygonalization when it does not return false.

Lemma 3.4.2 *Algorithm 5 generates each possible simple polygonalization of given set S of planar points.*

Proof Let P be an arbitrary simple polygonalization of S. We consider a triangulation $T(S)$ which contains a triangulation $T(P)$ of P as a subgraph.

Algorithm 6: Computing a random polygon tree $\mathcal{T}$

Input : A triangulation $T(S) = (V, E)$ and its dual graph $\mathcal{D}(T) = (V_{\mathcal{D}}, E_{\mathcal{D}})$.

Output: A random polygon tree $\mathcal{T}$.

```
1  forall v ∈ V do mark v as the label unvisited ;
2  forall e* ∈ E_D do mark e* as the label dual_edge ;
3  Randomly choose a vertex v* ∈ V_D and mark to all vertices of triangle g^{-1}(v*) as
   visited ;
4  X ← {v*};
5  Y ← {(v*, w*) ∈ E_D | v* ∈ X,  w* ∈ V_D \ X};
6  while Y ≠ ∅ do
7      Randomly choose an edge e* = (v*, w*) from Y, and Y ← V \ {e*};
       (*Assumption:  Assumption:  v* ∈ X,  w* ∉ X.  *)
8      if g^{-1}(w*) has a vertex v marked unvisited then
9          Mark visited to v;
10         Mark polygon_edge to e*;
11         Y ← Y ∪ {edges in E_D incident to w* but not e*};
12     X ← X ∪ {w*};
13 if |X| ≠ |V_D| then return false;
14 else
15     E_T ← {e* ∈ E_D | e* is marked polygon_edge};
16     V_T ← {v* ∈ V_D | v* is a vertex of e* ∈ E_T};
17     return T = (V_T, E_T);
```

From Theorem 3.3.3, $T(P)$ has at least two ears for $n > 3$. Let v_i, v_{i+1} and v_{i+2} be one of the ears in $T(P)$. Cutting the ear from $T(P)$, we obtain a $(n-1)$-gon $(v_1, v_2, \ldots, v_i, v_{i+2}, \ldots, v_n)$. We repeat this procedure until P converges to a triangle (v_a, v_b, v_c) while maintaining the order performing ear-cutting operations. We can obtain a simple polygonalization P, when Algorithm 6 is executed as follows: starting at the vertex corresponding to triangle (v_a, v_b, v_c); and growing the current polygon tree in the reverse order of ear-cuttings. Note that Algorithm 6 computes a polygon tree $\mathcal{T}$ without going into a deadlock. Hence this completes the proof. □

Now, we estimate the time and space complexities of Algorithm 5. We can see that the space complexity is $O(n)$ for given set of n planar points from observations in Section 3.3. For the time complexity, the most expensive part is to compute a triangulation of S, or to perform random flipping-edges, in Algorithm 5, We assume that the number f of

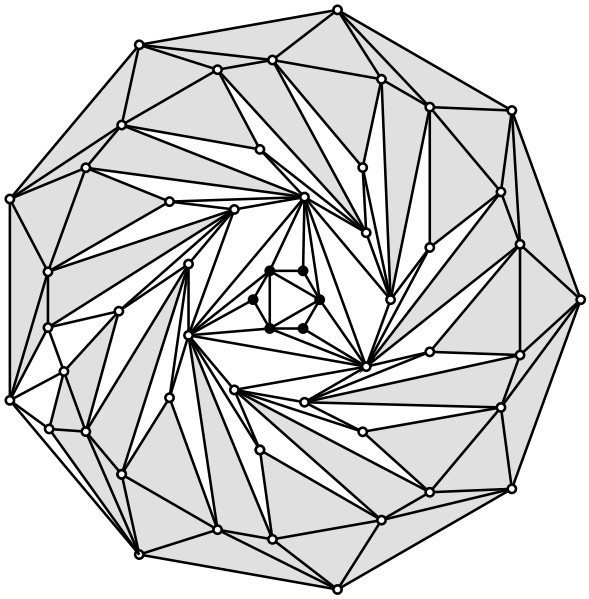


Figure 3.6: An example in which Algorithm 6 goes into a deadlock; there exist points in S which cannot be covered by resulting simple polygonalization. White points are the vertices of the resulting simple polygon, Black points are unvisited by Algorithm 6.

edge-flipping operations are given in advance. Since there are a number of algorithms for constructing a triangulation $T(S)$ in the optimal time $\Theta(n \log n)$ in the worst case (see e.g., [106, 21]), the time complexity is $O(n \log n + f)$.

Theorem 3.4.3 *Algorithm 5 can compute a simple polygonalization of given set of n planar points in $O(n \log n + f)$ time when it does not returns* false.

3.4.3 The simple salvage procedure

To salvage isolated points, we firstly apply a simple refinement procedure for the current plane triangulation $T(S)$ and the incomplete simple polygon P. Let I be the set of isolated points or uncovered points in Algorithm 5. This procedure ensures that each points uncovered in Algorithm 5 can be covered as many points in I as possible by locally modifying the current triangulation $T(S)$. Since an isolated point $v_i \in I$ can be salvaged if p has at least one fully visible polygon edge (v, w) in the current simple polygon P and the triangle (v, v_i, w) does not contain any other isolated points, we simply find such a polygon edge in P for each isolated vertex. Algorithm 7 shows the pseudo code that finds an fully visible polygon edge for each isolated vertex. Roughly speaking, it is obvious that the time complexity of Algorithm 7 is $O(n)$ for each $p \in I$, since $T(S)$ is maintained by Halfedge data structure, and getting a local information or updating local topology in

$T(S)$ can be done in $O(1)$ time.

Algorithm 7: Simple Salvaging Procedure

Data : Triangulation $T(S)$, current simple polygon P, and the isolated points I.
Result: A simple polygonalization P' which is salvaged as many as possible by greedy searching polygon edges.

```
1  P' ← P;
2  forall v_i ∈ I do
3      Find all triangles Δ_i := {(v_i, v', v'')} which are incident to point v_i in T(S);
4      while Δ_i ≠ ∅ do
5          Let δ_j := (v_i, v', v'') be a triangle in Δ_i, and Δ_i ← Δ_i \ {δ_j};
6          Find a triangle δ'_j := (v'', v', x) sharing edge (v', v'');
7          if the quadrilateral δ_j ∪ δ'_j is flippable then
8              Update the current triangulation T(S) by flipping edge (v', v'') and
               obtain new two triangles δ_1 = (v, v', x) and δ_2 = (v, x, v'');
9              if either δ_1 or δ_2 shares an edge with the current polygon P' then
10                 P' ← P' ∪ δ_1 (resp. δ_2);
11                 Terminate salvage procedure for v_i;
12             else
13                 Δ_i ← Δ_i ∪ {δ_1, δ_2};
```

Figure 3.7 depicts the behavior of Algorithm 7; the left figure is an initial condition, and the right figure is a successful result obtained by applying Algorithm 7. Unfortunately, we cannot always obtain a simple polygonalization of S by applying only for Algorithm 7, since there exist undesirable conditions so that an isolated point has no such visible polygon edge. In addition, since the simple polygon P' is updated while performing Algorithm 7, it may happens that an isolated point p has initially a visible edge e but e will become invisible from p. Therefore, we have to take care for not only modifying triangulation $T(S)$ but also modifying the current simple polygon P'. However, even if we can obtain the algorithm for completely computing a simple polygonalization of S, it seems to be difficult to show the correctness of the algorithm, since it is too complex to analyze the terminating condition. We trade in proposing a complete salvaging procedure for showing experimental results to make sure the practical efficiency of Algorithm 5 with Algorithm 7 in Section 3.5.

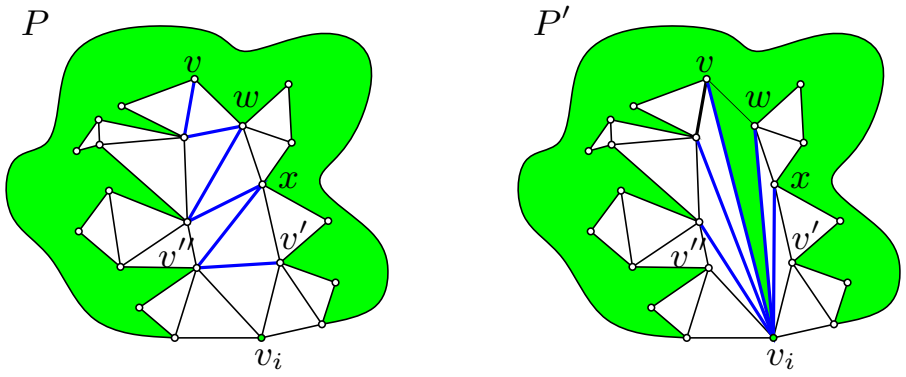


Figure 3.7: Example for the behavior of Algorithm 7.

3.5 Experimental Results

We have implemented Algorithm 5 and Algorithm 7 to evaluate the efficiency and usefulness. Table 3.1 describes our environment of the experiment. We coded Algorithm 5 with Algorithm 7 in LEDA [94]. To compare the time complexity over the well-known and widely used algorithm "2-opt Moves," we used program; `random_polygon_2` included in CGAL [25]. Both libraries LEDA & CGAL are C++ class libraries.

Table 3.1: The environment of experiment

CPU	Main memory
Intel ® Pentium III 870 MHz	256 MB
OS	External libraries
FreeBSD 5.4	CGAL-3.1 & LEDA-4.2

Characteristics for resulting polygons We start to estimate characteristics of resulting simple polygons. It is important to be able to control types of generated polygons, when we consider testing efficiencies of geometric computation problems which deal with simple polygons. Actually, our heuristic algorithm generates simple polygons which reflect the shape in base triangulations. Figure 3.8 and Figure 3.9 show two different types of simple polygons for the same point set. The triangulation in Figure 3.8 is computed by sweeping techniques from left to right, and one in Figure 3.9 is Delaunay triangulation.

To see the flexibility in our heuristic algorithm handling types of simple polygons, we consider a kind of degree of simple polygon. The winding number or revolution number for a simple polygon P is one of the important factors which figure out a simple polygon, since it may sometimes act on the time complexity and the correctness of geometric algorithm.

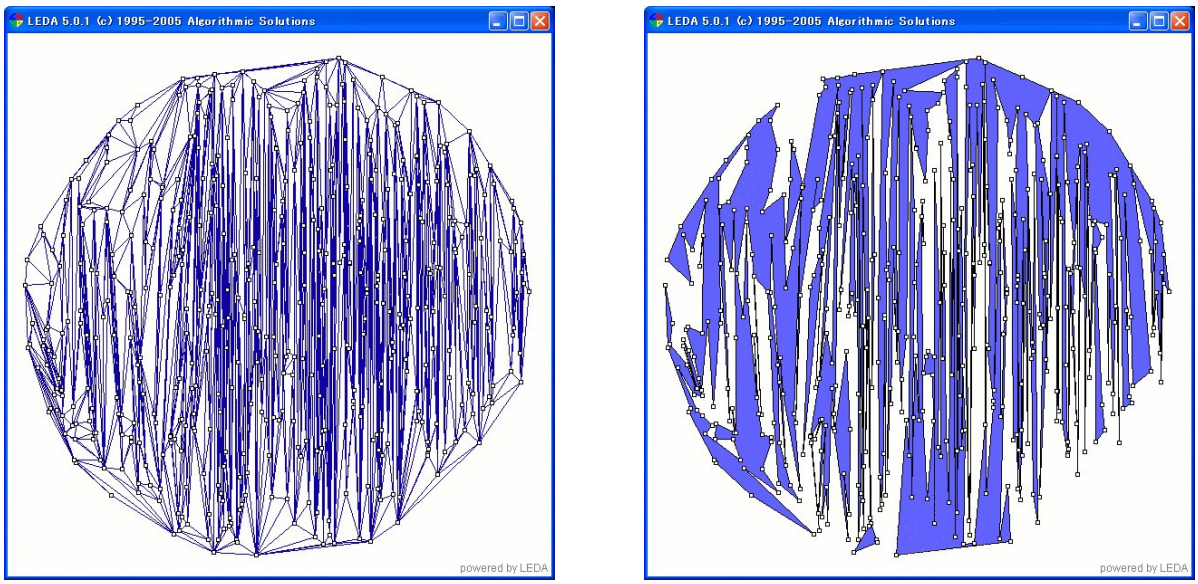


Figure 3.8: A resulting simple polygon based on a skinny triangulation.

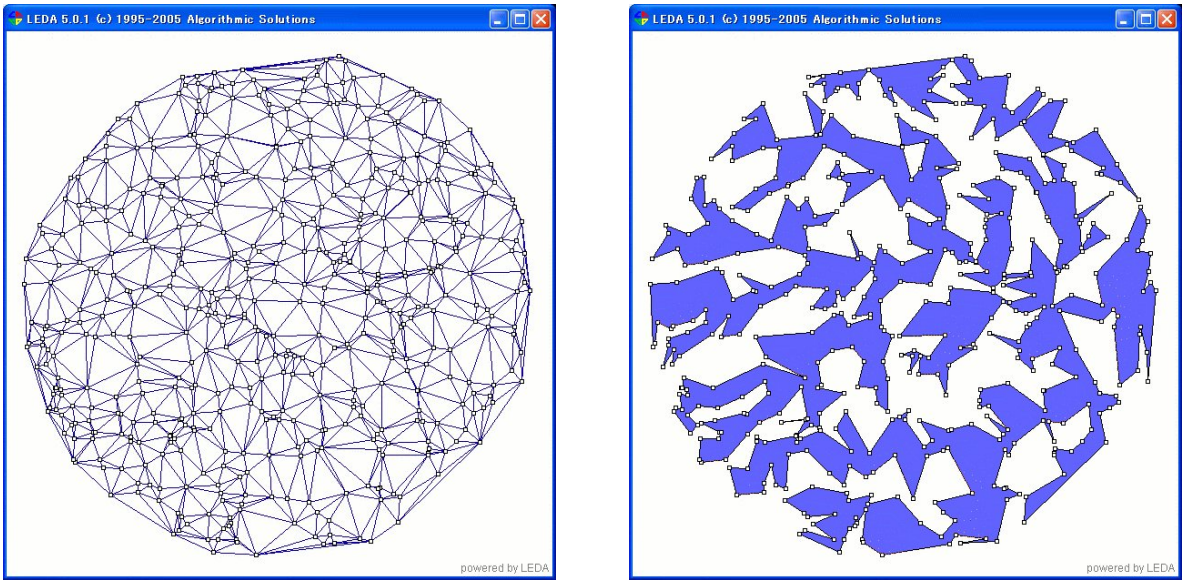


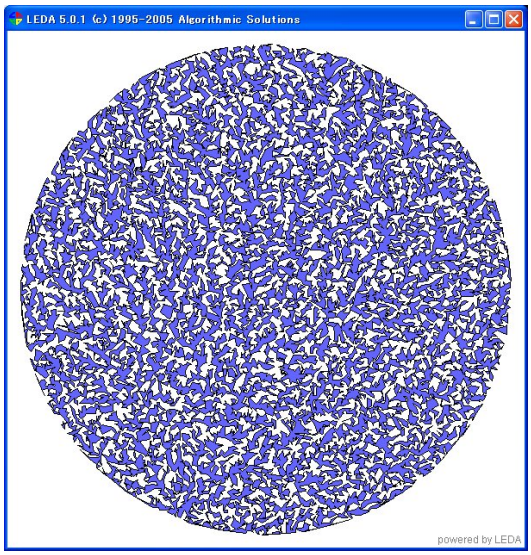
Figure 3.9: A resulting simple polygon based on a fat triangulation; Delaunay triangulation.

For example, some algorithms for computing a visibility polygon from a point in a simple polygon [50, 22], or for computing a triangulation in a simple polygon [27, 51, 60, 74, 67] come under the influence of winding numbers. The *winding number* of a point p with respect to a simple polygon P is the number of revolutions that the boundary of P makes about p : the total signed angular turn (with the usual convention: counterclockwise turns are positive, and clockwise turns negative) divided by 2π. This notion is found in

[32, 58, 102].

To emphasize generating a simple polygon with higher winding number, we shift away randomly chosen edges of line 7 in Algorithm 6, and simply chosen a edge from LIFO list, or stack data structure. Figure 3.10 shows an example of experimental results for $n = 5000$; the left image is obtained random selection, the right LIFO selection. In the right figure in Figure 3.10, a point in the simple polygon at the center of window can have higher winding number.

Random edge-choosing



LIFO edge-choosing

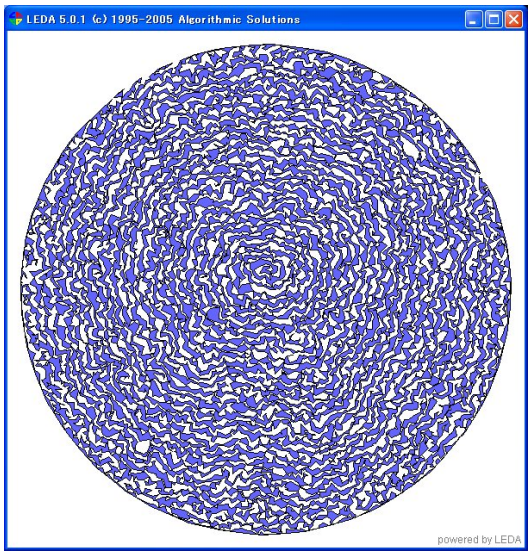


Figure 3.10: Example of generating a simple polygon with higher winding number.

Running times We show our heuristic algorithm much faster than the program included in CGAL. Note that our heuristic algorithm may fail in one procedure call, so we evaluate the time until the heuristics correctly computes a simple polygonalization. Actually, we could know that our heuristic algorithm does not so often fail with Algorithm 7. In this experiment, the computational accuracy of coordinate systems in the both algorithms are represented by `double` data type.

Figure 3.11 shows the graph of experimental results with 2-opt Moves with our heuristic algorithm. The abscissa axis represents the number of points S with logarithmic scale. The longitudinal axis represents the second of average running time per 5 trials for each number of points. In Figure 3.11, we do not show the running time of 2-opt Moves for 400 points or later, since 2-opt Moves consumes high cost and resources. As in rough sketches, 2-opt Moves program take about 3.5 (sec), 8 (min), and 70 (min) for $n = 250, 2000$, and 5000, respectively. On the other hand, our heuristic algorithm takes about 0.01 (sec), 0.1

(sec), and 0.3 (sec), for similar instance sizes, respectively. Moreover, we experiment for more larger instances with our heuristics. Table 3.2 shows the summary of computation times of our heuristic algorithm. As a disadvantage, although our algorithm requires $O(n)$ space, maintaining Halfedge data structure may be expensive in this computation environment, to compute simple polygonalizations for 500,000 points or later.

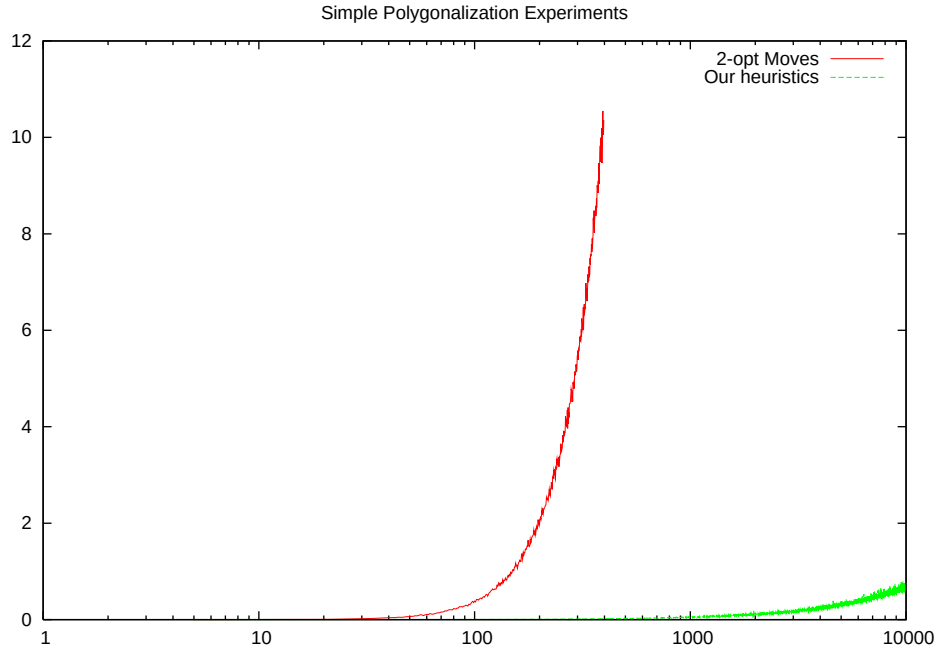


Figure 3.11: Experimental results on running time: 2-opt Moves v.s. Our heuristics.

Table 3.2: The average running time of our heuristic algorithm for larger instances

$\|S\|$	1000	2000	5000	10000	50000	100000	200000
Time(sec)	0.0515	0.1094	0.3016	0.5969	3.78125	7.97187	18.15

Quality assessment We evaluated the experimental probability in which the number of each generated simple polygons against total number of generated simple polygons for up to about 2.0×10^9 trials.

Figure 3.12 depicts an experimental result with respect to the rate of the counting number for each generated simple polygonalizations of `set15` released in Triangulation Olympics [4]. In Figure 3.12, the abscissa axis represents the number of generations, and the longitudinal axis represents the number of simple polygonalizations. We perform 2.1×10^9 trials, and correctly obtained 2,097,292,568 simple polygonalizations,

and 4,311,771 different simple polygonalizations. In this result, 628,615 different simple polygonalizations are just once, however one of generated polygonalizations is counted 5,181,305 times. This implies some of simple polygonalizations have high possibility to be generated.

As the other experiments, each number shown in Figure 3.13 is the counting number of each simple polygonalization in 100,000 trials. Actually, the randomness for generating simple polygonalizations is not uniform: the lowest probability is 0.00561, and the largest probability 0.102.

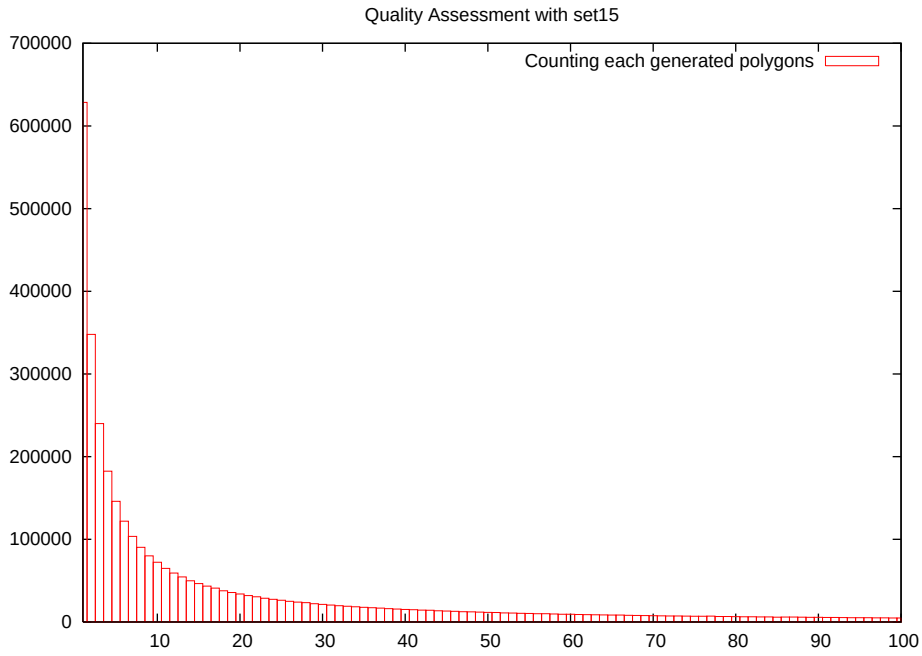


Figure 3.12: Quality assessment for set15 in Triangulation Olympics.

On counting simple polygonalizations The number of simple polygonalizations for planar points has been studied assiduously in discrete geometry. However there are many open problems as shown in [23, 42]. One of the reasons for the difficulty is that the number of simple polygonalizations increases exponentially with respect to the number n of points. Let $simple(n)$ be the maximum number of simple polygonalizations for any n-point set. The function $simple(n)$ is easily seen to satisfy the following inequality:

$$simple(a+b) \geq simple(a)simple(b),$$

so the limit $\lim_{n\to\infty} (simple(n))^{1/n}$ exists (it is finite by [6].) The currently best bounds [64, 114, 7] are

$$4.642 \leq \lim_{n\to\infty} (simple(n))^{\frac{1}{n}} \leq 87.$$

The upper bound follows by combining the bound on the number of triangulations of a set by Sharir and Welzl [114] with the bound on the number of simple cycles within a triangulation by Alt, Fuchs, and Kriegel [7]. With the help of computers, exact values have been determined for $n \leq 10$ by Aichholzer and Krasser [5] (see Table 3.3).

Table 3.3: The exact values simple(n) for $n \leq 10$.

n	3	4	5	6	7	8	9	10
simple(n)	1	3	8	29	92	339	1282	4994

Figure 3.13 shows all simple polygonalizations for 6 points achieving Each simple polygonalization in Figure 3.13 is obtained by the experiments.

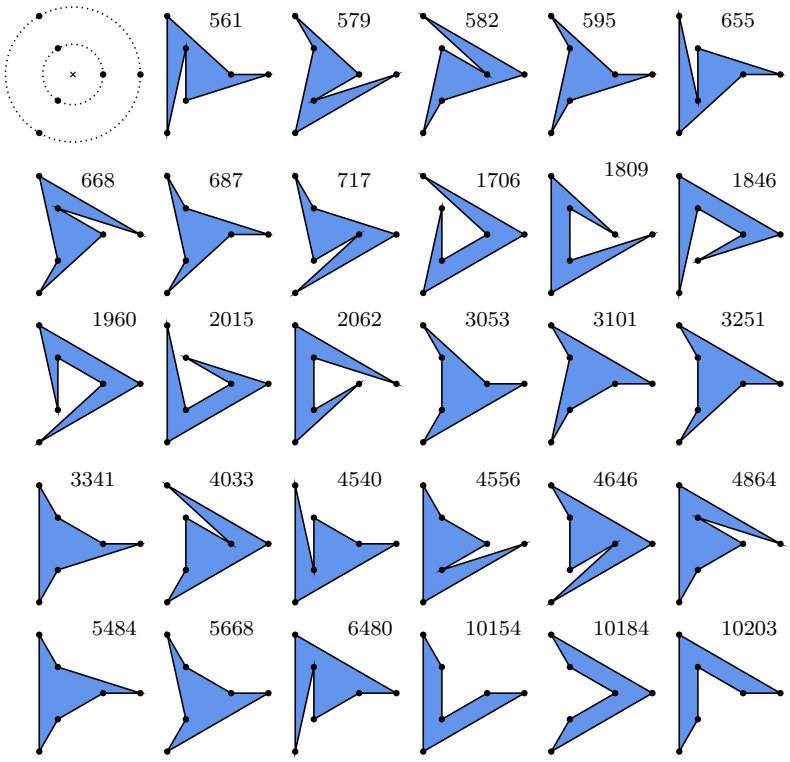


Figure 3.13: Example of a 6-point set which maximizes the number of simple polygonalizations. The number for each simple polygonalization depicts counting numbers for 100,000 trials.

We are interested in point sets released at "Counting Triangulations – Olympics" [4] by Aichholzer. The coordinates of points for each point set is also found at the web site. As mentioned above, since the bounds of the number of simple polygonalizations is followed by combining the number of triangulations and the number of simple cycles, we

would like to compare the number between triangulation with simple polygonalizations. In this experiments, the computational accuracy is critical to exact counting, so that we use `rational` data type in LEDA. Hence, the resulting number is that the number of simple polygonalizations generated correctly. Firstly, we experiment up to $n = 14$ points by applying our algorithm up to 2.0×10^9 trials for each. Table 3.4 shows the experimental results. From obtained result, we can find out that our algorithm can compute correct simple polygonalizations with about 97%. In fact, the success ratio depends on the arrangement of given point sets. However, the size of well-distributed instance becomes larger and larger, the ratio of failure in our algorithm decreases. Hence, we can rely on the resulting numbers of simple polygonalizations up to 10 points.

Table 3.4: Comparing with Triangulations & Simple Polygonalizations

$\|S\|$	Name	Triangulation	Polygon	Name	Triangulation	Polygon
4	dcirc4	1	3	convex4	2	1
5	dcirc5	2	6	convex5	5	1
6	dcirc6	4	16	convex6	14	1
7	dcirc7	11	30	convex7	42	1
8	dcirc8	30	41	swrpp8	150	335
9	dcirc9	89	73	swrpp9	780	1,252
10	dcirc10	250	407	swrpp10	4,550	4,956
11	dcirc11	776	1,717	set11	26,888	20,887
12	dcirc12	2,236	3,174	set12	168,942	82,421
13	dcirc13	7,147	9,779	set13	1,098,904	363,884
14	dcirc14	20,147	18,047	set14	7,281,984	1,540,451

3.6 Some genelarized simple polygonalizations

We consider the following extension, we call Simple k-Polygonalizations Problem. Given a set S of n points in the plane, generate a set of disjoint simple polygons $\{P_1, P_2, \ldots, P_k\}$ such that the union of all vertices of each P_i is exactly S. The random simple polygonalizations may be an instance of Robot Motion Planning Problems (see e.g., [21, Chapter 13, 15]). We could have a few preliminary results so that our heuristic approach is applicable to this problem with empirical study.

Algorithm 8 shows our idea for this generalized problem. First, we compute a random triangulation $T(S)$ of S. This step is the same as in Algorithm 6. Second, we consider

the dual $\mathcal{D}(T) = (V_{\mathcal{D}}, E_{\mathcal{D}})$ and the *map graph* $\mathcal{M}(T) = (V_{\mathcal{D}}, E_{\mathcal{M}})$ of $T(S)$. Map graph is a generalized dual of planar graph G in which two regions of G are adjacent when they share any vertex of their boundaries (not an edge, as standard planarity requires), see Chen, Grigni, and Papadimitriou [29].

Third, we compute an independent set $I_{\mathcal{M}(T)} = \{v_1^*, \ldots, v_k^*\}$ on the map graph $\mathcal{M}(T)$. Note that $g^{-1}(v_i^*) \cup g^{-1}(v_j^*) = \emptyset$ for any two different vertices $v_i^*, v_j^* \in I_{\mathcal{M}(T)}$. However, this phase is computationally quite difficult, since the problem is sub-problem for finding maximum independent set (MIS) on map graph. Indeed, it is shown that the problem MIS on a map graph is NP-hard due to Chen who also proposed approximation algorithms [28], and PTAS combining the results of Thorup [119]. Hence, we relaxed for the number k of simple polygons, i.e., shifting our objective, generating at most k simple polygons is considered.

Finally, we compute a random maximal polygon forest $\mathcal{F}$ which is defined as a set of maximal polygon trees. The maximal polygon forest can be computed by Algorithm 6 with $X = I_{\mathcal{M}(T)}$ and $Y = \{(v^*, w^*) \in E_{\mathcal{D}} \mid v^* \in X, w^* \in V_{\mathcal{D}} \setminus X\}$.

In the simple polygonalizations problem, Algorithm 8 can always generate a number of simple polygons, since there is no deadlocks if we restart growing a new polygon tree from $v^* \notin X$ and $g^{-1}(v^*)$ has an unvisited vertex.

Algorithm 8: Heuristic for computing simple polygonalizations

Input : A set S of points in general position in the plane

Output: A random simple polygonalization

1 Initialize $T(S)$ with a randomly generated triangulation of S;
2 Construct the dual graph $\mathcal{D}(T)$ and the map graph $\mathcal{M}(T)$ of $T(S)$;
3 Compute a random independent set $I_{\mathcal{M}}(T) = \{v_1^*, \ldots, v_k^*\}$ on $\mathcal{M}(T)$;
4 Compute a random maximal polygon forest $\mathcal{F}$ on $\mathcal{M}(T)$;
5 Construct simple polygons by traversing each tree in $\mathcal{F}$ with depth-first searches;

This heuristic algorithm solves a relaxed problem proposed as a future work by Auer and Held [15]: Given a set S of n points and a natural number $k \leq \frac{n}{3}$, generate k random polygons on S. If we apply our idea to this problem, we must solve a difficult problem for generating an independent set on a map graph whose cardinality is just k. Figure 3.14 shows examples for generating k simple polygons on a given point set S.

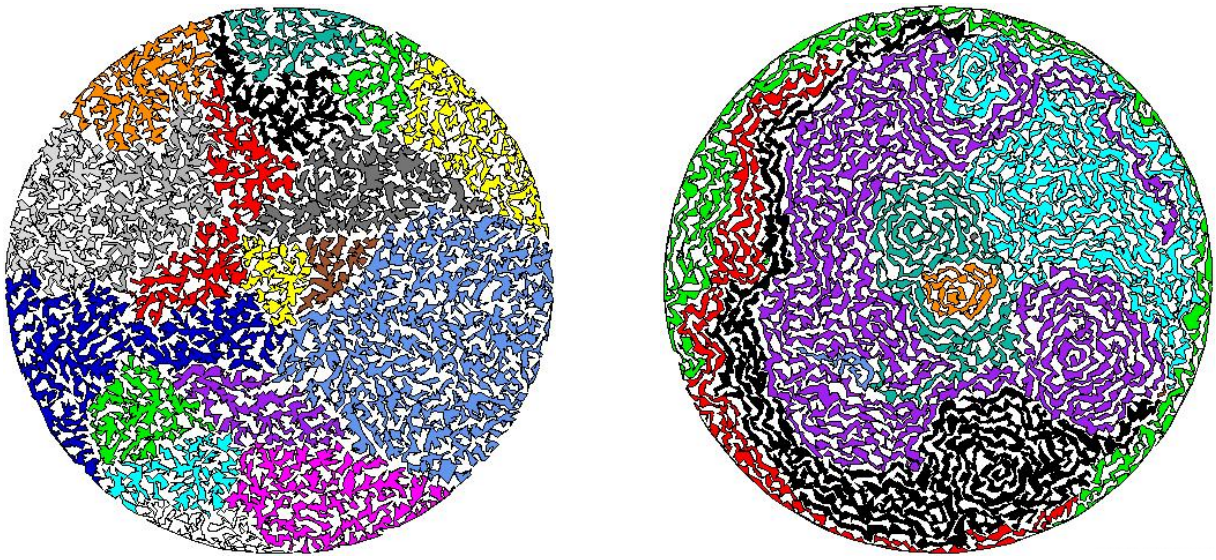


Figure 3.14: Examples for generating k simple polygonalizations: $n = 15,000$ and $k = 20$.

3.7 Concluding remarks and Future works

We have presented a triangulation-base heuristic algorithm for computing a simple polygonalization of given planar point set in $O(n \log n + f)$ time, where f is the number of edge-flipping operations. Our heuristic algorithm is Monte-Carlo Algorithm, that is, it has undesirable situations. However, almost all such undesirable situations can be removed with proposed simple refinement procedure. In actual, we have found our that this statement is followed from results in computer experiments. Moreover, the running time of our algorithm has an advantage against well-known and widely used algorithm, 2-opt Moves.

One of the most important future works is to guarantee that Algorithm 6 always can compute a simple polygonalization of S. In other words, it may be necessary to backtrack and restore while computing a maximal polygon tree if it is trapped by a deadlock. We have designed a recursive procedure to restart growing polygon tree, but there still exist undesirable situations.

In the simple polygonalizations problem, we are also interested in designing an efficient algorithm for enumerating maximal independent sets in the map graph, although there exists an algorithm for generating all maximal independent sets in a polynomial delay [78].

Chapter 4

Equilibrium and Disequilibrium in Spatial Competition

4.1 Introduction

4.1.1 Competitive facility location

In this chapter, we study combinatorial game models in spatial competition, and propose several strategies in the graph models. *Spatial competition* is one of the most realistic branch of location theory for spatial economics and industrial organization [8, 99], mathematics [69], and operations research [49, 83, 120]. For a survey of various competitive facility location models see [45, 46, 47, 48, 68, 70].

Facility location models deal mainly with the location of plants, warehouses and other industrial facilities. One branch of location theory deals with the location of retail and other commercial facilities which operate in a competitive environment. The facilities compete for customers and market share, with a profit maximization objective. The customary objective function to be maximized is the market share captured by the facilities. All competitive location models attempt to estimate the market share captured by each competing facility in order to optimize its location. The best location for a new facility is at the point at which its market share is maximized. Typically, the objective to be optimized is the maximum distance from customers to the facility–this results in the minimum enclosing disk problem studied by Megiddo [88], Welzl [125] and Aronov et al. [9].

We consider a model where the behavior of the customers is deterministic in the sense that a facility can determine the set of customers more attracted to it than to any other facility. We assume that customers in the underlying space are uniformly distributed. This implies each facility captures a constant market area. The collection of market

areas forms a tessellation of the underlying space. If customers choose the facility on the basis of distance in some metric, the tessellation is the Voronoi diagram of the set of facilities [100, 17]

4.1.2 Combinatorial game theory

Combinatorial games lead to several interesting, clean problems in algorithms and complexity theory, many of which remain open. The complexity of generalized versions of popular games and puzzles has been studied. Indeed, many classic games are known to be computationally intractable: one-player puzzles are often NP-complete, and two-player games are often PSPACE-complete or EXPTIME-complete. The complexity classes are comprehensively summarized by Papadimitriou [104], Arora & Barak [11], and so on.

For example, the problems of determining the winner are shown to be exponential-time complete for generalized Chess [61], Checkers [110], Go [109], and Shogi [1]; the problems of determining the winner or whether there is a solution are PSPACE-complete for generalized Hex [54, 108], Gomoku [107], Othello [76], Sokoban [38], and Sliding-Block (箱入り娘 in Japanese) [72]; and the problems of determining whether there is a solution in generalized Hi-Q (peg-solitaire) [121], minesweeper [80, 81], and Tetris [24] are shown to be NP-complete. Surprisingly, many seemingly simple puzzles and games are also hard. Demaine [40] and Eppstein [52] surveyed a more comprehensive results in the algorithmic combinatorial game theory [40].

A *combinatorial game* typically involves two players, say $\mathcal{W}$ (white) and $\mathcal{B}$ (black), alternating play in well-defined *moves*. However, in the interesting case of a *combinatorial puzzle*, there is only one player, and for *cellular automata* such as Conway's Game of Life, there are no players. In all cases, no randomness or hidden information is permitted: all players know all information about gameplay (*perfect information*). The problem is thus purely strategic: how to best play the game against an ideal opponent.

Combinatorial game theory is to be distinguished from other forms of game theory arising in the context of economics. Economic game theory has applications in computer science as well, most notably in the context of auctions [43, 57] and analyzing behavior on the Internet [105].

It is useful to distinguish several types of two-player perfect-information games [19, pp. 16–17]. A common assumption is that the game terminates after a finite number of moves (the game is *finite* or *short*), and result is a unique winner. Of course, there are exceptions: some games (such as Chess) can be *drawn* out forever, and some games (such as tic-tac-toe and Chess) define *tie* in certain cases. However, in the combinatorial-game setting, it is useful to define the *winner* as the last player who is able move; this is called

normal play. (We will normally assume normal play.) A game is *loopy* if it is possible to return to previously seen positions (as in Chess, for example). Finally, a game is called *impartial* if the two players are treated identically, that is, each player has the same moves available from the same game position; otherwise the game is called *partisan*.

A particular two-player perfect-information game without ties or draws can have one of four *outcomes* as the result of ideal play: player $\mathcal{W}$ winds, player $\mathcal{B}$ wins, the first player to move wins, or the second player to move wins. One goal in analyzing two-player games is to determine the outcome as one of these four categories, and to find a strategy for the winning player to win. Another goal is to compute a deeper structure to games, called the *value* of the game.

A beautiful mathematical theory has been developed for analyzing two-player combinatorial games. The most comprehensive reference is the book *Winning Ways* by Berlekamp, Conway, and Guy [19], but a more mathematical presentation is the book *On Numbers and Games* by Conway [34]. See also [35, 63] for overviews and [62] for a bibliography. The basic idea behind the theory is simple: a two-player game can be described by a rooted tree, each node having zero or more *left* branches correspond to options for player $\mathcal{W}$ to move and zero or more *right* branches corresponding to options for player $\mathcal{B}$ to move; leaves corresponding to finished games, the winner being determined by either normal or misére play. The interesting parts of combinatorial game theory are the several methods for manipulating and analyzing such games/trees.

4.2 Summary

The Voronoi game is motivated as an idealized model for competitive facility location, which was proposed by Ahn, Cheng, Cheong, Golin, and Oostrum [2]. The Voronoi game is played on a bounded continuous arena by two players. Two players $\mathcal{W}$ (white) and $\mathcal{B}$ (black) put n points alternately, and the continuous field is subdivided according to the *nearest neighbor rule*. At the final step, the player who dominates the larger area wins.

The Voronoi game is a natural game, but the general case seems to be very hard to analyze from the theoretical point of view. Hence, in [2], Ahn et al. investigated the case that the game field is a bounded 1-dimensional continuous domain. On the other hand, Cheong, Har-Peled, Linial, and Matoušek [31], and Fekete and Meijer [56] deal with a 2- or higher-dimensional case, but they restrict themselves to the one-round game; first, $\mathcal{W}$ puts all n points, and next $\mathcal{B}$ puts all n points.

In this paper, we introduce the *discrete* Voronoi game. Two players alternately occupy n vertices on a graph, which is a bounded discrete arena. (Hence the graph contains at

least $2n$ vertices.) This restriction seems to be appropriate since real estates are already bounded in general, and we have to build shops in the bounded area. More precisely, the discrete Voronoi game is played on a given finite graph G, instead of a bounded continuous arena. Each vertex of G can be assigned to the nearest vertices occupied by $\mathcal{W}$ or $\mathcal{B}$, according to the *nearest neighbor rule*. (Hence a vertex can be a "tie" when it has the same distance from a vertex occupied by $\mathcal{W}$ and another vertex occupied by $\mathcal{B}$.) Finally, the player who dominates larger area (or a larger number of vertices) wins. We note that the two players can tie in some cases.

We first consider the case that the graph G is a complete k-ary tree. A complete k-ary tree is a natural generalization of a path which is the discrete analogy of 1-dimensional continuous domain. We also mention that complete k-ary trees form a very natural and nontrivial graph class. In [2], Ahn et al. showed that the second player $\mathcal{B}$ has an advantage on a 1-dimensional continuous domain. In contrast to this fact, we first show that the first player $\mathcal{W}$ has an advantage for the discrete Voronoi game on a complete k-ary tree, when the tree is sufficiently large (comparing to n and k). More precisely, we show that $\mathcal{W}$ has a winning strategy if (1) $2n \leq k$, or (2) k is odd and the complete k-ary tree contains at least $4n^2$ vertices. On the other hand, when k is even and $2n > k$, two players tie if they do their best.

Next, we show computational hardness results for the discrete Voronoi game. When we admit a general graph as a game arena, the discrete Voronoi game becomes intractable even in the following strongly restricted case: the game arena is an arbitrary graph, the first player $\mathcal{W}$ occupies just one vertex which is predetermined, the second player $\mathcal{B}$ occupies n vertices in any way. The decision problem for the strongly restricted discrete Voronoi game is defined as follows: determine whether $\mathcal{B}$ has a winning strategy for given graph G with the occupied vertex by $\mathcal{W}$. This restricted case seems to be advantageous for $\mathcal{B}$. However, the decision problem is NP-complete. This result is also quite different from the previously known results in the 2- or higher-dimensional problem (e.g., $\mathcal{B}$ can always dominate the fraction $\frac{1}{2} + \varepsilon$ of the 2- or higher-dimensional domain) by Cheong et al. [31] and Fekete and Meijer [56]. However, Fekete and Meijer [56] showed that maximizing the area $\mathcal{B}$ can claim is NP-hard in the one-round game in which the given arena is a polygon with holes.

We also show that the discrete Voronoi game is PSPACE-complete in the general case. This can be seen as a positive answer to the conjecture by Fekete and Meijer [56].

4.3 Problem definitions – *Voronoi Game on Graphs*

In this section, we formulate the discrete Voronoi game on a graph. Let us denote a Voronoi game by $VG(G, n)$, where G is the game arena, and the players play n rounds. Hereafter, the game arena is an undirected and unweighted simple graph $G = (V, E)$ with $N = |V|$ vertices.

For each round, the two players, $\mathcal{W}$ (white) and $\mathcal{B}$ (black), alternately occupy an unoccupied vertex on the graph G ($\mathcal{W}$ always starts the game, as in Chess). This implies that $\mathcal{W}$ and $\mathcal{B}$ cannot occupy a common vertex at any time. Hence it is implicitly assumed that the game arena G contains at least $2n$ vertices.

Let W_i (resp. B_i) be the set of vertices occupied by player $\mathcal{W}$ (resp. $\mathcal{B}$) at the end of the i-th round. We define the distance $d(v, w)$ between two vertices v and w as the number of edges along the shortest path between them, if such path exists; otherwise $d(v, w) = \infty$. Each vertex of G can be assigned to the nearest vertices occupied by $\mathcal{W}$ and $\mathcal{B}$, according to the *nearest neighbor rule*. So, we define a *dominance set* $\mathcal{V}(A, B)$ (or *Voronoi regions*) of a subset $A \subset V$ against a subset $B \subset V$, where $A \cap B = \emptyset$, as

$$\mathcal{V}(A, B) = \{u \in V \mid \min_{v \in A} d(u, v) < \min_{w \in B} d(u, w)\}.$$

The dominance sets $\mathcal{V}(W_i, B_i)$ and $\mathcal{V}(B_i, W_i)$ represent the sets of vertices dominated at the end of the i-th round by $\mathcal{W}$ and $\mathcal{B}$, respectively. Let $\mathcal{V}_\mathcal{W}$ and $\mathcal{V}_\mathcal{B}$ denote $\mathcal{V}(W_n, B_n)$ and $\mathcal{V}(B_n, W_n)$, respectively. Since some vertex can be a "tie" when it has the same distance from a vertex occupied by $\mathcal{W}$ and another vertex occupied by $\mathcal{B}$, there may exist a set N_i of *neutral* vertices, $N_i := \{u \in V \mid \min_{v \in W_i} d(u, v) = \min_{w \in B_i} d(u, w)\}$, disjoint from both $\mathcal{V}(W_i, B_i)$ and $\mathcal{V}(B_i, W_i)$.

Finally, the player who dominates a larger number of vertices wins the discrete Voronoi game. More precisely, $\mathcal{W}$ *wins* if $|\mathcal{V}_\mathcal{W}| > |\mathcal{V}_\mathcal{B}|$; $\mathcal{B}$ wins (or $\mathcal{W}$ *loses*) if $|\mathcal{V}_\mathcal{W}| < |\mathcal{V}_\mathcal{B}|$; and the players *tie* otherwise. The *outcome* for each player, $\mathcal{W}$ or $\mathcal{B}$, is the size of the dominance set $|\mathcal{V}_\mathcal{W}|$ or $|\mathcal{V}_\mathcal{B}|$. In our model, note that any vertices in N_n do not contribute to the outcomes $\mathcal{V}_\mathcal{W}$ and $\mathcal{V}_\mathcal{B}$ of the players (see Fig. 4.1).

4.4 Discrete Voronoi Game on a Complete k-ary Tree

In this section, we consider the case that the game arena G is a complete k-ary tree T, which is a rooted tree whose inner vertices have exactly k children, and all leaves are at the same level (the highest level).

Firstly, we show a simple observation for Voronoi games $VG(T, n)$ that satisfy $2n \leq k$. In this game of a few rounds, $\mathcal{W}$ occupies the root of T with her first move, and then $\mathcal{W}$

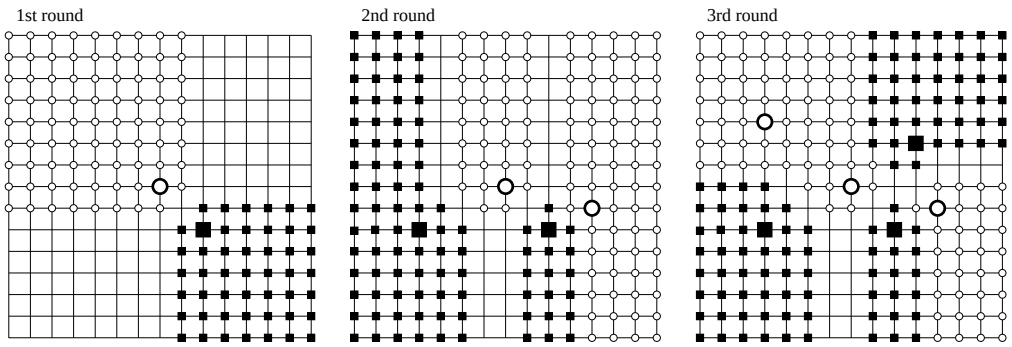


Figure 4.1: Example of a discrete Voronoi game $VG(G, 3)$, where G is the 15×15 grid graph; each bigger circle is a vertex occupied by $\mathcal{W}$, each smaller circle is an unoccupied vertex dominated by $\mathcal{W}$, each bigger black square is a vertex occupied by $\mathcal{B}$, each smaller black square is an unoccupied vertex dominated by $\mathcal{B}$, and the others are neutral vertices. In this example, the 2nd player $\mathcal{B}$ won by 108–96.

can dominates at least $\frac{N-1}{k}n + 1$ vertices. Since $\mathcal{B}$ dominate at most $\frac{N-1}{k}n$ vertices, $\mathcal{W}$ wins. More precisely, we show the following algorithm as $\mathcal{W}$'s winning strategy.

Algorithm 9: Simple strategy

Stage I: ($\mathcal{W}$'s first move) $\mathcal{W}$ occupies the root of T;

Stage II: $\mathcal{W}$ occupies the unoccupied child of the root for her remaining rounds;

In the strategy of Algorithm 9, $\mathcal{W}$ alternately pretends to occupy the unoccupied children of root, though $\mathcal{W}$ may occupy any vertex. This strategy is obviously well-defined and a winning strategy for $\mathcal{W}$, whenever the game arena T satisfies $2n \leq k$.

Proposition 4.4.1 *Let $VG(G, n)$ be the discrete Voronoi game such that G is a complete k-ary tree with $2n \leq k$. Then the first player $\mathcal{W}$ always wins.*

We next turn to a more general case. We call a k-ary tree odd (resp. even) if k is odd (resp. even). Let T be a complete k-ary tree as a game arena, N be the number of vertices of T, and H be the height of T. Note that $N = \frac{k^{H+1}-1}{k-1}$ and $H \sim \log_k N$. [1] For this game, we show the following theorem.

Theorem 4.4.2 *In the discrete Voronoi game $VG(G, n)$ where G is a complete k-ary tree such that $N \geq 4n^2$, the first player $\mathcal{W}$ always wins if G is an odd k-ary tree; otherwise the game ends in tie when the players do their best.*

[1] In this paper, we write $f(x) \sim g(x)$ when $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$.

In section 4.4.1, we first show winning strategy for the first player $\mathcal{W}$ when k is odd and the complete k-ary tree contains at least $4n^2$ vertices. Since our game arena is discrete, it is necessary to deliberate the relation between the number of children k and the game round n. Indeed, $\mathcal{W}$ chooses one of two strategies according to the relation between k and n. We next consider the even k-ary tree in section 4.4.2, which completes the proof of Theorem 4.4.2.

4.4.1 Discrete Voronoi game on a large complete odd k-ary tree

We generalize the simple strategy to Voronoi games $VG(T, n)$ on a large complete k-ary tree, where $2n > k$ and k is odd ($k \geq 3$). We define that a level h is the *keylevel* if the number k^h of vertices satisfies $n \leq k^h < 2n$, and a vertex v is a *key-vertex* if v is in the keylevel. Let T_i denote the number of vertices in the subtree rooted at a vertex in level i (i.e., $T_0 = N$, $T_i = kT_{i+1} + 1$). Let $\{V_1^h, V_2^h, \ldots, V_{k^{h-1}}^h\}$ be a family of vertices in the keylevel h such that the set V_i^h consists of k vertices which have the same parent for each i.

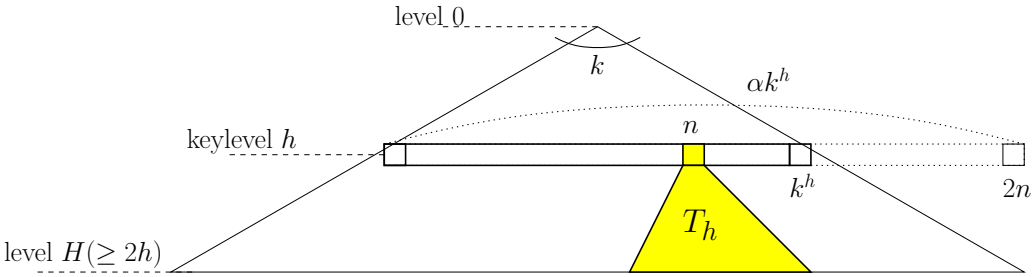


Figure 4.2: The notations on the game arena T.

As mentioned above, a winning strategy is sensitive to the relation between k, h, and n. So, we firstly introduce a magic number $\alpha = \frac{2n}{k^h}$, $1 < \alpha < k$ (see Fig. 4.2). We note that since k is odd, we have neither $\alpha = 1$ nor $\alpha = k$. By assumption, we have that the game arena T is sufficiently large such that the subtrees rooted at level h contain sufficient vertices comparing to the number of vertices between level 0 and level h. More precisely, by the assumption $N \geq 4n^2$, we have $H \geq 2h$ and $N \geq \frac{4n^2}{\alpha^2}$. We define $\gamma := H - 2h$, and hence $\gamma \geq 0$.

The winning strategy for $\mathcal{W}$ chooses one of two strategies according to the condition whether the magic number α is greater than $1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$ or not. The strategy is shown in Algorithm 10.

Lemma 4.4.3 *The keylevel strategy is well-defined in a discrete Voronoi game $VG(T, n)$, where T is a sufficiently large complete k-ary tree so that $N \geq 4n^2$.*

Proof By assumption, there exists the keylevel h.

In Stage (a)-I, if $\mathcal{B}$ occupied a key-vertex in V_i^h and $\mathcal{W}$ has not occupied any vertex in V_i^h, $\mathcal{W}$ occupies an unoccupied key-vertex in V_i^h rather than occupying the other unoccupied key-vertices. This implies that $\mathcal{W}$ can occupy at least one key-vertex in each $V_i^h, i = 1, 2, \ldots, k^{h-1}$. Since the situation $\mathcal{W}$ follows Stage (a)-II may happen when $\mathcal{B}$ occupies at least one key-vertex, there exists such a child. If $\mathcal{W}$ follows the case (b), then this is obviously well-defined. So, the keylevel strategy is well-defined. □

Lemma 4.4.4 *The keylevel strategy is a winning strategy for $\mathcal{W}$ in a discrete Voronoi game $VG(T, n)$, where T is a sufficiently large complete odd k-ary tree so that $N \geq 4n^2$.*

Proof We first argue that $\mathcal{W}$ follows the case (a), or $\alpha > 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$. When the game ends in Stage (a)-I (i.e., $\mathcal{B}$ never occupies any key-vertices, or does not occupy so many key-vertices), the best strategy of $\mathcal{B}$ is as follows. Firstly, $\mathcal{B}$ occupies all vertices in level $h-1$ for the first k^{h-1} rounds, and then occupies a child of key-vertex dominated by $\mathcal{W}$ to dominate as many vertices as possible with her remaining moves. In fact, the winner dominates more leaves than that of the opponent. So, it is not so significant to occupy the vertices in a level strictly greater than $h+1$, and strictly less than $h-1$.

Now we estimate the players' outcomes $|\mathcal{V}_\mathcal{W}|$ and $|\mathcal{V}_\mathcal{B}|$. Firstly, $\mathcal{W}$ dominates nT_h vertices and $\mathcal{B}$ dominates $(k^h - n)T_h + \frac{k^h-1}{k-1}$ vertices. Since $\mathcal{B}$ dominates the subtrees of $\mathcal{W}$ with her remaining $n - k^{h-1}$ vertices,

$$\begin{aligned} |\mathcal{V}_\mathcal{W}| &= nT_h - (n - k^{h-1})\, T_{h+1}, \\ |\mathcal{V}_\mathcal{B}| &\leq (k^h - n)\, T_h + (n - k^{h-1})\, T_{h+1} + \frac{k^h - 1}{k-1}. \end{aligned}$$

Since $2n = \alpha k^h$ and $\alpha > 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$, we have

$$\begin{aligned} &|\mathcal{V}_\mathcal{W}| - |\mathcal{V}_\mathcal{B}| \\ &\quad \geq nT_h - 2(n - k^{h-1})\, T_{h+1} - (k^h - n)\, T_h - \frac{k^h - 1}{k-1} \\ &\quad > (k^{h+1}\alpha + 2k^{h-1} - k^h\alpha - k^{h+1})T_{h+1} - \frac{k^h - 1}{k-1} \\ &\quad \geq \frac{1}{k^\gamma}\, T_{h+1} - \frac{k^h - 1}{k-1}. \end{aligned}$$

By the definition of γ with $\gamma = H - 2h$, we have

$$\begin{aligned}
\frac{1}{k^\gamma}\, T_{h+1} - \frac{k^h - 1}{k-1} &= \frac{1}{k^\gamma}\,(kT_{h+2} + 1) - \frac{k^h - 1}{k-1} \\
&= \frac{1}{k^\gamma}\,\frac{k^{H-h} - 1}{k-1} - \frac{k^h - 1}{k-1} \\
&= \frac{1}{k^\gamma}\frac{k^{(2h+\gamma)-h} - 1}{k-1} - \frac{k^h - 1}{k-1} \\
&= \frac{1}{k-1}\left(1 - \frac{1}{k^\gamma}\right) \;>\; 0.
\end{aligned}$$

Next, we consider the case that $\mathcal{W}$ follows Stage (a)-II. At a level greater than h, there are three types of $\mathcal{B}$'s occupation (see Fig. 4.3). In cases (2) and (3) of Fig. 4.3, $\mathcal{B}$ has

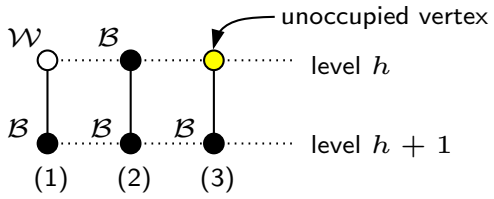


Figure 4.3: $\mathcal{B}$'s occupations at the level greater than h.

no profits. Therefore, when $\mathcal{B}$ uses his best strategy, we can assume that $\mathcal{B}$ only occupies vertices under $\mathcal{W}$'s vertices. This implies that $\mathcal{B}$ tries to perform a similar strategy to $\mathcal{W}$, that is, to occupy many key-vertices. More precisely, $\mathcal{B}$ chooses his move from the following options at every round:

- $\mathcal{B}$ occupies an unoccupied key-vertex; or
- $\mathcal{B}$ occupies a vertex v in level $h+1$, where the parent of v is a key-vertex of $\mathcal{W}$; or
- $\mathcal{B}$ occupies a vertex w in level $h+1$, where the parent of w is a key-vertex of $\mathcal{B}$.

This implies that almost all key-vertices are occupied by either $\mathcal{W}$ or $\mathcal{B}$, and then the subtree of T consisting of the vertices in level 0 through $h-1$ is negligibly small so that these vertices cannot have much effect on outcomes of $\mathcal{W}$ and $\mathcal{B}$. It is not significant to the occupation of these vertices for both players.

Let x_i (resp. y_i) be the number of vertices occupied by $\mathcal{W}$ (resp. $\mathcal{B}$) in level i. Let y_i^+ (resp. y_i^-) be the number of vertices occupied by $\mathcal{B}$ in higher (resp. lower) than or equal to level i.

When Stage (a)-I ends, $\mathcal{W}$ has x_h key-vertices and $\mathcal{B}$ has y_h key-vertices. Note that $x_h + y_h \le k^h$ and $y_h < \lceil \frac{k^h}{2} \rceil \le x_h < n$. x_{h+1} is the number of vertices occupied in Stage (a)-II. Let y'_{h+1} be the number of occupations used to dominate vertices of $\mathcal{W}$'s

dominance set by $\mathcal{B}$ in level $h+1$, and y''_{h+1} be $y_{h+1} - y'_{h+1}$. (see Fig. 4.4). Note that $x_h - y_h \geq y'_{h+1} - x_{h+1}$ (with equality if $y''_{h+1} + y^-_{h-1} + y^+_{h+2} = 0$). Now, we estimate their

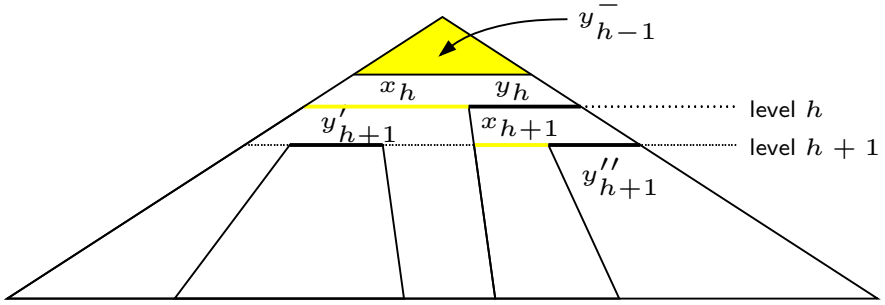


Figure 4.4: The notations in the case (a) of keylevel strategy.

outcomes. Since $\mathcal{W}$ can dominate at least $x_h T_h + (x_{h+1} - y'_{h+1})T_{h+1}$ vertices, and $\mathcal{W}$ dominates $y_h T_h + (y'_{h+1} - x_{h+1})T_{h+1}$ vertices, the difference between the outcomes of $\mathcal{W}$ and $\mathcal{B}$ is

$$\begin{aligned}
&|\mathcal{V}_\mathcal{W}| - |\mathcal{V}_\mathcal{B}| \\
&\quad = x_h T_h + (x_{h+1} - y'_{h+1})T_{h+1} - y_h T_h - (y'_{h+1} - x_{h+1})T_{h+1} \\
&\quad \geq \big(k(x_h - y_h) - 2(y'_{h+1} - x_{h+1})\big)\, T_{h+1} > T_{h+1} > 0.
\end{aligned}$$

$\mathcal{W}$ can dominate at least T_{h+1} vertices more than that of $\mathcal{B}$, which is more vertices dominated by $\mathcal{B}$ using y_0 vertices between level 0 and h. So, $\mathcal{W}$ wins when $\alpha > 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$.

We next argue that $\mathcal{W}$ follows the case (b), or $\alpha \leq 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$. When $x_{h-1} = k^{h-1}$, the best strategy for $\mathcal{B}$ is to occupy as many key-vertices as possible. So, the differences of outcomes are estimated as follows:

$$\begin{aligned}
&|\mathcal{V}_\mathcal{W}| - |\mathcal{V}_\mathcal{B}| \\
&\quad = (k^h - 2n)\; T_h + 2(n - k^{h-1})\; T_{h+1} + \frac{k^h - 1}{k-1} \\
&\quad \geq (k^{h+1} - 2k^{h-1} - k^h(k-1)\alpha)T_{h+1} + 2 \cdot \frac{k^h - 1}{k-1} \\
&\quad \geq 2 \cdot \frac{k^h - 1}{k-1} - \frac{1}{k^\gamma}\; T_{h+1} \\
&\quad = 2 \cdot \frac{k^h - 1}{k-1} - \frac{1}{k^\gamma}\frac{k^{h+\gamma} - 1}{k-1} = \frac{1}{k-1}\left(k^h - 2 + \frac{1}{k^\gamma}\right) \\
&\quad > 0.
\end{aligned}$$

Finally, we consider the case of $\alpha < 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$ and $x_{h-1} < k^{h-1}$ (or $x_{h-1} + y_{h-1} = k^{h-1}$). In this case, the similar arguments in which $\mathcal{W}$ follows Stage (a)-

II can be applied. Each x_{h-1}, x_h, and x_{h+1} is the number of vertices occupied in Stage (b)-I, (b)-II, and (b)-III, respectively. As mentioned above, y^-_{h-2} and y^+_{h+2} should be 0 to maximize $\mathcal{B}$'s outcome $|\mathcal{V}_\mathcal{B}|$. Let y'_h be the number of key-vertices occupied by $\mathcal{B}$ whose parent is occupied by $\mathcal{W}$, and $y''_h = y_h - y'_h$. Fig. 4.5 shows these notations. If $\mathcal{W}$ does not

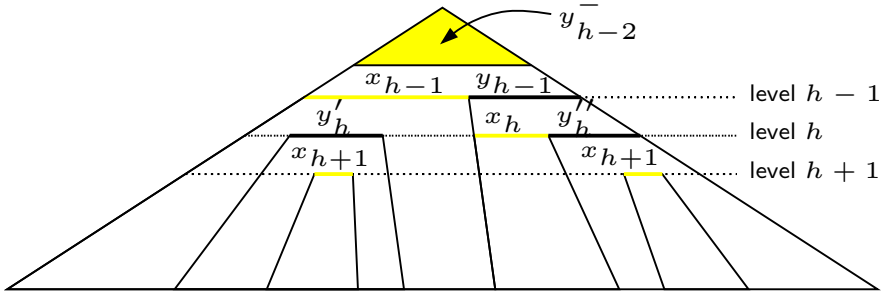


Figure 4.5: The notations in the case (b) of keylevel strategy.

follow Stage (b)-III, then $\mathcal{W}$ wins since $x_{h-1} - y_{h-1} \geq y'_h - x_h$ and $k(x_{h-1} - y_{h-1}) - 2(x_h - y'_h) > 0$. If $\mathcal{W}$ follows Stage (b)-III, then we have $y_{h-1} + y'_h + y''_h \leq n$, $x_h + y''_h = y_{h-1}$, and $x_{h-1} > \frac{1}{2}k^{h-1} > y_{h-1}$ by the keylevel strategy. We can estimate the outcome of $\mathcal{W}$ as follows;

$$\begin{aligned}
|\mathcal{V}_\mathcal{W}| - |\mathcal{V}_\mathcal{B}| &= x_{h-1}T_{h-1} + (x_h - 2y'_h - y''_h)T_h + 2x_{h+1}T_{h+1} \\
&> kx_{h-1} + x_h - 2y'_h - y''_h \\
&\geq k^h + 2(k^{h-1} - x_{h-1}) - \alpha k^h \\
&\geq \frac{k^{h-1}}{k-1} - \frac{1}{k^\gamma(k-1)} \\
&> 0.
\end{aligned}$$

Therefore, the first player $\mathcal{W}$ wins when she follows case (b) in the keylevel strategy. This completes the proof of Lemma 4.4.4. □

4.4.2 Discrete Voronoi game on a large complete even k-ary tree

We consider the case that the game arena T is a large complete even k-ary tree. We assume that the game $VG(T, n)$ is sufficed $k > 2n$, since $\mathcal{W}$ always wins if $k \leq 2n$ as mentioned above. Moreover, we assume that the game arena T contains at least $4n^2$ vertices. Hence the first player $\mathcal{W}$ always loses if she occupies the root of T, since the second player $\mathcal{B}$ can use the keylevel strategy of $\mathcal{W}$ and $\mathcal{W}$ cannot drive $\mathcal{B}$ in disadvantage.

In fact, since T is an even k-ary tree, $\mathcal{B}$ can take the symmetric moves of $\mathcal{W}$ if $\mathcal{W}$ does not occupy the root. Therefore, $\mathcal{B}$ never loses. However, we can show that $\mathcal{W}$ also never loses if she follows the keylevel strategy.

If $\mathcal{B}$ has a winning strategy, then the strategy must not be the symmetric strategy of $\mathcal{W}$. However, such a strategy does not exist, since $\mathcal{W}$ can occupy at least half of the vertices on the important level, although the important level is varied by the condition $\alpha > 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma(k-1)}}$. This implies that $\mathcal{W}$ can dominate at least half the vertices of T if she follows the keylevel strategy. Therefore, if both players do their best, then the game always ends in a tie.

4.5 NP-Hardness for General Graphs

In this section, we show that the discrete Voronoi game is intractable on general graphs even if we restrict ourselves to the one-round case. To show this, we consider the following special case:

Problem 1:
Input: A graph $G = (V, E)$, a vertex $u \in V$, and n.
Output: Determine whether $\mathcal{B}$ has a winning strategy on G by n occupations after just one occupation of u by $\mathcal{W}$.

That is, $\mathcal{W}$ first occupies u, and never occupies any more, and $\mathcal{B}$ can occupy n vertices in any way. Then we have the following theorem:

Theorem 4.5.1 *Problem 1 is* NP*-complete.*

Proof It is clear Problem 1 is in NP. Hence we prove the completeness by showing a polynomial time reduction from a restricted 3SAT such that each variable appears at most three times in a given formula [104, Proposition 9.3]. Let F be a given formula with the set W of variables $\{x_1, x_2, \ldots, x_n\}$ and the set C of clauses $\{c_1, c_2, \ldots, c_m\}$, where $n = |W|$ and $m = |C|$. Each clause contains at most 3 literals, and each variable appears at most 3 times. Hence we have $3n \geq m$.

Now we show a construction of G. Let

$W^+ := \{x_i^+ \mid x_i \in W\}$,
$W^- := \{x_i^- \mid x_i \in W\}$,
$Y := \{y_i^j \mid i \in \{1, 2, \ldots, n\}, j \in 1, 2, 3\}$,
$Z := \{z_i^j \mid i \in \{1, 2, \ldots, n\}, j \in 1, 2, 3\}$,
$C' := \{c'_1, c'_2, \ldots, c'_m\}$,
$D := \{d_1, d_2, \ldots, d_{2n-2}\}$.

Then the set of vertices of G is defined by $V := \{u\} \cup W^+ \cup W^- \cup Y \cup Z \cup C \cup C' \cup D$.

The set of edges E is defined by the union of the following edges:

$\{\{u, z\} \mid z \in Z\}$,
$\{\{y_i^j, z_i^j\} \mid y_i^j \in Y, z_i^j \in Z \text{ with } 1 \leq i \leq n, 1 \leq j \leq 3\}$,
$\{\{x_i^+, y_i^j\} \mid x_i^+ \in W^+, y_i^j \in Y \text{ with } 1 \leq i \leq n, 1 \leq j \leq 3\}$,
$\{\{x_i^-, y_i^j\} \mid x_i^- \in W^+, y_i^j \in Y \text{ with } 1 \leq i \leq n, 1 \leq j \leq 3\}$,
$\{\{x_i^+, c_j\} \mid x_i^+ \in W^+, c_j \in C \text{ if } c_j \text{ contains literal } x_i\}$,
$\{\{x_i^-, c_j\} \mid x_i^- \in W^-, c_j \in C \text{ if } c_j \text{ contains literal } \bar{x}_i\}$,
$\{\{c_j, c'_j\} \mid c_j \in C, c'_j \in C' \text{ with } 1 \leq j \leq m\}$,
$\{\{c'_j, u\} \mid c'_j \in C' \text{ with } 1 \leq j \leq m\}$, and
$\{\{u, d_i\} \mid d_i \in D \text{ with } 1 \leq i \leq 2n-2\}$.

An example of the reduction for the formula $F = (\bar{x_1} \vee x_2 \vee x_3) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$ is depicted in Fig. 4.6. Small white and black circles are the vertices in Z and Y, respectively; large black circles are the vertices in $W^+ \cup W^-$; black and white rectangles are the vertices in C and C', respectively; two white large diamonds are the same vertex u; and small diamonds are the vertices in D. It is easy to see that G contains $10n + 2m - 1$ vertices, and hence the reduction can be done in polynomial time.

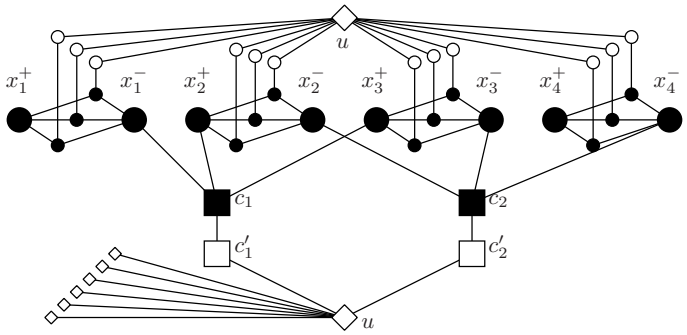


Figure 4.6: Reduction from $F = (\bar{x_1} \vee x_2 \vee x_3) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$

Now we show that F is satisfiable if and only if $\mathcal{B}$ has a winning strategy. We first observe that for $\mathcal{B}$, occupying the vertices in $W^+ \cup W^-$ gives more outcome than occupying the vertices in $Y \cup Z \cup C \cup C'$. More precisely, occupying either x_i^+ or x_i^- for each i with $1 \leq i \leq n$, $\mathcal{B}$ dominates all vertices in $W^+ \cup W^- \cup Y$, and it is easy to see that any other way achieves less outcome. Therefore, we can assume that $\mathcal{B}$ occupies one of x_i^+ and x_i^- for each i with $1 \leq i \leq n$.

When there is an assignment $(a_1, a_2, \ldots, a_n)$ that satisfies F, $\mathcal{B}$ can also dominates all vertices in C by occupying x_i^+ if $a_i = 1$, and occupying x_i^- if $a_i = 0$. Hence, $\mathcal{B}$ dominates $5n + m$ vertices in this case, and then $\mathcal{W}$ dominates all vertices in Z, C' and D, that is, $\mathcal{W}$ dominates $1 + 3n + m + 2n - 2 = 5n + m - 1$ vertices. Therefore, $\mathcal{B}$ wins if F is

satisfiable.

On the other hand, if F is unsatisfiable, $\mathcal{B}$ can dominate at most $5n+m-1$ vertices. In this case, the vertex in C corresponding to the unsatisfied clause is dominated by u. Thus $\mathcal{W}$ dominates at least $5n+m$ vertices, and hence $\mathcal{W}$ wins if F is unsatisfiable.

Therefore, Problem 1 is NP-complete. □

Next we show that the discrete Voronoi game is NP-hard even in the one-round case. More precisely, we show the NP-completeness of the following problem:

Problem 2:
Input: A graph $G=(V,E)$, a vertex set $S \subseteq V$ with $n := \|S\|$.
Output: Determine whether $\mathcal{B}$ has a winning strategy on G by n occupations, after n occupations of the vertices in S by $\mathcal{W}$.

Corollary 4.5.2 *Problem 2 is* NP*-complete.*

Proof We use the same reduction in the proof of theorem 4.5.1. Let S be the set that contains u and $n-1$ vertices in D. Then we immediately have NP-completeness of Problem 2. □

Corollary 4.5.3 *The (n-round) discrete Voronoi game on a general graph is* NP*-hard.*

4.6 PSPACE-Completeness for General Graphs

In this section, we show that the discrete Voronoi game is intractable on general graphs. More precisely, we consider the following general case:

Problem 3:
Input: A graph $G=(V,E)$ and n.
Output: Determine whether $\mathcal{W}$ has a winning strategy on G by n occupations.

Then we have the following theorem:

Theorem 4.6.1 *The discrete Voronoi game is* PSPACE*-complete in general.*

Proof We show that Problem 3 is PSPACE-complete. It is clear Problem 3 is in PSPACE. Hence we prove the completeness by showing a polynomial time reduction from the following two-person game:

G_{pos}(Pos Dnf):

Input: A positive DNF formula A (that is, a DNF formula containing no negative literal).

Rule: Two players alternately choose some variable of A which has not been chosen. The game ends after all variables of A has been chosen. The first player wins if and only if A is true when all variables chosen by the first player are set to 1 and all variables chosen by the second player are set to 0. (In other words, the first player wins if and only if he takes every variable of some disjunct.)

Output: Determine whether the first player has a winning strategy for A.

The game G_{pos}(Pos Dnf) is PSPACE-complete even with inputs restricted to DNF formulas having at most 11 variables in each disjunct (see [113, Game 5(b)]).

Let A be a positive DNF formula with n variables $\{x_1, \ldots, x_n\}$ and m disjuncts $\{d_1, \ldots, d_m\}$. Without loss of generality, we assume that n is even. Now we show a construction of $G = (V, E)$. Let $X = \{x_1, \ldots, x_n\}$, $D = \{d_1, \ldots, d_m\}$, $U = \{u_1, u_2\}$, and $P = \{p_1, \ldots, p_{2n^2+6n}\}$. Then the set of vertices of G is defined by $V := X \cup D \cup U \cup P$.

In this reduction, each pendant in P is attached to some vertex in $X \cup U$ to make it "heavy."

The set of edges E consists of the following edges: (1) make X a clique with edges $\{x_i, x_j\}$ for each $1 \leq i < j \leq n$, (2) join a vertex x_i in X with a vertex d_j in D if A has a disjunct d_j that contains x_i, (3) join each d_j with u_2 by $\{d_j, u_2\}$ for each $1 \leq j \leq m$, (4) join u_1 and u_2 by $\{u_1, u_2\}$, (5) attach $2n$ pendants to each x_i with $1 \leq i \leq n$, and (6) attach $3n$ pendants to each u_i with $i = 1, 2$.

An example of the reduction for the formula $A = (x_1 \wedge x_2 \wedge x_4 \wedge x_5) \vee (x_3 \wedge x_5 \wedge x_7 \wedge x_8) \vee (x_6 \wedge x_8)$ is depicted in Fig. 4.7. Black diamond and white diamond are u_1 and u_2, respectively; white squares are the vertices in D; and small circles are vertices in X. Large white numbered circles are pendants, and the number indicates the number of pendants attached to the vertex.

Each player will occupy $(n/2) + 1$ vertices in G. It is easy to see that G contains $n + m + 2 + 6n + 2n^2 = 2n^2 + 7n + m + 2$ vertices, and hence the reduction can be done in polynomial time.

Now we show that the first player of G_{pos}(Pos Dnf) for A wins if and only if $\mathcal{W}$ of the discrete Voronoi game for G wins.

Since the vertices in X and U are heavy enough, $\mathcal{W}$ and $\mathcal{B}$ always occupy the vertices in X and U. In fact, occupying a vertex d_j in D does not bring any advantage; since X induces a clique, the pendants attached to some x_i in $N(d_j)$ will be canceled by occupying any $x_{i'}$ by the other player.

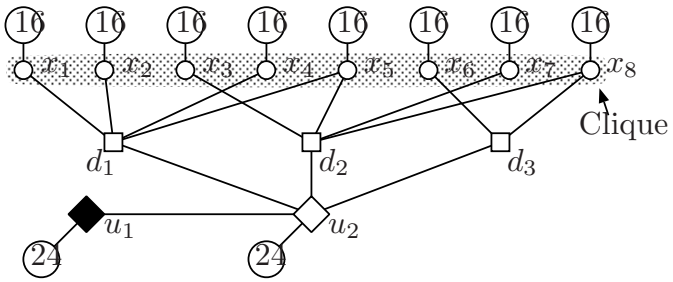


Figure 4.7: Reduction from $A = (x_1 \wedge x_2 \wedge x_4 \wedge x_5) \vee (x_3 \wedge x_5 \wedge x_7 \wedge x_8) \vee (x_6 \wedge x_8)$

Since the vertices in U are heavier than the vertices in X, $\mathcal{W}$ and $\mathcal{B}$ first occupy one of u_1 and u_2, and occupy the vertices in X, and the game will end when all vertices in X are occupied.

The player $\mathcal{W}$ has two choices.

We first consider the case in which $\mathcal{W}$ occupies u_2. Then $\mathcal{B}$ has to occupy u_1, and $\mathcal{W}$ and $\mathcal{B}$ occupy $n/2$ vertices in X. It is easy to see that, in this case, they tie on the graph induced by $U \cup X \cup P$. Hence the game depends on the occupation of D. In G_{pos}(Pos Dnf), if the first player has the winning strategy for A, the first player can take every variable of a disjunct d_j. Hence, following the strategy, $\mathcal{W}$ can occupy every variable in $N(d_j)$ on G. Then, since $\mathcal{W}$ also occupies u_2, d_j is dominated by $\mathcal{W}$. On the other hand, $\mathcal{B}$ cannot dominate any vertex in D since $\mathcal{W}$ occupies u_2. Hence, if the first player of G_{pos}(Pos Dnf) has a winning strategy, so does $\mathcal{W}$. (Otherwise, the game ends in a tie.)

Next, we consider the case in which $\mathcal{W}$ occupies u_1. Then $\mathcal{B}$ can occupy u_2. The game again depends on the occupation of D. However, in this case, $\mathcal{W}$ cannot dominate any vertex in D since $\mathcal{B}$ has already occupied u_2. Hence $\mathcal{W}$ will lose or they will tie at best.

Thus $\mathcal{W}$ has to occupy u_2 at first, and then $\mathcal{W}$ has a winning strategy if the first player of G_{pos}(Pos Dnf) has it.

Therefore, Problem 3 is PSPACE-complete. □

4.7 Concluding Remarks and Further Research

We gave winning strategies for the first player $\mathcal{W}$ on the discrete Voronoi game $VG(T, n)$, where T is a large complete k-ary tree with odd k. It seems that $\mathcal{W}$ has an advantage even if the complete k-ary tree is not large, which is future work.

In our strategy, it is essential that each subtree of the same depth has the same size.

Therefore, considering general trees is the next problem. The basic case is easy: When $n = 1$, the discrete Voronoi game on a tree is essentially equivalent to finding a *median* vertex of a tree. The deletion of a median vertex partitions the tree so that no component contains more than $n/2$ of the original n vertices. It is well known that a tree has either one or two median vertices, which can be found in linear time (see, e.g., [71]). In the former case, $\mathcal{W}$ wins by occupying the median vertex. In the later case, two players tie. This algorithm corresponds to our Algorithm 9.

Algorithm 10: Keylevel strategy for $\mathcal{W}$

if $\alpha > 1 + \frac{2}{k} - \frac{1}{k-1} + \frac{1}{k^{h+\gamma}(k-1)}$ **then**

Stage (a)-I:

$\mathcal{W}$ occupies an unoccupied key-vertex so that at least one vertex is occupied in each V_i^h;

(Stage (a)-I ends after the last key-vertex is occupied by either $\mathcal{W}$ or $\mathcal{B}$. Note that the game may finish in Stage (a)-I.)

end

Stage (a)-II:

$\mathcal{W}$ occupies an unoccupied vertex which is a child of the vertex v, such that v is occupied by $\mathcal{B}$, and v has the minimum level greater than or equal to h;

($\mathcal{W}$ dominates as many vertices as possible from $\mathcal{B}$.)

end

else

Stage (b)-I:

$\mathcal{W}$ occupies an unoccupied vertex in level $h-1$;

(Stage (b)-I ends when such unoccupied vertices are not exists.)

end

Stage (b)-II:

$\mathcal{W}$ occupies an unoccupied key-vertex whose parent is not occupied by $\mathcal{W}$;

(Stage (b)-II ends when such unoccupied key-vertices are not exist.)

end

Stage (b)-III:

if *there exists an unoccupied vertex v in level $h+1$ such that the parent of v is occupied by $\mathcal{B}$* **then** $\mathcal{W}$ occupies v;

else $\mathcal{W}$ occupies an unoccupied key-vertex in level $h+1$ whose parent is occupied by $\mathcal{W}$;

end

end

Bibliography

[1] H. Adachi, H. Kamekawa, and S. Iwata, "Shogi $n \times n$ board is complete in exponential time," *Trans. IEICE* J70-D:1843–1852 (in Japanese), 1987.

[2] H. -K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum, "Competitive facility location: the Voronoi game," *Theoretical Computer Science,* 310:457–467, 2004.

[3] O. Aichholzer, "The path of a triangulation," *Proc. 15th Ann. ACM Symp. Computational Geometry,* pp. 14–23, 1999.

[4] O. Aichholzer, "Counting Triangulations – Olympics," `http://www.igi.tugraz.at/oaich/triangulations/counting/counting.html`.

[5] O. Aichholzer, and H. Krasser, "The point set order type data base: a collection of applications and results," *Proc. 13th Canadian Conference on Computational Geometry,* pp. 17–20, 2001.

[6] M. Ajtai, V. Chvátal, M. M. Newborn, and E. Szemerédi, "Crossing-free subgraphs," *Annals Discrete Math.,* 12:9–12, 1982.

[7] H. Alt, U. Fuchs, and K. Kriegel, "On the number of simple cycles in planar graphs," *Combinatorics Probab. Comput.,* 8:397–405, 1999.

[8] S. P. Anderson, "Equilibrium existence in a linear model of spatial competition," *Economica,* 55:479–491, 1988.

[9] B. Aronov, M. van Kreveld, R. van Oostrum, and K. Varadarajan, "Facility location on terrains," *9th Internat. Symp. of Algorithms and Computation,* vol. 1533, Lecture Notes in Computer Science, Springer, Berlin, pp. 19–28, 1998.

[10] B. Aronov, T. Asano, Y. Kikuchi, S. C. Nandy, S. Sasahara, and T. Uno, "A Generalization of Magic Squares with Applications to Digital Halftoning," to appear in *Theory of Computing System.*

[11] S. Arora and B. Barak, "Computational Complexity: A Modern Approach," to appear: `http://www.cs.princeton.edu/theory/complexity/`.

[12] T. Asano, "Computational Geometric and Combinatorial Approaches to Digital Halftoning," *Proc. of Conferences in Research and Practice in Information Technology,* vol. 51, p. 3, 2006.

[13] T. Asano, P. Brass, and S. Sasahara, "Disc Covering Problem with Application to Digital Halftoning," *Proc. Int. Conf. on Computer Science and Applications,* vol. 3, pp. 11–21, 2004.

[14] T. Asano, N. Katoh, K. Obokata, and T. Tokuyama, "Matrix rounding under the L_p-discrepancy measure and its application to digital halftoning," *SIAM Journal on Computing,* 32(6):1423–1435, 2003.

[15] T. Auer and M. Held, "Heuristics for the Generation of Random Polygons," *Proc. 8th Canadian Conference on Computational Geometry,* pp. 38–44, 1996.

[16] T. Auer. "Heuristics for the Generation of Random Polygons," Master's thesis, Computerwissenschaften, U. Salzburg, A-5020 Salzburg, Austria, June 1996.

[17] F. Aurenhammer, and R. Klein, "Voronoi Diagrams," Ch. 5 in *Handbook of Computational Geometry* (Ed. J.-R. Sack and J. Urrutia), Amsterdam, Netherlands: North-Holland, pp. 201-290, 2000.

[18] D. Avis and K. Fukuda, "Reverse Search for Enumeration," *Discrete Applied Mathematics,* 65:21–46, 1996.

[19] E. R. Berlekamp, J. H. Conway, and R. K. Guy, "Winning Ways," *Academic Press,* London, 1982.

[20] S. Bespamyatnikh, "An efficient algorithm for enumeration of triangulations," *Computational Geometry Theory and Application,* 23(3):271–279, 2002.

[21] M. de Berg, O. Schwarzkopf, M. van Kreveld and M. Overmars, "Computational Geometry: Algorithms and Applications 2nd edition," *Springer-Verlag,* 2000.

[22] B. K. Bhattacharya, S. K. Ghosh, and T. C. Shermer, "A linear time algorithm to remove winding of a simple polygon," *Computational Geometry Theory and Applications,* 33:165–173, 2006.

[23] P. Brass, W. O. J. Moser, and J. Pach, "Research Problems in Discrete Geometry," *Springer-Verlag,* 2005.

[24] R. Breukelaar, E. D. Demaine, S. Hohenberger, H. J. Hoogeboom, W. A. Kosters, and D. Liben-Nowell, "Tetris is Hard, Even to Approximate," *International Journal of Computational Geometry and Applications,* 14:41–68, 2004.

[25] CGAL: Computational Geometry Algorithms Library. `http://www.cgal.org/`.

[26] B. Chazelle, "The Discrepancy Method: Randomness and Complexity," *Cambridge University Press,* 2000.

[27] B. Chazelle, and J. Incerpi, "Triangulation and Shape-complexity," *ACM Trans. Graph.,* 3:135–152, 1984.

[28] Z. -Z. Chen, "Approximation Algorithms for Independent Sets in Map Graphs," *Journal of Algorithms,* 41:20–40, 2001.

[29] Z. -Z. Chen, M. Grigni and C. H. Papadimitriou, "Map Graphs," *Journal of ACM,* 49(2):127–138, 2002.

[30] S. -W. Cheng, "Planar Straight Line Graphs," Handbook of Data Structures and Applications, Edited by D.P. Mehta and S. Sahni, *Chapman & Hall*, 2005.

[31] O. Cheong, S. Har-Peled, N. Linial, and J. Matousek, "The one-round Voronoi game," *Discrete and Computational Geometry*, 31:125–138, 2004.

[32] W. G. Chinn, and N. E. Steenrod, "First Concepts of Topology: The Geometry of Mappings of Segments, Curves, Circles, and Disks," *Mathematical Association of America,* Washington, DC, 1966.

[33] C. R. Collins and K. Stephenson, "A Circle Packing Algorithm," *Computational Geometry Theory and Applications,* 25(3):233–256, 2003.

[34] J. H. Conway, "On Numbers and Games," *Academic Press,* London, 1976

[35] J. H. Conway, "All games bright and beautiful," *American Mathematical Monthly,* 84:417–434, 1977

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," *MIT Press,* 2001.

[37] I. K. Crain, "The Monte-Carlo generation of random polygons," *Computers and Geosciences,* 4:131–141, 1978.

[38] J. Culberson, "Sokoban is PSPACE-complete," *Technical Report TR97-02,* Department of Computer Science, The University of Alberta, `http://web.cs.ualberta.ca/~joe/Preprints/Sokoban/`, 1997.

[39] K. Dalal, "Counting the Onion," *Random Structures and Algorithms,* 24(2):155–165, 2004.

[40] E. D. Demaine. "Playing Games with Algorithms: Algorithmic Combinatorial Game Theory," *Mathematical Foundations of Computer Science,* 2136 of Lecture Notes in Computer Science, Springer, Berlin, pp. 18–32, 2001.

[41] E. D. Demaine, "Simple Polygonizations," `http://theory.lcs.mit.edu/~edemaine/polygonization/`.

[42] E. D. Demaine, J. S. B. Michell, and J. O'Rourke, "The Open Problems Project," `http://maven.smith.edu/~orourke/TOPP/`.

[43] S. de Vries and R. Vohra, "Combinatorial auction: A survey," *INFORMS Journal on Computing,* 15(3):284–309, 2003.

[44] B. Doerr, "Nonindependent Randomized Rounding and an Application to Digital Halftoning," *SIAM Journal on Computing,* 34(2):299-317, 2005.

[45] T. Drezner, "Locating a Single New Facility Among Existing Unequally Attractive Facilities," *Journal of Regional Science,* 34:237–252, 1994.

[46] T. Drezner, "Optimal Continuous Location of Retail Facility, Facility Attractiveness, and Market Share: an Interactive Model," *Journal of Retailing,* 70:49–64, 1994.

[47] T. Drezner and Z. Drezner, "Competitive Facilities: Market Share and Location with Random Utility," *Journal of Regional Science,* 36:1–15, 1996.

[48] H. A. Eiselt, G. Laporte, and J.-F. Thisse, "Competitive Location Models: a Framework and Bibliography," *Transportation Science* 27:44–54, 1993.

[49] H. A. Eiselt, and G. Laporte, "Competitive spatial models," *European J. Oper. Res.* 39:231–242, 1989.

[50] H. ElGindy, and D. Avis, "A linear algorithm for computing the visibility polygon from a point," *Journal of Algorithm,* 2:186–197, 1981.

[51] H. ElGindy, and G. T. Toussaint, "On triangulating palm polygons in linear time," N. M. -Thalmann and D. Thalmann, editors, *New Trends in Computer Graphics,* pp. 308–317. Springer-Verlag, 1988.

[52] D. Eppstein, "Computational complexity of games and puzzles." `http://www.ics.uci.edu/~eppstein/cgt/hard.html`.

[53] P. Epstein and J. Sack, "Generating triangulations at random," *Proc. 4th Canadian Conference on Computational Geometry,* pp. 305–310, 1992.

[54] S. Even, and R. E. Tarjan, "A combinatorial problem which is complete in polynomial space," *Journal of ACM* 23:710–719, 1976.

[55] S. P. Fekete, "On Simple Polygonalizations with Optimal Area," *Discrete Comput. Geom.,* 23:73–110, 2000.

[56] S. P. Fekete, and H. Meijer, "The one-round Voronoi game replayed," *Computational Geometry Theory and Applications,* 30:81–94, 2005.

[57] A. Flat, A. Goldberg, J. Harline, and A. Karlin, "Competitive generalized auctions," *Proc. 34th ACM Symposium on Theory of Computing,* pp. 72–81, 2002.

[58] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, "Computer Graphics: Principles and Practice," *Addison-Wesley,* 1990.

[59] S. Fortune, "Voronoi Diagrams and Delaunay Triangulations," Computing in Euclidean Geometry, Edited by Ding-Zhu Du and Frank Hwang, World Scientific, *Lecture Notes Series on Computing – Vol. 1,* pp. 193–233, 1992.

[60] A. Fournier and D. Y. Montuno, "Triangulating simple polygons and equivalent problems," *ACM Trans. Graph.,* 3:153–174, 1984.

[61] A. S. Fraenkel, and D. Lichtenstein, "Computing a Perfect Strategy for $n \times n$ Chess Requires Time Exponential in n," *J. Combin. Theory Ser. A.,* 31(2):199–214, 1981.

[62] A. S. Fraenkel, "Combinatorial Games: Selected Bibliography with a Succinct Gourmet Introduction," *Electron. Journal of Combinatorics,* 14 Dynamic Survey 2 (electronic), 2007. `http://www.combinatorics.org/Surveys/ds2.pdf`.

[63] A. S. Fraenkel, "Scenic trails ascending from sea-level Nim to alpine Chess," R. J. Nowakowski, editor, *Games of No Chance,* pp. 13–42, *Cambridge University Press,* 1996.

[64] A. García, M. Noy and J. Tejel, "Lower bounds on the number of crossing-free subgraphs of K_N," *Computational Geometry Theory and Applications,* 16:211–221, 2000.

[65] J.E. Goodman and J. O'Rourke, "Handbook of Discrete and Computational Geometry 2nd Edition," *Chapman & Hall,* 2004.

[66] S. Gooran, "Context Dependent Colour Halftoning in Digital Printing," *Proceedings of the Conference on Image Processing, Image Quality, Image Capture Systems,* pp.242–246, 2000.

[67] S. K. Ghosh, A. Maheshwari, S. P. pal, and C. E. Veni Madhavan, "An algorithm for recognizing palm polygons," *The Visual Computer,* 10(8):443–451, 2005.

[68] A. Ghosh, and S. McLafferty, "Location Strategies for Retail and Service Firms," *Lexington Books,* Lexington, Mass.

[69] S. L. Hakimi, "Location with spatial interactions: competitive location and games," *R. L. Francis, P. B. Mirchandani (Eds.), Discrete Location Theory, Wiley,* New York, pp. 439–478, 1990.

[70] H. W. Hamacher, and Z. Drezner, "Facility Location: Applications and Theory," *Springer,* 2004.

[71] F. Harary, "Graph Theory," *Addison-Wesley*, 1972.

[72] R. A. Hearn, and E. D. Demaine, "PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation," *Theoretical Computer Science,* 343:72–96, 2005.

[73] J. Hershberger and S. Suri, "Matrix Searching with the Shortest Path Metric," *SIAM Journal on Computing,* 26:1612–1634, 1997.

[74] S. Hertel, and K. Mehlhorn, "Fast triangulation of simple polygons," *Proc. 4th Internat. Conf. Found. Comput. Theory. Lecture Notes in Computer Science,* 158:207–218, *Springer-Verlag,* 1983.

[75] F. Hurtado, M. Noy and J. Urrutia, "Flipping Edges in Triangulations," *Discrete Comput Geom.,* 22:333–346, 1999.

[76] S. Iwata, and T. Kasai, "The Othello game on an $n \times n$ board is PSPACE-complete," *Theoretical Computer Science,* 123:329–340, 1994.

[77] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, "Random Generation of Combinatorial Structures from a Uniform Distribution," *Theoretical Computer Science,* 43:169–188, 1986.

[78] D. S. Johnson, M. Yannakakis and C. H. Papadimitriou, "On generating all maximal independent sets," *Information Processing Letters,* 27(3):119–123, 1988.

[79] K. Kanatani, "Statistical Optimization for Geometric Computation –Theory and Practice," *Dover Pubns,* 1996.

[80] R. Kaye, "Infinite versions of minesweeper are Turing-complete," *Math. Intelligencer* 22:9–15, 2000. `http://for.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf`.

[81] R. Kaye, "Minesweeper is NP-complete," *Mathematical Intelligencer* 22:9–15, 2000.

[82] M. van Kreveld and I. Reinbacher, "Good NEWS: Partitioning a Simple Polygon by Compass Directions," *International Journal of Computational Geometry & Applications,* 14:233–259, 2004.

[83] M. Labbé and S. L. Hakimi, "Market and locational equilibrium for two competitors," *Oper. Res.* 39:749–756, 1991.

[84] C. L. Lawson, "Transforming triangulations," *Discrete Math.,* 3:365–372, 1972.

[85] J. van Leeuwen and A. A. Schoone, "Untangling a traveling salesman tour in the plane," In J. R. Muhlbacher, editor, *Proc. 7th Conf. Graph-theoretic Concepts in Comput. Sci.,* pp. 87–98, 1981.

[86] J. Matoušek, "Geometric Discrepancy," *Springer,* 1999.

[87] L. McShine and P. Tetali, "On the mixing time of the triangulation walk and other Catalan structures," DIMACS-AMS volume on Randomization Methods in Algorithm Design (Eds. P.M. Pardalos, S. Rajasekaran, and J. Rolim) *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* 43:147–160, 1998.

[88] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," *SIAM Journal on Computing,* 12:759–776, 1983.

[89] G. H. Meisters, "Polygons have ears," *American Mathematical Monthyly,* 82:648–651, 1975.

[90] H. Meijer and D. Rappaport, "Upper and lower bounds for the number of monotone crossing free Hamiltonian cycles from a set of points," *ARS Combinatoria,* 30:203–208, 1990.

[91] J. S. B. Mitchell and J. O'Rourke, "Computational geometry column 42," *International Journal of Computational Geometry and Applications,* 11(5):573–582, 2001.

[92] M. Molloy, B. Reed and W. Steiger, "On the mixing rate of the triangulation walk," DIMACS-AMS volume on Randomization Methods in Algorithm Design (Eds. P.M. Pardalos, S. Rajasekaran, and J. Rolim) *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* 43:179–190, 1998.

[93] R. Motwani and P. Raghavan, "Randomized Algorithms," *Cambridge University Press,* 1995.

[94] K. Mehlhorn and S. Näher, "LEDA – A Platform of Combinatorial and Geometric Computing," *Cambridge University Press,* Cambridge, England, 1999.

[95] T. Mitsa and K. J. Parker, "Digital Halftoning using a Blue-Noise Mask," *Journal of the Optical Society of America* A, 9, 11, pp.1920-1929, 1992.

[96] K. J. Nurmela, P. R. J. Östergård and R. aus dem Spring, "Asymptotic Behavior of Optimal Circle Packings in a Square," *Canadian Mathematical Bulletin,* 42(3):380–385, 1999.

[97] K. J. Nurmela and P. R. J. Östergård, "Packing up to 50 Equal Circles in a Square," *Discrete and Computational Geometry,* 18(1):111–120, 1997.

[98] K. J. Nurmela, "More Optimal Packings of Equal Circles in a Square," *Discrete and Computational Geometry,* 22(3):439–457, 1999.

[99] A. Okabe, and M. Aoyagi, "Existence of Equilibrium Configurations of Competitive Firms on an Infinite Two-Dimensional Space," *J. Urban Econom.* 29:349–370, 1991.

[100] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu: "Spatial Tessellations: Concepts and Applications of Voronoi Diagrams," *John Wiley & Sons,* 2000.

[101] J. O'Rourke, "The art gallery theorems and algorithms," *Oxford University Press*, 1987.

[102] J. O'Rourke, "Computational Geometry in C," *Cambridge University Press,* 2001.

[103] J. O'Rourke and M. Virmani, "Generating Random Polygons," *Technical Report 011, CS Dept. Smith College,* Northhampton, MA 01063, July 1991.

[104] C. H. Papadimitriou, "Computational Complexity," *Addison-Wesley Publishing Company,* 1994.

[105] C. H. Papadimitriou, "Algorithms, games, and the Internet," *Proc. 33rd Annual ACM Symposium on Theory of Computing,* pp. 749–753, 2001.

[106] F. P. Preparata and M. I. Shamos, "Computational Geometry: An Introduction," *Springer-Verlag,* 1985.

[107] S. Reisch, "Gobang ist PSPACE-vollständig," *Acta Inform.* 13:59–66, 1981.

[108] S. Reisch, "Hex ist PSPACE-vollständig," *Acta Inform.* 15:167–191, 1981.

[109] J. M. Robson, "The complexity of GO," *Proc. IFIP* pp. 413–417, 1983.

[110] J. M. Robson, "N by N checkers is EXPTIME-complete," *SIAM Journal on Computing* 13:252–267, 1984.

[111] J. -R. Sack and J. Urrutia, "Handbook of Computational Geometry," *Elsevier Science Publishers,* 2000.

[112] K. Sadakane, N. Takki Chebihi, and T. Tokuyama, "Discrepancy-based digital halftoning: Automatic evaluation and optimization," *Interdisciplinary Information Sciences,* 8(2):219–234, 2002.

[113] T. J. Schaefer, "On the Complexity of Some Two-Person Perfect-Information Games," *Journal of Computer and System Sciences,* 16:185–225, 1978.

[114] M. Sharir and E. Welzl, "On the number of crossing-free matchings, cycles, and partitions," *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 860–869, 2006.

[115] M. Sharir and E. Welzl, "Random Triangulations of Planar Point Sets," *Proc. 22nd Ann. ACM Symp. Computational Geometry,* pp. 273–281, 2006.

[116] T. Shermer, "Recent results in art galleries," *Proc. IEEE,* 80:1384–1399, 1992.

[117] A. Sinclair, "Algorithms for Random Generation & Counting: A Markov Chain Approach," *Birkhäuser,* 1993.

[118] C. Sohler, "Generating Random Star-Shaped Polygons," *Proc. 11th Canadian Conference on Computational Geometry,* pp. 174–177, 1999.

[119] M. Thorup, "Map graphs in polynomial time," *Proc. 39th IEEE Symposium on Foundations of Computer Science,* pp. 396–405, 1998.

[120] R. L. Tobin, T. L. Friesz, T. Miller, "Existence theory for spatially competitive network facility location models," *Ann. Oper. Res.,* 18:267–276, 1989.

[121] R. Uehara, and S. Iwata, "Generalized Hi-Q is NP-complete," *Trans. IEICE* E73:270–273, 1990.

[122] R. Ulber, "On The Number Of Star-Shaped Polygons And Polyhedra," *Proc. 11th Canadian Conference on Computational Geometry,* pp. 170–173, 1999.

[123] R. Ulichney, "Digital Halftoning," *MIT Press,* 1987.

[124] V. V. Vazirani, "Approximation Algorithms," *Springer-Verlag,* Berlin, 2001.

[125] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," *New Results and New Trends in Computer Science,* vol. 555, Lecture Notes in Computer Science, Springer, Berlin, pp. 359–370, 1991.

[126] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. B. Mitchel. "Generating Random Polygons with Given Vertices," *Computational Geometry Theory and Application,* 6(5):277–290, 1996.

Publications

[1] 寺本幸生: “ハードウェア支援による各種ボロノイ図の高速描画とその応用,” *Technical Report of IEICE, COMP2002-45,* (Nov. 2002).

[2] S. Teramoto, and T. Asano: “Fast Implementation of Laguerre Voronoi Diagram with Hardware Assistance,” *Proc. International Symposium on Voronoi Diagrams in Science and Engineering,* pp. 165–171 (Sep. 2004).

[3] S. Teramoto, T. Asano, B. Doerr, and N. Katoh, “Inserting Points Uniformly at Every Instance,” *Proc. 2005 Korea Japan Joint Workshop on Algorithms and Computation (WAAC 2005),* pp. 3–9, (Aug. 2005).

[4] S. Teramoto, and R. Uehara, “Voronoi game on graphs and its complexity,” *IPSJ SIG Technical Report, 2006-AL-104-2,* pp. 9–16, (Jan 2006).

[5] S. Teramoto, M. Motoki, R. Uehara, and T. Asano, “Heuristics for Generating a Simple Polygonalization,” *IPSJ Technical Report, 2006-AL-106-6,* pp. 41–48, (May 2006).

[6] S. Teramoto, E. D. Demaine, and R. Uehara, “Voronoi game on graphs and its complexity,” *2nd IEEE Symposium on Computational Intelligence and Games (CIG 2006),* pp. 265–271, (May 2006).

[7] R. Uehara, and S. Teramoto, “The complexity of a Pop-up book,” *IPSJ SIG Technical Report, 2006-AL-107-10,* pp. 59–64, (Jul. 2006).

[8] R. Uehara, and S. Teramoto, “The complexity of a Pop-up book,” *18th Canadian Conference on Computational Geometry (CCCG 2006)*, pp. 3-6, (Aug. 2006).

[9] S. Teramoto, T. Asano, N. Katoh, and B. Doerr, “Inserting Points Uniformly at Every Instance,” *IEICE Trans. Inf. & Syst.* VOL. E89-D, No. 8 AUGUST 2006, 2348–2356 (Aug. 2006).

[10] R. Uehara and S. Teramoto, "Computational Complexity of a Pop-up book," *4th International Conference on Origami in Science*, Poster, (Sep. 2006).

[11] S. Teramoto, E. D. Demaine, and R. Uehara, "Voronoi Game on Graphs and its Complexity," submitted to *Theoretical Computer Science*.