THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Doctor of Philosophy (Computer Engineering)

DEGREE

Computer Engineering

FIELD

Computer Engineering

DEPARTMENT

TITLE: Algorithms for Counting Problems in Geometry and Combinatorics

NAME: Mr. Nattawut Phetmak

THIS THESIS HAS BEEN ACCEPTED BY

..... **THESIS ADVISOR**

(Assistant Professor Jittat Fakcharoenphol, Ph.D.)

..... **THESIS CO-ADVISOR**

(Associate Professor Pradondet Nilagupta, M.Eng.)

..... **THESIS CO-ADVISOR**

(Assistant Professor Thanawin Rakthanmanon, Ph.D.)

..... **DEPARTMENT HEAD**

(Associate Professor Punpiti Piamsa-Nga, D.Sc.)

..... **DEAN**

(Associate Professor Srijidtra Charoenlarpnopparut, Ph.D.)

THESIS

ALGORITHMS FOR COUNTING PROBLEMS IN GEOMETRY AND COMBINATORICS

NATTAWUT PHETMAK

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy (Computer Engineering)
Graduate School, Kasetsart University
2023

Nattawut Phetmak 2023: Algorithms for Counting Problems in Geometry and Combinatorics. Doctor of Philosophy (Computer Engineering),
Major Field: Computer Engineering, Department of Computer Engineering.
Thesis Advisor: Assistant Professor Jittat Fakcharoenphol, Ph.D. 81 pages.

We consider three problems at the intersection between algorithms and geometry. These problems essentially are counting problems in their respective domains.

The first problem consider folding and unfolding many points on a sheet of paper with the goal of computing the region bounded by creases. Akitaya, Ballinger, Demaine, Hull, and Schmidt (2021) previously consider folding corners to a fixed point within the sheet of paper. We extend their problem by consider folding every points on the boundary. Our contribution is an analysis on the shape and structure of the region. We also give a linear-time algorithm for finding the region.

The second problem, related to wireless sensor networks, is the k-sink minimum movement target coverage problem. In this problem, we would like to employ sensors from k stations to cover all the targets with the minimum aggregate sensor movement. Gao, Chen, Wu, and Chen (2017) previously give an $n^{O(1/\epsilon^2)}$-time approximation scheme for the problem. We improve the running time to $n^{O(1/\epsilon)}$.

The third problem is to uniformly generate a derangement with a specific number of cycles. Mendonça (2020) previously give a linear-time algorithm for generating random derangements without the constraint on number of cycles. We give an $O(n^{2.5} \log n)$ time for generating a random derangement of n items with exactly m number of cycles.

_____ _____ ____ / ____ / ____

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENTS

My thesis odyssey would not have arrived ashore without my thesis advisor, Jittat Fakcharoenphol, who endured my laziness and forgave my stubbornness. “Choose the right problem” will always be my favorite lesson. I am eternally grateful.

Next, I would like to express my gratitude to my lifelong friend, Supanut Chaidee, who introduced me to the world of mathematics since my childhood. It is my honor that we share memorable moments, including my thesis defense.

I would like to show appreciation to my co-advisors, Pradondet Nilagupta, who share many great wisdom, and Thanawin Rakthanmanon, who taught me applications IRL. I must thank my co-authors, Chaiporn Jaikaeo and Nonthaphat Wongwattanakij, for their joint research. And I must also thank Hugo A. Akitaya, Otfried Cheong and Suppakitpaisarn Vorapong for the discussions during the conferences. Thanks should also go to Hiro Ito and Hee-Kap Ahn, who organized the conferences and workshops.

My journey would be lifeless without my lab-mates at the Theory Research Group, including but not limited to Pongsakorn Ajchariyasakchai, Sirakorn Lamyai, Tanee Kumpijit, Pattara Sukprasert, Attakorn Putwattana, Phanu Vajanopath, Kunanon Burathep, and Patinya Sangiamchit. I also need to mention online communities, which served me as a second laboratory, where I have met Sorawee Porncharoenwase, Nat Sothanaphan, Panas Kalayanamit, and others. Thank you for great times not only in discussing serious mathematics, but also in enjoying entertaining hobbies.

Finally, I am grateful to have been born into this world and raised with love. Thank you, Mom and Dad.

Nattawut Phetmak

July 2023

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF PUBLICATIONS

ALGORITHMS FOR COUNTING PROBLEMS IN GEOMETRY AND COMBINATORICS

STRUCTURE OF STUDY

This thesis studies various algorithms in diverged fields, which consists of geometry and combinatorics. The algorithms considered are for geometric problems in folding and movement planning. Additionally, efficient algorithms for generating random combinatorial objects are also studied. Essentially, the problems studied in this thesis are counting problems.

Three results are presented.

The first result focus on the mathematics of paper folding. We give an extension to the family of *fold and unfold* problems, precisely the problems established by Akitaya, Ballinger, Demaine, Hull, and Schmidt (2021). They consider two scenarios for folding a sheet of paper then observe an inner region bounded by creases. Their first problem is folding every corner to a point within, result in a region bounded by Voronoi cell. Their second problem is folding every edges to a line segment within, result in a region governed by a straight skeleton. Our proposed problem extend their first problem by folding every point on the boundary of the sheet of paper instead. We study the shape of the region, give a linear time algorithm to find the region, and finally analyse the structural complexity of the region, i.e., we provide structural result for counting the number of arcs in the output regions. This chapter is a joint work with Jittat Fakcharoenphol and appeared in (Phetmak and Fakcharoenphol, 2023a).

The second result study a coverage problem in wireless sensor network, specifically, the k-sink minimum movement target coverage problem. We are interested in designing the polynomial-time approximation scheme (PTAS) algorithm for this problem. The previous algorithm by Gao, Chen, Wu, and Chen (2017) runs in time $n^{O(1/\epsilon^2)}$. We improve this running time to $n^{O(1/\epsilon)}$. To obtain the bound on the running time, we

have to give an upperbound on the number of sensors in the optimal solution; again this is a counting problem as well. This chapter is a joint work with Nonthaphat Wongwattanakij, Chaiporn Jaikaeo, and Jittat Fakcharoenphol and appeared in (Wongwattanakij *et al.*, 2023).

The last result consider a problem of generating a derangement, which is a specific permutation that contains no fixed-point. Mendonça (2020) give a linear-time algorithm for generating a general random derangement, i.e., a derangement without constraint on the number of cycles. However, since the number of cycles tends toward standard distribution, their algorithm is not suitable to generate a derangement with a specific number of cycles. We propose an $O(n^{2.5} \log n)$ algorithm for generating a derangement of size n with m cycles. As in many uniform generation algorithms, the key insight lies in the way we could decompose the construction of derangements so that we can count the number of required derangements while enumerating them. This chapter is a joint work with Jittat Fakcharoenphol and to be appear in (Phetmak and Fakcharoenphol, 2023b).

INTRODUCTION

Counting is a fundamental operation in human civilization. Some counts are physically straightforward, e.g., counting fruits in a basket. Some counts must be approximated, e.g., counting atoms in a cube. Some counts need conceptual enumeration, e.g., counting different ways to deal a deck of cards. Some counts rely on computational structure, e.g., counting regions in a circle divided by chords.

Background and Rationale

When designing an algorithm, counting is the very first question. Since it determines the input/output size, which often leads to the evaluation of the complexity of the algorithm. We shall see the impact of counting, despite being approximated, in Wongwattanakij *et al.* (2023), where we give a bound to the problem; thus, guaranteeing the worst-case complexity of the algorithm.

On the other hand, counting can be tied to the algorithm itself. Especially in combinatorics algorithms, where we have to enumerate the possible ways to rearrange a set of objects. This aspect of counting can be seen in Phetmak and Fakcharoenphol (2023b), where are concerned about the permutation without a fixed point, then devised algorithms based on the counting knowledge.

Sometimes, counting is not an easy task to answer, so we may need to go around and answer other questions first. In Phetmak and Fakcharoenphol (2023a), we start with an algorithm to compute the region. Afterward, we use the algorithm to make sampling, then analyse its complexity, i.e., count how many sides the region possesses.

Objectives

To help devise/improve algorithms, which rely on counting.

Contributions and Outcome of Research

In Phetmak and Fakcharoenphol (2023a), when we try to answer the problem of counting the number of sides of the region, we uncover the structure of the region. Which in turn made the problem easier and yields a simpler algorithm.

In Phetmak and Fakcharoenphol (2023b), we extend a counting formula, called the *Stirling number of the first kind*. Specifically, we consider the case where the shortest cycles of the permutation must exists at least k times. We then devise algorithms for derangements from it.

Finally, in Wongwattanakij *et al.* (2023), we give a better counting bound to the previous work. It allows us to perform the dynamic programming technique, results in a faster algorithm.

We first look at the result from Wongwattanakij *et al.* (2023), which is a problem in mobile sensor network movement planning.

1. Overview of the Mobile Sensor Movement Problem

Wireless sensor networks (WSNs) are used in various types of applications such as military, industry, and disaster responses (Akyildiz *et al.*, 2002; Yick *et al.*, 2008). To effectively monitor a region of interest, sensor nodes are placed at specific locations so that they can collectively provide overall sensing coverage for the entire area (Wang, 2011; Wu *et al.*, 2020). WSNs may consist of static sensor nodes, mobile sensor nodes, or both. With presence of mobile sensor nodes, monitor areas may be dynamically adjusted by having these mobile nodes navigate to appropriate positions. This capability helps reducing the overall number of sensor nodes and/or provides better sensing coverage over regions of interest.

In this chapter we study the k-Sink Minimum Movement Target Coverage Prob-

lem, where we would like to schedule sensors, with sensing range r, from k stations to cover all targets, while minimizing the total distance for every sensor. Gao, Chen, Wu, and Chen (2017) proposed a polynomial-time approximation scheme for this problem that runs in time $n^{O(1/\epsilon^2)}$ and returns a solution whose cost is at most $(1+\epsilon)$ of the optimal cost when the targets are distributed uniformly and randomly in the surveillance region. This result first appeared as (Chen *et al.*, 2016) in INFOCOM'16.

Our contribution is as follows. We observe that, as in many PTAS's for other geometric problems, their general framework, based on exhaustive search and shifting, does not depend on the target distribution; thus it works without the distribution assumption. More over, we notice that the running time bound of their algorithm depends on an erroneous claim of the bound on the number of sensors needed in the optimal solution (in Theorem 3 in (Chen *et al.*, 2016) and in Theorem 5 in (Gao *et al.*, 2017)) for a particular region (see Subsection *"A Counter Example to the Sensor Bound in Gao* et al. *(2017)"*) and provide a correction. Our proof actually provides a stronger guarantee, allowing us to devise a more refined dynamic programming subroutine, the main contribution of this work, used in the algorithm that runs in time $n^{O(1/\epsilon)}$, which is an exponential improvement on the running time. While we consider our result a fairly theoretical one, we hope that the dynamic programming approach presented here might find more applications in other networking problems.

We describe the network model, problem statements and a review of Gao *et al.* (2017) (formerly Chen *et al.* (2016)) algorithm, called Energy Effective Movement Algorithm (EEMA), in Section *"Preliminary and Problem Statement"*. Section *"The Improved Algorithm"* describes the improved algorithm and proves the running time and its performance guarantee. Our running time analysis depends on the bound on the sensors needed in the optimal solution, proved in Section *"Bounds on the Number of Sensors"*.

2. Related Works

Coverage problems has been widely studied in computational geometry (O'Rourke, 1987). These problems are important to wireless sensor networks (see recent survey in (Wang, 2011) and the book by Wu *et al.* (2020). There are numerous heuristic algorithms for the placement and deployment problem, e.g., the algorithms for full coverage based on packing has been proposed by Wang and Tseng (2008) for areas without obstacles and by Wang, Hu, and Tseng (2008) for areas with obstacles, for coverage enhancement using virtual forces (Zou and Chakrabarty, 2004), for strategies that deals with coverage holes using Voronoi polygons (Mahboubi *et al.*, 2014), and for sensor relocation using Delaunay triangulation (Khampeerpat and Jaikaeo, 2017). This chapter as well as that of Gao *et al.* (2017) studies movement scheduling problems. For dynamic area coverage, Liu, Dousse, Nain, and Towsley (2013) considered coverage using mobile sensor networks. Liu *et al.* (2008) and He *et al.* (2012) considered scheduling problem when the goal is to protect an area using a barrier of sensors. Ammari (2012) tried to minimize sensor movement cost while ensuring area coverage. For strategies for improving coverage, Wang, Lim, and Ma (2009) provided a survey. Guo and Jafarkhani (2019) presented centralized and decentralized heuristics for optimizing sensor movement under network lifetime constraints. Elhoseny *et al.* (2018) employed the genetic algorithm as a heuristic to deal with coverage requirements.

When theoretical guarantees are needed, approximation algorithms for many versions of coverage problems have been studied. Section *"Polynomial-Time Approximation Scheme"* describes related work on polynomial-time approximation schemes on geometric graphs closely related to this work. In this section, we list a few notable works on approximation algorithms for coverage problems. For target coverage, if the goal is to schedule sensor activation groups to maximize the life time, Ding *et al.* (2012) devised a $(4+\xi)$-approximation algorithm for any $\xi > 0$. For restricted form of k-coverage, Xu and Song (2014) gave a 3-approximation algorithm. When a threshold for movement of each sensor is given and the objective is to maximize target coverage, a recent result by Liang, Shen, and Chen (2021) gave an algorithm with an approx-

imation ratio of $(1 - 1/e)$. Another important class of coverage problem is to find dominating sets in unit disk graphs. Marathe *et al.* (1995) described a 5-approximation algorithm for finding dominating sets and a 10-approximation algorithm for finding connected dominating sets. For weighed dominating sets, Erlebach and Mihalák (2010) and Zou *et al.* (2011) independently devised a $(4 + \epsilon)$-approximation algorithms. Zou *et al.* (2011) also present a $(5 + \epsilon)$-approximation algorithm for the connected version of the problem. A recent book by Wu *et al.* (2020) is an excellent source for these developments.

3. Preliminary and Problem Statement

3.1 Model and Problem Statement

Following Gao *et al.* (2017), the homogeneous network considered in this chapter can be modeled as $G = (T, P, r, S)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is the target set, $P = \{p_1, p_2, \ldots, p_k\}$ is the station set, r is the sensing range, and $S = \{s_1, s_2, \ldots, s_m\}$ is the mobile sensor set. The targets are static points on the plane that we want to cover using sensors with sensing range r. We say that a sensor set S *covers* a point t_i if there exists a sensor $s_j \in S$ such that $d(s_j, t_i) \leq r$, where $d(a, b)$ denotes the Euclidean distance between two points a and b on the plane. The coverage requirement ensures that every target $t_i \in T$ must be covered. Each sensor $s_j \in S$ must be scheduled to move to its location from some station $p_\ell \in P$; the movement distance is $d(s_j, p_\ell)$. Since the station set P is static we can easily find $p_\ell \in P$ that minimizes $d(s_j, p_\ell)$. We would later refer to the distance just as $d(s_j)$ defined to be $\min_{p_\ell \in P} d(s_j, p_\ell)$.

Gao *et al.* (2017) defined the k-*Sink Minimum Movement Target Coverage Problem* (k-MMTC) as follows. Given a set of targets T, station set P, and sensing radius r, find a set of sensors S covering T such that the total movement distance is minimized. They proposed an algorithm called Energy Effective Movement Algorithm (EEMA) that is claimed to be a PTAS that solves this problem (reviewed in Subsection *"Review of EEMA (Chen* et al.*, 2016; Gao* et al.*, 2017)"*).

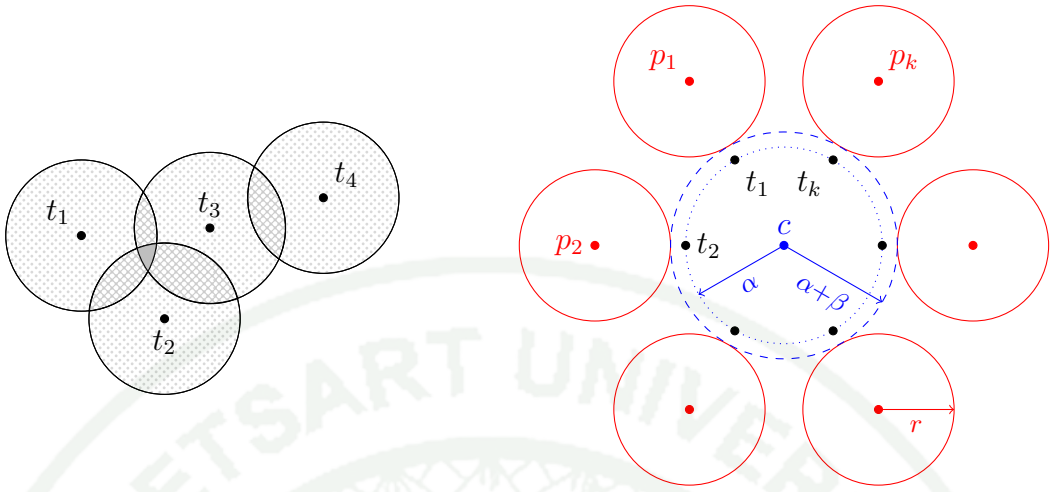
We would like to note that Gao *et al.* (2017) also assumed that the targets are distributed uniformly and randomly in the surveillance region. Our work does not need this requirement. In fact, we observe that their algorithm and proofs work even without this distribution assumption as well.

3.2 Polynomial-Time Approximation Scheme

A *polynomial-time approximation scheme* (PTAS) is an algorithm that, given a constant $\epsilon > 0$, returns a solution to the problem of cost within $(1 + \epsilon)$ of the optimal in time polynomial in n, the size of the problem, assuming that ϵ is a constant. Note that the running time may depend exponentially on $1/\epsilon$, i.e., the running time might increase very rapidly as the solution quality increases (as $1/\epsilon \to \infty$ when $\epsilon \to 0$).

A general framework for developing PTAS for geometric problems introduced by Hunt *et al.* (1998) follows the approach used in planar graphs by Baker (1994). First the problem is decomposed into much smaller subproblems which can be solved separately exhaustively either by dynamic programming or brute-force search. Then, the "locally-optimal" solutions from these subproblems are combined to obtain the final solution. Clearly, the combined solution might not be globally optimal because the dependency between each subproblem is neglected. However, one can typically ensure that the neglected cost is small, i.e., at most $1/\epsilon$ fractions of the optimal cost, when the size of each subproblem is large enough using an averaging argument. For geometric problems mostly on unit disk graphs, Hunt *et al.* (1998) consider an area of size $O(1/\epsilon) \times O(1/\epsilon)$ and use the fact that there is a feasible solution of size $O(1/\epsilon^2)$ to devise many PTAS's that run in time $n^{O(1/\epsilon^2)}$. For dominating sets, which is another form of coverage problems, if the nodes are unweighted, there are several PTAS for the standard problem (Hunt *et al.*, 1998; Nieberg and Hurink, 2006) and for the case when the dominating set is required to be connected (Zhang *et al.*, 2009). (See (Wu *et al.*, 2020) for a comprehensive treatment of this topics.)

The dynamic programming approach for covering problems is widely used.



(a) Detection cycles given 4 targets (b) A counter example that requires k sensors

Figure 1 Problem formulation as circles from sensors and targets

Erik Jan van Leeuwen (van Leeuwen, 2005, 2006, 2009) has employed roughly the same speed-up idea to the dominating set problem in unit disk graphs.

4. Review of EEMA (Chen *et al.*, 2016; Gao *et al.*, 2017)

EEMA proposed by Chen, Gao, Wu, and Chen (2016); Gao, Chen, Wu, and Chen (2017) works in two phases. The first phase divides the surveillance region into a set of subareas. This changes the nature of the problem from continuous optimization problem to a combinatorial one. The second phase approximates the solution to the combinatorial problem constructed in the first phase. We shall describe both phases and our contribution is essentially on the second phase of their proposed EEMA.

4.1 First phase of EEMA

For each target t_i, the *detection cycle* $D(t_i)$ is a circle of radius r centered at t_i. Note that a sensor in $D(t_i)$ covers t_i. Detection cycles may intersect; thus, a single sensor in the intersection can detect all targets associated with these cycles. The first phase of EEMA identifies all minimal subareas defined by intersections of detection cycles (see Figure 1a, which illustrated detection cycles for t_1, t_2, t_3, and t_4, with 9

minimal subareas to be considered). Each subarea is called a *curved-boundary.* Gao *et al.* (2017) proved the following lemma (implicitly in Theorem 2 in (Gao *et al.*, 2017)).

Lemma 1 ((Gao *et al.*, 2017))**.** *The number of minimal subareas defined by intersections of all* n *detection cycles is* $O(n^2)$

They also gave an $O(n^2)$-time algorithm (called the Graph Conversion algorithm) for finding all minimal subareas.

4.2 Second phase of EEMA

During the second phase, Gao *et al.* (2017) noted that placing a sensor anywhere in a particular minimal subarea covers the same set of target, defined as *covered-set* of that subarea. Therefore, to minimize the total distance, if one has to schedule a sensor to that subarea, the sensor must travel from the closest station. For the subarea σ_i, they gave an algorithm for finding $w(\sigma_i)$, the minimum distance from any point in σ_i to any station in P. Given the representation of all subareas, computing $w(\sigma_i)$ for all subareas σ_i takes only $O(n^2k)$.

After all these preprocessing steps, we are left with a combinatorial problem of choosing a subset of subareas to cover all targets.

Let Q be the bounding box containing all targets. As in many geometric PTAS, Gao *et al.* (2017) then employed the algorithm, called the Partition algorithm, that divides Q into cells $cell(Q)$, each of size $2mr \times 2mr$, solves the selection optimally for each cell $e \in cell(Q)$, and returns the union of selected subareas.

The essential subproblems for the Partition algorithm deals with each cell e. To solve each subproblem, Gao *et al.* (2017) used a brute-force search that runs in time $n^{O(m^2)}$. To bound the running time they proved the following key lemma (implicitly in Theorem 5).

Lemma 2 (Chen *et al.* (2016); Gao *et al.* (2017))**.** *The number of sensors (and subareas selected) in the optimal solution in each cell e is at most $O(m^2)$.*

However, this lemma is incorrect (see Subsection *"A Counter Example to the Sensor Bound in Gao* et al. *(2017)"*). We provide a fix to this lemma (in Section *"Bounds on the Number of Sensors"*) that implies that the number of sensors is $O(m^2 + k)$, yielding that their algorithm for each subproblem actually runs in time $n^{O(m^2+k)}$. More generally, if we have an upperbound of L on the number of sensors, a brute-force algorithm would run in time $n^{O(L)}$. We also provide a stronger bound that allows us to improve the running time to $n^{O(m+k)}$, improving the running time exponentially.

After solving each subproblem, each solution is combined straight-forwardly. Gao *et al.* (2017) used shifting techniques to show that if the targets are distributed uniformly in Q (as described in the model in Section III-A), the total cost is $(1 + 3/m)$ times the optimal cost, providing the guarantee of $1 + \epsilon$ of the PTAS. We note that their proof works even without the assumption on the distribution of the targets, for completeness, we also provide the proof of this performance guarantee in Section *"Shifting and the Performance Guarantee"*.

5. A Counter Example to the Sensor Bound in Gao *et al.* (2017)

The algorithm proposed by Gao *et al.* (2017) (first appeared as Chen *et al.* (2016)) relies on the fact that the number of sensors in the optimal solution required for each cell e of size $2mr \times 2mr$ is at most $O(m^2)$. Note that this bound is independent of k, the number of stations. The proof presented in the chapter uses the fact that there exists a feasible solution with that many sensors. However, the actual cost of k-MMTC is the total distance, not the number of sensors. In this section, we provide a counter example to the sensor bound in Theorem 5 in (Gao *et al.*, 2017) that requires k sensors.

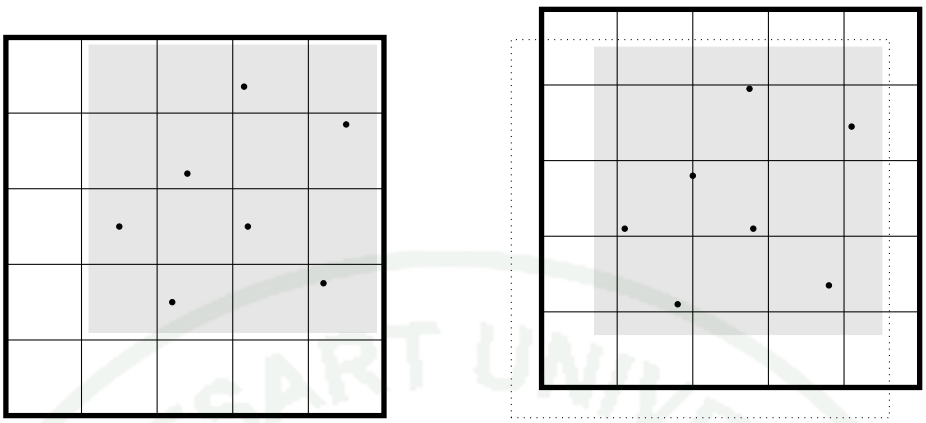
We assume that m is large enough, which is true when the error requirement ϵ is required to be small enough as $m = \Omega(1/\epsilon)$. Let c be the point at the center of cell e. For any $\alpha > 0$ and very small $\beta > 0$, place k targets uniformly at distance α from c and for each target t_i place a station p_i at distance $r + \alpha + \beta$ from c in the same direction from c as t_i. Figure 1b illustrates this example. Consider a solution that for each target t_i we move a sensor from p_i with distance β to cover it, resulting in the cost of $k\beta$ with k sensors. If one wants to use smaller number of sensors, some sensor is required to moved further with distance that depends on α and k; thus we can always choose β to be small enough so that that extra distance is larger than $k\beta$. This ensures that the optimal solution needs at least k sensors as claimed.

6. The Improved Algorithm

We focus on the second phase of EEMA where we work entirely on the combinatorial version of the problem (after the transformation using the Graph Conversion algorithm of (Chen *et al.*, 2016; Gao *et al.*, 2017)). Recall that we are given a set of h minimal subareas $R = \{\sigma_1, \sigma_2, \ldots, \sigma_h\}$ defined by intersections of detection cycles. Recall also that $h = O(n^2)$. Also for each subarea σ_i, we also know the minimum distance $w(\sigma_i)$ needed if we schedule a sensor into this subarea. The goal is to select a subset of subareas $R' \subseteq R$ that minimizes the total distance while ensuring that these subareas cover all targets.

Our main subroutine works with a square region of size $2mr \times 2mr$, referred to as a *cell*. In Subsection *"Dynamic Programming"*, we present our key contribution, an algorithm for solving the problem optimally when all targets are inside a cell that runs in time $n_e^{O(m+k)}$, where n_e is the number of subareas that can cover targets in e. Since later on we set $m = \Theta(1/\epsilon)$, this implies the total running time of $n^{O(1/\epsilon)}$, an improvement over the brute-force algorithm of (Gao *et al.*, 2017) that runs in time $n^{O(1/\epsilon^2)}$.

To solve the whole problem, the surveillance region is partitioned into many cells, where the dynamic programming can be applied to. Let Q be the bounding box



(a) The original position of the bounding box Q.

(b) The shifted bounding box Q_f and its cells $cell(Q_f)$.

Figure 2 The input targets, the bounding box Q, and the set of all cells

covering the surveillance region. More specifically, assume that the lower-left corner of Q is at $(0, 0)$ and its upper-right corner is at (M, M). To facilitate the shifting, we further assume that targets are inside the square whose corners are $(2mr, 2mr)$ and (M, M), i.e., there are no targets whose x or y co-ordinates are less than $2mr$. We divide Q uniformly into square cells of size $2mr \times 2mr$. There will be $\lceil M/2mr \rceil \cdot \lceil M/2mr \rceil$ cells. Each cell is a square whose lower-left and upper-right corners are of the form $(i \cdot 2mr, j \cdot 2mr)$ and $((i + 1) \cdot 2mr, (j + 1) \cdot 2mr)$. Let $cell(Q)$ be the set of all cells. See Figure 2a.

The algorithm then applies the dynamic programming subroutine to every cell $e \in cell(Q)$, and returns the union of the selected subareas. Gao *et al.* (2017) proved that the solution returned is 4-approximate.

Using standard technique for PTAS design, as also in (Gao *et al.*, 2017), we shift the bounding box to "average out" the cost of the optimal to obtain the $(1 + \epsilon)$-approximate solution. We discuss this step in Subsection *"Shifting and the Performance Guarantee"* and present the proof of the performance guarantee.

Finally, using the bounds to be proved in Section *"Bounds on the Number of*

Sensors", we give the analysis of the running time in Subsection *"Running Time"*.

6.1 Dynamic Programming

In this section, we describe a dynamic programming that, for a given cell e of size $2mr \times 2mr$, finds the optimal schedule for sensors to cover all the targets in e. The cell e is partitioned into vertical strips each of size $r \times 2mr$ (defined formally later). If we know that the number of sensors in the optimal solution for each strip of e is at most L, the dynamic programming algorithm in this section runs in time $n_e^{O(L)}$, where n_e is the number of subareas containing any targets inside cell e. We give a proof in Lemma 3 that $L = O(m + k)$; since we assume that k is a constant, the running time of the dynamic programming is $n_e^{O(m)}$.

Because we use dynamic programming, we shall define other types of subproblems. To avoid confusion later in this section, we shall refer to this main subproblem in a cell as the *cell problem*. We further divide the cell into $2m$ vertical strips, each of size $r \times 2mr$. We refer to these strips as strips $H_1, H_2, \ldots, H_{2m}$. See Figure 3.

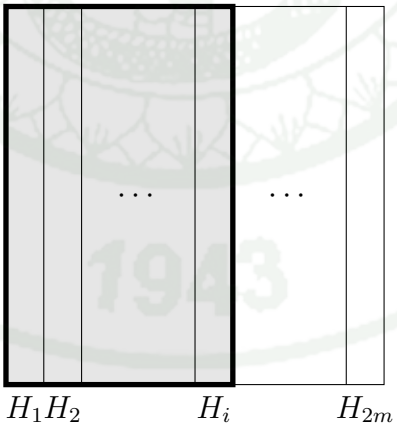


Figure 3 Strips $H_1, \ldots, H_{2m}$ and subproblem A_i (shown in shade)

We define the following subproblems $\mathcal{A}_i$ for each $1 \leq i \leq 2m$ as follows.

Subproblem $\mathcal{A}_i$: find subset of subareas R' that covers all targets in $H_1 \cup$

$H_2 \cup \cdots \cup H_i$ with the minimum total distance.

Let a_i denote the optimal total distance for $\mathcal{A}_i$. Recall that with this definition a_{2m} is the optimal cost for the cell problem. Here we only consider the algorithm that finds the minimum cost; to also find the actual solution, one only need to keep the best solutions together with its costs in the dynamic programming states.

For strip H_i, let $R_i \subseteq R$ be subset of subareas that cover some target in H_i. For a subset $U \subseteq R_i$ of subareas of size at most L covering some targets in H_i, we also define subproblems $\mathcal{B}_i(U)$ for each $1 \leq i \leq 2m$ as follows.

Subproblem $\mathcal{B}_i(U)$: find subset of subareas R' that covers all targets in $H_1 \cup H_2 \cup \cdots \cup H_i$ such that $R' \cap R_i = U$ with the minimum total distance.

Note that $\mathcal{B}_i(U)$ considers the same problem as $\mathcal{A}_i$ but with additional conditions that to cover targets in H_i one must use subareas in U. Let $b_{i,U}$ denote the optimal total distance for $\mathcal{B}_i(U)$. If it is impossible to do so, we let $b_{i,U} = \infty$. Therefore, we know that

$$a_i = \min_{U \subseteq R_i : |U| \leq L} b_{i,U}.$$

Hence, it is enough to solve $\mathcal{B}_i(U)$.

As the base case, to solve $\mathcal{B}_1(U)$ for each $U \subseteq R_1$, we can easily check if U covers all targets and keep its cost or keep the value of ∞ otherwise. Because $|U| \leq L$, there are at most $O(n_e^L)$ subproblems to consider. Since it takes $O(n^3)$ for each subproblem, the total running time is $n_e^{O(L)}$.

To solve $\mathcal{B}_i(U)$ for each $U \subseteq R_i$, first note that since the strip width is r, any subareas σ' covering any targets in H_{i-2} cannot be in R_i, i.e., they cannot help covering targets in H_i. On the other hand, subareas in R_i cannot cover any targets in $H_1, \ldots, H_{i-2}$. Therefore, when considering solutions, we only need to worry about

dependencies between subareas in $U \in R_i$ and $U' \in R_{i-1}$. Thus to find the solution for $\mathcal{B}_i(U)$, we enumerate all $U' \in R_{i-1}$ and checks

1. if U and U' are compatible (i.e., for subareas σ in both R_{i-1} and R_i, U and U' make the same decision, i.e., $\sigma \in U$ iff $\sigma \in U'$), and
2. if $U \cup U'$ covers all targets in H_i.

If both conditions are met, the set of subareas for $\mathcal{B}_{i-1}(U')$ and U form a feasible solution for $\mathcal{B}_i(U)$. Finally, we choose U' that minimizes the cost.

For each U, there are $O(n_e^L)$ subsets U' to consider. Therefore to solve all $\mathcal{B}_i(U)$ for all $U \in R_i$, the algorithm considers $O(n_e^L) \cdot O(n_e^L) = n_e^{O(L)}$, resulting in the running time of $n_e^{O(L)}$ as well. The total running time for solving all $\mathcal{A}_i$ is thus $m \cdot n_e^{O(L)} = m \cdot n_e^{O(m+k)}$, since $L = O(m+k)$.

6.2 Shifting and the Performance Guarantee

The shifting procedure proceeds in m rounds. For each round f, for $0 \leq f \leq m-1$, the bounding box Q is translated so that its lower-left corner is at $(2f, 2f)$. Let Q_f denote the translated bounding box and $cell(Q_f)$ be the cells in Q_f. Let S_f be the set of subareas returned from round f. The shifting procedure selects the solution with the minimum cost over returned solutions S_f's. See Figure 2b.

We shall prove that there exists round f such that the cost of S_f is $1 + 4/m$ of the optimal cost. Note that Gao *et al.* (2017) also presented a similar proof in Theorem 8 of (Gao *et al.*, 2017) (appeared as Theorem 6 in (Chen *et al.*, 2016)) but they did not realize that this proof works for general case without the target distribution assumption. We include the proof here for completeness.

Theorem 1. *The shifting procedure returns a solution whose cost is at most* $1 + 4/m$ *of the optimal cost.*

Proof. For any set of subareas S', let $w(S')$ be its cost, i.e,

$$w(S') = \sum_{\sigma \in S'} w(\sigma).$$

Let S^* be the optimal set of subareas and $w(S^*)$ be the optimal total movement cost. Consider round f and the returned solution S_f. For each $e \in cell(Q_f)$, for a set of subareas S', let $S'(e)$ be the set of subareas in S' that cover some target in e. From the optimality of the dynamic programming algorithm, we have that for any e,

$$\sum_{\sigma \in S_f(e)} w(\sigma) \leq \sum_{\sigma \in S^*(e)} w(\sigma).$$

For a cell e and a set of subareas S', let $S'(\delta(e))$ be the set of subareas in S' that cover some target in e and also intersect the boundary of e. We can split the cost of $S^*(e)$ into two parts, i.e.,

$$\sum_{\sigma \in S^*(e)} w(\sigma) = \sum_{\sigma \in S^*(e) \setminus S^*(\delta(e))} w(\sigma) + \sum_{\sigma \in S^*(\delta(e))} w(\sigma).$$

Note that for different $e, e' \in cell(Q_f)$, $S^*(e) \setminus S^*(\delta(e))$ and $S^*(e') \setminus S^*(\delta(e'))$ are

disjoint. Thus summing the cost for S_f, we have that $w(S_f)$ equals

$$\begin{aligned}\sum_e \sum_{\sigma \in S_f(e)} w(\sigma) &\leq \sum_e \sum_{\sigma \in S^*(e)} w(\sigma) \\ &= \sum_e \sum_{\sigma \in S^*(e) \setminus S^*(\delta(e))} w(\sigma) + \\ &\quad \sum_e \sum_{\sigma \in S^*(\delta(e))} w(\sigma), \\ &\leq w(S^*) + \sum_e \sum_{\sigma \in S^*(\delta(e))} w(\sigma),\end{aligned}$$

where e is summed over $e \in cell(Q_f)$, and the last inequality follows from the disjointness of subareas strictly inside a cell e.

Taking the average over $w(S_f)$ for all f, we get

$$\begin{aligned}\frac{1}{m}\sum_{f=0}^{m-1} w(S_f) &\leq \frac{1}{m}\sum_{f=0}^{m-1}\left(w(S^*) + \sum_{e \in cell(Q_f)} \sum_{\sigma \in S^*(\delta(e))} w(\sigma)\right) \\ &= w(S^*) + \frac{1}{m}\sum_{f=0}^{m-1}\left(\sum_{e \in cell(Q_f)} \sum_{\sigma \in S^*(\delta(e))} w(\sigma)\right).\end{aligned}$$

Consider a particular subarea $\sigma \in S^*$. Note that there are at most 4 cells e over all shifted instances such that $\sigma \in S^*(\delta(e))$, because σ may intersect at most one vertical cell boundary twice and another one horizontal cell boundary twice. This implies that the second term is at most

$$\sum_{f=0}^{m-1}\left(\sum_{e \in cell(Q_f)} \sum_{\sigma \in S^*(\delta(e))} w(\sigma)\right) \leq 4w(S^*).$$

Plugging this inequality into the previous bound, we have that

$$\frac{1}{m}\sum_{f=0}^{m-1} w(S_f) \leq w(S^*) + \frac{4}{m}w(S^*) = (1+4/m)w(S^*).$$

Since all $w(S_f)$'s are non-negative, at least one f exists such that $w(S_f) \leq (1+4/m)w(S^*)$, as required. □

6.3 Running Time

From Theorem 1, we need to set $m = 4/\epsilon$ to guarantee that the algorithm gives a $(1+\epsilon)$-approximate solution. For each shifted bounding box Q_f and for each cell $e \in cell(Q_f)$, the dynamic programming algorithm from Section *"Dynamic Programming"* runs in time $m \cdot n_e^{O(L)}$ where n_e is the number of subareas covering some target in e and L is the upper bound on the number of sensors needed.

In Section *"Bounds on the Number of Sensors"*, we show that $L = O(m+k) = O(m)$ since we assume that k is a constant. Therefore, the dynamic programming algorithm runs in time $n_e^{O(m)}$. Combining the running time for the dynamic programming for all cells in all shifted bounding boxes, we get that the algorithm runs in time

$$\sum_f \sum_{e \in cell(Q_f)} m \cdot n_e^{O(m)} = m^2 h^{O(m)} = m^2 n^{O(m)} = (1/\epsilon^2)n^{O(1/\epsilon)},$$

as claimed.

7. Bounds on the Number of Sensors

Our key result for bounding the running time of the dynamic programming in Section *"Dynamic Programming"* is the following lemma.

Lemma 3. *For each cell of size $2mr \times 2mr$, the number of sensors in a square of size $\sqrt{r/2} \times \sqrt{r/2}$ in an optimal solution is at most $O(1) + O(k) = O(k)$. Moreover, for any vertical strip of size $r \times 2mr$ in a cell, there exists an optimal solution that employs at most $O(m + k)$ sensors in this strip.*

The following two lemmas are sufficient to establish Lemma 3. Lemma 4 deals with sensors which are far from any stations, i.e., sensors whose moving distances are at least $r/2$. Lemma 5 considers sensors which stay close to a station. We remark that we make no attempt to optimize the constants in the proofs.

Lemma 4. *For each cell of size $2mr \times 2mr$, the number of sensors in a square q of size $\sqrt{r/2} \times \sqrt{r/2}$ in the optimal solution whose moving distances are at least $r/2$ is at most a constant.*

Proof. Note that sensors in q can cover targets in adjacent squares (all of size $\sqrt{r/2} \times \sqrt{r/2}$). However since the sensing radius is at most r, they cannot cover targets that much further. We draw additional 20 adjacent squares as shown in Figure 4. Note that moving a sensor inside q to any of these squares requires moving it for additional distance of r; thus increasing the cost by this much. However, since every sensor considered in this lemma has already moved for the distance of $r/2$, removing one sensor reduces the cost by $r/2$.

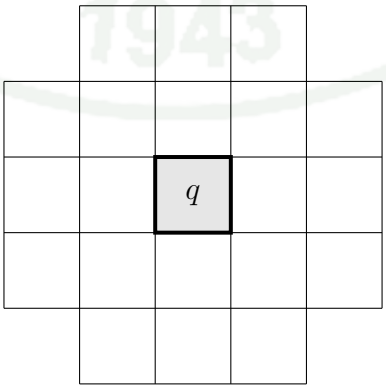


Figure 4 Adjacent squares inside which targets can be covered by sensors in q

We shall prove the lemma by contradiction. Let m_q be the number of sensors inside q. Assume that there are at least 63 sensors in q, i.e., $m_q \geq 63$. Consider another solution where we remove $m_q - 21$ sensors, arbitrarily, from q, and move another 20 sensors to all 20 adjacent squares. Note that the new schedule covers all previously covered targets. On the one hand, removing sensors decreases the cost by at least $(m_q - 21) \cdot r/2 \geq (63 - 21) \cdot r/2 = 20r$. On the other hand, moving sensors to adjacent squares increases the cost by at most $20r$. Thus, we obtain a cheaper solution and this contradicts the assumption. □

It remains to analyze sensors that are close to the station. The following Lemma 5 shows that for a station $p \in P$, there exists an optimal solution that schedules at most $O(1)$ sensors within the distance of $r/2$ from p. We again remind that we do not try to optimize the constant in the lemma.

Lemma 5. *In an optimal solution, the number of sensors whose moving distances are at most $r/2$ from a particular station p_ℓ is at most a constant.*

Consider station $p \in P$. Since it is free to place a sensor at p, we assume that all targets within the radius r of p are covered. We start with an important observation.

Lemma 6. *For any distance $0 < a \leq r/2$, the number of sensors originating from p whose distance from p is in the range $[a/2, a]$ is at most a constant.*

Proof. Sensors in this range may cover targets t whose distance $d(p, t)$ satisfies $r \leq d(p, t) \leq r + a$. Call the area in that distance a ring $Ring(r, r + a)$. Note that we can place 6 sensors around a circle of radius $2a \leq r$ to cover entirely $Ring(r, r + a)$, paying the cost of $12a$. Since each sensor whose distance to station p falls in the range $[a/2, a]$ must travel a distance of at least $a/2$, having more than 24 sensors in this range costs more than this simple feasible solution, leading to a contradiction. □

While the previous lemma is a good step in proving the bound, it can be applied

infinitely (as distances can be made smaller and smaller) resulting in an unbounded number of sensors. To resolve that we need a notion of progress. We first give an overview for the proof of Lemma 5.

To prove this lemma, we shall apply Lemma 6 repeatedly to the optimal solution. If we can show that the number of times we apply the lemma is constant, we know that the total number of sensors is also constant. Later in this section, we mainly state the required preliminary lemmas.

Consider the furthest sensor from station p in the optimal solution whose distance from p fall in the range $(0, r/2]$. Let a_1 be the distance and denote that sensor by s_1. Lemma 6 ensures that the number of sensors whose distance from p are in the range of $[a_1/2, a_1]$ is constant.

To apply Lemma 6 again, we look for the furthest sensor which is closer to p than $a_1/2$. Let a_2 be its distance. We repeat this process, yielding a sequence of distances $a_1, a_2, \ldots$, where

$$a_{i+1} \leq a_i/2.$$

These distances define a sequence of levels of sensors, where L_i is the set of sensors whose distances from p are in the range $(a_i/2, a_i]$.

To show that the number of levels is at most a constant, we define the following notion of covered angles to capture the progress. From the station p located at (x_p, y_p), fix a reference line ℓ to be a (half) straight line originating from (x_p, y_p) to (∞, y_p). For any target t, given p and line ℓ, we can define the angle θ_t to be the angle determined by segment $\overline{pt}$ and line ℓ. We say that t is *at angle* θ_t and we also refer to θ_t as the *angle* of t. We say that the interval $[\theta_1, \theta_2]$ of angles is *covered at radius* a' if every target at angle $\theta \in [\theta_1, \theta_2]$ at distance between r and a' is covered.

To be precise, we have to describe how to account for covered angles. When

dealing with sensors at level L_i, we consider a single sensor s in that level furthest from p. Let a be the distance from s to station p. At this level we consider the interval of angles covered by p at radius $a/2$. When we consider the next level L_{i+1}, we again consider a single sensor $s' \in L_{i+1}$ at furthest distance a' from p and try to account for covered angles at radius $a'/2$. Since previously covered intervals at radius $a/2$ remain covered at radius $a'/2 < a/2$; we can take all covered intervals from the previous level to this level as well. We note that while sensors at lower levels s may cover more angles at radius $a'/2$ (for higher level analysis), we keeps the covered intervals fixed when we analyze the additional covered angles; thus the intervals of covered angles act as lowerbounds on the actual angles covered.

Thus, to account for the progress, we maintain a set S of (circular) disjoint intervals of angles and the current guarantee radius α, with the condition that any target t with distance at most α from station p at angle θ_t such that $\theta_t \in I$ for some interval $I \in S$ is covered by the current set of sensors. We remark that if this condition holds for some set S with radius α, it also holds for any radius $\alpha' < \alpha$.

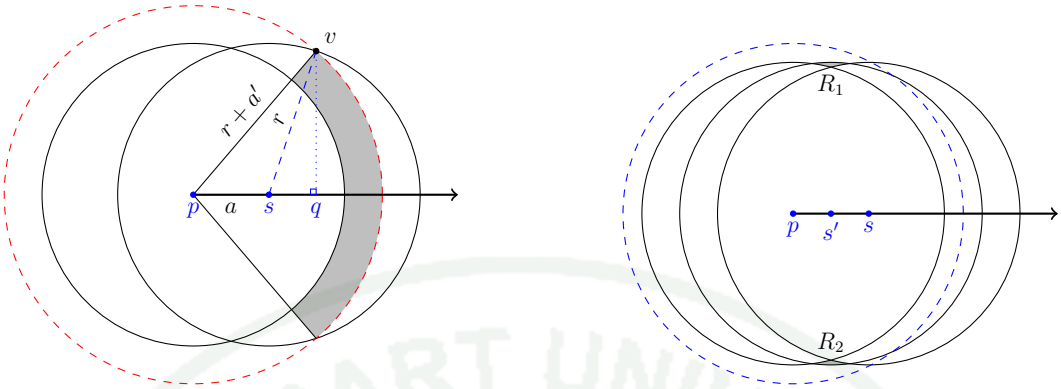
The following lemma shows that if we place a sensor at distance a from p, it would cover an interval of angles at radius a' of width at least $2 \cdot \arccos(13/20)$, where $a' \leq a/2$. (Also, see Figure 5a.)

Lemma 7. *Assume that station p is at $(0, 0)$. If we place sensor s at distance a from p, w.l.o.g. at $(a, 0)$, where $a \leq r/2$, the interval*

$$[-\arccos(13/20), \arccos(13/20)]$$

is covered at radius a', where $a' \leq a/2$.

Proof. Let s the sensor at distance a from p. Consider two circles C_p and C_s centered at p and s with radius r. Let C' be another circle centered at p of radius $r + a' \leq r + a/2$.



(a) A single sensor at s covers targets in angle interval $[-\theta, \theta]$

(b) Two uncovered regions R_1 and R_2 in $C_{s'} \setminus (C_p \cup C_s)$

Figure 5 Coverage region of a single sensor near the station

Since targets within distance $r + a'$ from p in C_s and C_p are covered, an uncovered target must be in $C' \setminus (C_s \cup C_p)$. The worst case would attain when C' is largest, so we assume that $a' = a/2$. Also, note that the covered interval would be smallest when a is largest; thus, we also assume that $a = r/2$. This case is shown in Figure 5a where a single sensor at $(a, 0)$ covers targets in angle interval $[-\theta, \theta]$. With the black circles are C_p and C_s; the dashed circle is C', and the covered area considered is shaded.

We calculate the angle θ of vps which is half the angle covered. From the assumptions, we know that the length of $\overline{pv}$ is $r + a' = r + r/4 = 5r/4$, the length of $\overline{sv}$ is r, and the length of $\overline{ps} = a = r/2$. Let q be the point on ℓ such that angle pqv is the right angle. Elementary calculation shows that the length of $\overline{sq}$ = $5r/16$. Thus, the angle θ equals

$$\arccos\left(\frac{r/2 + 5r/16}{5r/4}\right) = \arccos(13/20).$$

Since the covered interval is $[-\theta, \theta] = [-\arccos(13/20), \arccos(13/20)]$, the lemma follows. □

Lemma 7 essentially allows us to make progress from one level to the next. However, we need more structures to ensure that when we have circles from many levels. Consider two sensors s and s' at two consecutive levels (i.e., $s \in L_i$ and $s' \in$

L_{i+1}). Assume that s is at distance a from the station p where $a \leq r/2$; this also implies that s' is at distance $a' \leq a/2$ from p. We consider the regions covered additional by s'. Formally, let circles C_p, C_s, and $C_{s'}$ denoted circles of radius r centered at p, s, and s' respectively. The regions covered additionally by s' is $C_{s'} \setminus (C_p \cup C_s)$, and the blue dashed circle indicated the distance $r + a'/2$, are shown in Figure 5b. The following lemma, regarding these regions, is crucial.

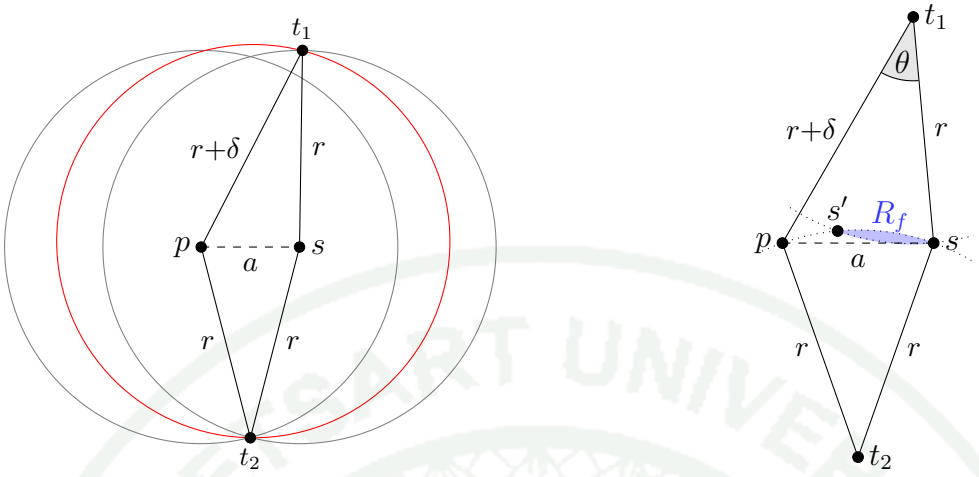
Lemma 8. *Let s and s' be sensors at levels L_i and L_{i+1} with distances a and a' from station p, respectively, where $a' \leq a/2 \leq r/4$. Consider $C_{s'} \setminus (C_p \cup C_s)$. If the set contains two disjoint regions R_1 and R_2, the further points in both regions is at distance at most $r + a'/2$ from station p.*

Proof. To prove this lemma, we assume that $C_{s'} \setminus (C_p \cup C_s)$ contains two regions R_1 and R_2 and there is some point t_1 in the regions at distance at least $r + a'/2$ from station p. Without loss of generality, we assume that region R_1 is the region of points with positive y co-ordinates and R_2 is the region of points with negative y co-ordinates. We further assume that $t_1 \in R_1$.

Let $\delta = a'/2 \leq a/4$. Given this configuration, we note that we can certainly move s' such that (1) t_1 is at the distance exactly at $r + a'/2 = r + \delta$ from p and also lies on the boundary of C_s, and (2) region R_2 contains exactly one point t_2 at the intersection of the boundary of C_p and C_s. Furthermore, this can be done without increasing the distance from s' to p. Figure 6a illustrates this assumption where $\delta = a'/2$. We note that s' is not shown.

Our goal is to show that for any point s' satisfying the assumption, s' must be too far from p; more precisely,

$$d(p, s') > 2\delta = a';$$



(a) Uncovered points $t_1 \in R_1$ and $t_2 \in R_2$

(b) The feasible region R_f and the nearest candidate point s'

Figure 6 Analysis of the feasible region for placing another sensor

thus, we have the contradiction as required.

Throughout this section, we consider a simple quadrilateral $\square pt_2st_1$ (see Figure 6a). In our analysis, we let p be at location $(0, 0)$ and s be at $(a, 0)$. If we fix t_1 and t_2, the point s' must be in $C_{t_1} \cup C_{t_2}$, to be refered to later on as the feasible region $R_f = C_{t_1} \cup C_{t_2}$. Without loss of generality, we assume that s' be the point in R_f closest to p. (See Figure 6b.) The following lemma states the location of s'.

Lemma 9. *Point s' is at*

$$\left(\frac{\delta^2 + 2r\delta + a^2}{2a} - \frac{a}{2}, \frac{\sqrt{((2r+\delta)^2 - a^2)(a^2 - \delta^2)}}{2a} - \frac{\sqrt{(2r)^2 - a^2}}{2} \right).$$

Proof. The location for t_2 is at

$$\left(\frac{a}{2}, -\frac{\sqrt{(2r)^2 - a^2}}{2} \right),$$

and the location for t_1 is at

$$\left(\frac{\delta^2 + 2r\delta + a^2}{2a}, \frac{\sqrt{((2r+\delta)^2 - a^2)(a^2 - \delta^2)}}{2a}\right).$$

Let m be a midpoint between t_2 and t_1, the coordinate for m is the average of coordinates for t_1 and t_2. We remark that $s \in R_f$. Furthermore, s is the furthest point in R_f from p. Thus, the point s' is the reflection of s on the point m; its coordinate can be calculated to be as stated given that s is at $(a, 0)$. □

With Lemma 9, considering only the x coordinates of s', we have that

$$\begin{aligned}
\frac{\delta^2 + 2r\delta + a^2}{2a} - \frac{a}{2} &= \frac{\delta^2 + 2r\delta + a^2 - a^2}{2a} \\
&= \frac{\delta^2 + 2r\delta}{2a} \\
&\geq \frac{\delta^2 + 2r\delta}{r} \\
&= \frac{\delta^2}{r} + 2\delta \\
&> 2\delta,
\end{aligned}$$

where the first inequality follows from the fact that $a \leq r/2$ and the second strict inequality is true because $\delta > 0$. Thus, $d(p, s') > 2\delta$ as required. □

Given Lemma 8, we can argue that if $C_{s'} \setminus (C_p \cup C_s)$ contains two regions, it is cheaper to place two sensor at distance strictly less than $a'/2$ from p; as the cost would be $2 \cdot a'/2 < a'$; contradicting the optimality of the optimal solution. Therefore, the additional covering area of $C_{s'}$ contains only one region. In that case, we can definitely rotate $C_{s'}$ to maximize the additional covering angle at distance $a'/2$ to be at least $\arccos(13/20)$, half of the angle guaranteed by Lemma 7.

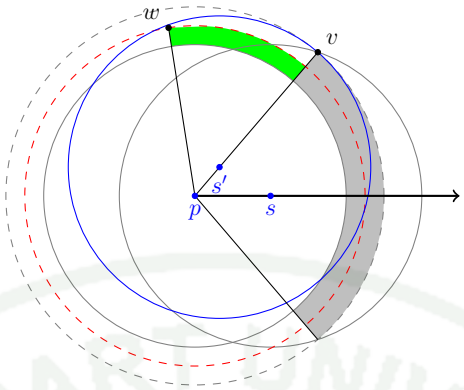


Figure 7 s' covers additional angle interval $[2\theta, \theta]$, where $\theta = \arccos(13/20)$

Figure 7 shows that s' (level L_{i+1}) at distance $a' = a/2$ from station p covers additional angle interval $[2\theta, \theta]$ at radius $a'/2$ (shown in green), where $\theta = \arccos(13/20)$.

Let us state the formal statement of the discussion above regarding two sensors at two consecutive levels.

Lemma 10. *Consider two sensor s and s' at consecutive levels L_i and L_{i+1}. Let a and a' be the distances from s and s' to p. Let the interval of angles covered by s at radius $a/2$ be I. The additional angle (w.r.t. I) covered by s' at radius $a'/2$ is at least* $\arccos(13/20)$.

Proof. Let C_s and $C_{s'}$ be two coverage circles for sensor s and s', and C_p be the circle of radius r centered at p. Lemma 8 implies that if the area $C_{s'} \setminus (C_s \cup C_p)$ contains two regions, they can be covered by two circles located at distance at most $a'/2$ from p, with the total cost of less than a'; but this leads to a contradiction.

Thus, $C_{s'} \setminus (C_s \cup C_p)$ contains only on region. Let v be the intersection of $C_{s'}$ and C_s outside C_p. (See Figure 7.) We claim that the additional angle covered by $C_{s'}$ is at least $\arccos(13/20)$. To see this, assume otherwise. In this case, the distance from v to p is less than $r + a'$; thus we can move s' along the circle of radius a' centered at p with the same movement cost while increasing the coverage area outside $C_s \cup C_p$. □

We are ready to prove our key Lemma 5

Proof. (of Lemma 5)

We show that the number of levels is constant. In our analysis we maintain a set S of disjoint covered angle intervals and the radius b.

Recall that $a_1, a_2, \ldots,$ are the furthest distances of sensors $s_1, s_2, \ldots,$ at levels $L_1, L_2, \ldots$ from station p. Initially, starting at level L_1, considering the furthest sensor $s_1 \in L_1$, from Lemma 7, we have that at radius $b = a_1/2$, s_1 covers the interval of width $2 \cdot \arccos(13/20)$. Thus, we have that S contains a single interval of width $2 \cdot \arccos(13/20)$ of covered angles at radius $b = a_1/2$.

As we consider sensors at higher levels, we make progress by either increase the total width of the intervals or decrease the number of intervals. Intuitively, we show that while iterating over these levels, one of these events occurs:

1. when considering a sensor s_i in level L_i with distance a_i, the sum of the width of the covered intervals at radius $a_i/2$ increases by at least a constant (more precisely at least $\arccos(13/20) \approx 0.863$), or

2. if s_i does not satisfy the previous property, it means that s_i is "between" two other sensors from the lower levels which are well-separated; in this case the show that the number of covered intervals decreases.

Consider sensor s_i at distance a_i at level L_i. We consider the intervals of covered angles at radius $b = a_i/2$. As discussed before every target whose angle is in any interval in S is covered at the new lower radius b. Let interval J_i be the interval of angles covered by s_i at radius $a_i/2$. Again from Lemma 7, we have that J_i is of width $2 \cdot \arccos(13/20)$.

We would like to add J_i to S. If J_i only intersects at most one interval J' already in S, we know that either the additional angle is at least $\arccos(13/20)$, or from Lemma 10, we can move s_i along the circle of radius a_i so that the additional angle after combining J_i with J' is at least $\arccos(13/20)$; resulting in event (1). The only case this is not true is when after moving s_i, J_i intersects two intervals; thus, adding J_i decreases the number of intervals in S and event (2) occurs.

Since the number of intervals cannot be less than 1, and each time event (1) occurs we increase the total width by a constant and introduce at most one new interval, we know that both events occurs for a constant number of times as well, because the total width is at most 2π. (To be precise, the number of levels is at most $\lceil 2 \cdot 2\pi / \arccos(13/20) \rceil = 15$.) This implies that the number of levels we encounter is at most constant, yielding the lemma. □

PUBLICATIONS

Publication 1

Phetmak, N. and J. Fakcharoenphol. 2023b. Uniformly generating derangements with fixed number of cycles in polynomial time. **Thai Journal of Mathematics**. 21 (SI): 1–17.

UNIFORMLY GENERATING DERANGEMENTS WITH FIXED NUMBER OF CYCLES IN POLYNOMIAL TIME

Nattawut Phetmak
Kasetsart University
nattawut.p@ku.th

Jittat Fakcharoenphol
Kasetsart University
jittat@gmail.com

Abstract

We study the uniform sampling of permutations without fixed points, i.e., derangements, that can be decomposed into m disjoint cycles. Since the number of cycles in a random derangement tends towards the standard distribution, rejection sampling may take exponential time when m largely deviates from the mean of $\Theta(\log n)$. We propose an algorithm for generating a uniformly random derangement of n items with m cycles in $O(n^{2.5} \log n)$ time complexity using dynamic programming with an assumption that all arithmetic operations can be done in time $O(1)$. Taking into account the arithmetic operations on large integers, the running time becomes $O(n^{3.5} \log^3 n)$. Our algorithm uses permutation types to structure our uniform generation of derangements.

1. Introduction

A *fixed point* in a permutation is an element that maps to itself. A *derangement* is a permutation without fixed points. The study of derangements can be traced back to 1708 when Pierre Rémond de Montmort proposed a problem for counting distinct derangements given n unique items. Together with Nicolaus I Bernoulli, they solved the problem half a decade after (de Montmort, 1713). Using the inclusion-exclusion principle, they showed that the number of derangements of n items, in its modern form, is $d(n) = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}$.

Naturally, the problem of generating derangements follows. Since the probability that a random permutation is a derangement is $d(n)/n! \approx 1/e$, a simple rejec-

tion sampling technique with a standard algorithm for generating random permutations works with an expectation of $O(1)$ trials until a derangement is generated. Recently, there have been efforts to efficiently generate a random derangement with smaller numbers of trials (Martínez *et al.*, 2008; Mendonça, 2020; Mikawa and Tanaka, 2017).

A permutation can be decomposed into disjoint cycles by tracing the result of applying the permutation repeatedly on each element. A cycle of length k is referred to as a k-cycle. From this definition, a derangement is a permutation with no 1-cycles. A permutation with only one cycle is called a *cyclic permutation*. When the number of elements is larger than one, a cyclic permutation is also a derangement.

In this chapter, we are interested in generating derangements with exactly m cycles. When $m = 1$, i.e., the goal is to generate a cyclic permutation; in this case, Sattolo (1986) gave an algorithm that runs in $O(n)$ time. For other values of m, since the number of cycles obeys Gaussian distributions (Flajolet and Soria, 1990), rejection sampling also works when m is close to the expectation, which is $\Theta(\log n)$. However, when m deviates badly from the expectation, the number of feasible derangements can be tiny, and it may take exponentially many trials to get the desired number of cycles. For example, when n is even and $m = n/2$, there are only

$$\frac{1}{(n/2)!}\binom{n}{2}\binom{n-2}{2}\cdots\binom{2}{2} = \frac{1}{(n/2)!}\binom{n}{2,2,\cdots,2} = \frac{n!}{(n/2)!\cdot 2^{(n/2)}}$$

derangements. The probability of successfully obtaining one from a random derangement is only $O(1/\sqrt{2^n})$.

In this chapter, we propose algorithms for uniformly generating a derangement of n items with m cycles in time $O(n^{2.5}\log n)$, assuming that all arithmetic operations can be done in $O(1)$ time. When accounting for arithmetic operations of large integers, the algorithm runs in time $O(n^{3.5}\log^3 n)$.

For a set of possible permutations $\mathcal{P}$, a *ranking function* for $\mathcal{P}$ is a bijection from $\mathcal{P}$ to $\{0, 1, 2, \ldots, |\mathcal{P}|-1\}$, and an *unranking function* is its inverse. Notable algorithms for ranking and unranking functions are permutations by Myrvold and Ruskey (2001) and derangements by Mikawa and Tanaka (2023). This chapter focuses only on devising an unranking function and uses it as a core routine to our algorithms.

While the Stirling number of the first kind (Comtet, 2012; Riordan, 2014) gives the number of permutations with m cycles, it does not give enough structure for reconstructing the i-th derangement with m cycles. Our key ingredient is a permutation type $\mathbf{t}$ (see Section 6.2 in (Comtet, 2012)) that lists the number of cycles $\mathbf{t}(k)$ for every cycle length k in a permutation. Our algorithm breaks down the random index i of the derangements into separate index $\mathbf{i}(k)$ for each cycle length k. With these details information on $\mathbf{i}$ and $\mathbf{t}$, we can reconstruct the required random derangement with m cycles.

While we work only on uniform generation, permutations, as well as derangements, can be weighted. One notable example of a weighting scheme is Ewens's Sampling Formula (Ewens, 1972) in population genetics (see, e.g., the work by da Silva, Jamshidpey, and Tavaré (2022)). Our approach can be adapted to deal with weighted derangements if one can sample permutation types correctly under the weighting scheme. Since our algorithm constructs and maintains partial types iteratively, if the weight function can be decomposed nicely with partially known types, we believe the approach here may be useful in that setting.

Section *"Preliminaries"* gives definitions and related algorithms. Two extensions of the Stirling number of the first kind that we need are also defined in this section. We present a simpler $O(n^3)$ algorithm in Section *"The $O(n^3)$ Algorithm"* based on dynamic programming. Using binary search, we obtain an improved algorithm with an $O(n^{2.5} \log n)$ running time described in Section *"The $O(n^{2.5} \log n)$ Algorithm"*. We discuss the issues with algorithmic implementation with large integers in Section *"Dealing with Large Numbers"*, resulting in an overhead of $O(n \log^2 n)$ time, primarily for com-

puting factorials. Finally, we present empirical data on the uniformity and running time of our algorithm in Section *"Experiments"*.

2. Preliminaries

In this section, we review definitions and results that we use. We also define two useful extensions on the Stirling numbers of the first kind.

2.1 Counting Permutations

A *permutation* is an arrangement of items. It is a bijection of a set onto itself. Let π denotes a permutation of set X, then $\pi(x)$ denotes an arrangement of an item x for $x \in X$. For convenience, when $x \notin X$, we let $\pi(x) = x$.

To write a permutation, we adopt Cauchy's two-line notation. For example if $\sigma(1) = 1$, $\sigma(2) = 3$, and $\sigma(3) = 2$, we write $\sigma = \left(\begin{smallmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{smallmatrix}\right)$.

It is natural to work with a sequence of permutation applications, e.g., applying a permutation σ *after* applied a permutation π. A *product* of a pair of such permutations, denotes as $\sigma{\cdot}\pi$, is a composition of them, i.e., $(\sigma{\cdot}\pi)(x) = \sigma(\pi(x))$. For example, $\left(\begin{smallmatrix} 2 & 4 \\ 4 & 2 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{smallmatrix}\right)$. Observe that for arbitrary permutations, the product does not commute. However, when both permutations are disjoint, i.e., they move different sets of items, the commutative property holds. We also write a product in a juxtaposition form $\sigma\pi$. The notation π^k denotes π composed onto itself for k times.

When we apply the same permutation repeatedly, the arrangement will eventually return to the initial arrangement. A *cycle* of a permutation π that contains x is a cyclic list $(x, \pi(x), \pi^2(x), \cdots, \pi^{k-1}(x))$, where k is the smallest integer greater than zero such that $x = \pi^k(x)$. We say that such a cycle has length k, or it is a *k-cycle*. A cycle of length one is called a *fixed point*.

A permutation may consist of more than one cycle. A permutation of n items with only one n-cycle is called a *cyclic permutation*, while a permutation that avoids creating any fixed points is called a *derangement*.

Given $n = kv$ distinct items, suppose that we would like to generate a permutation with v cycles each of length k. We may start by generating a partition of items into v disjoint sets, each of size k. We let $B(n, k)$, be the number of partitions; thus,

$$B(n,k) = \frac{1}{(n/k)!}\binom{n}{k,k,\cdots,k}. \tag{1}$$

To see this, note that $\binom{n}{k,k,\cdots,k}$ counts the number of ways one can choose a sequence of $v = n/k$ subsets of size k, so we divide it by $(n/k)!$ to obtain $B(n, k)$. We also note a recurrence relation $B(n,k) = \binom{n-1}{k-1}B(n-k,k)$. This sequence is useful in other contexts, and it appears as sequence A060540 in (Bottomley, 2001).

Our algorithm is structured around permutation types (Comtet, 2012). We say that a permutation is of *type* $\mathbf{t} = [\mathbf{t}(1), \mathbf{t}(2), \ldots, \mathbf{t}(n)]$, iff for $1 \leq k \leq n$, the permutation contains $\mathbf{t}(k)$ cycles of length k. Hence, the number of cycles m in a permutation is $\sum_k \mathbf{t}(k) = m$. It is known (see, e.g., (Comtet, 2012)) that the number of distinct permutations for n items of type $\mathbf{t}$ is

$$\frac{n!}{\prod_k k^{\mathbf{t}(k)}\mathbf{t}(k)!} = \frac{n!}{1^{\mathbf{t}(1)}2^{\mathbf{t}(2)}\cdots n^{\mathbf{t}(n)}\mathbf{t}(1)!\mathbf{t}(2)!\cdots\mathbf{t}(n)!}. \tag{2}$$

While permutation types give us insight on how cycles are formed, enumerating all types takes exponential time. The *Stirling number of the first kind* (Riordan, 2014), denoted by $s(n, m)$, is the number of distinct permutation of n items with

m cycles, which can be defined recursively as

$$s(n,m) = \begin{cases} 1 & \text{if } n = 0 \text{ and } m = 0, \\ 0 & \text{if } n \leq 0 \text{ or } m \leq 0, \\ (n-1)s(n-1,m) + s(n-1,m-1) & \text{otherwise.} \end{cases} \tag{3}$$

We shall define two related extensions of the Stirling number of the first kind. The first additionally takes the lower bound for the cycle length.

Definition 1. *The r-associated* Stirling number of the first kind *(Charalambides, 2002, Chapter 12), denoted by $s_r(n,m)$, is the number of permutations of n items with m cycles, where each cycle has length at least r.*

The following lemma states its recurrence.

Lemma 11.

$$s_r(n,m) = \begin{cases} 1 & \textit{if } n = 0 \textit{ and } m = 0, \\ 0 & \textit{if } n \leq 0 \textit{ or } m \leq 0, \\ (n-1)s_r(n-1,m) + \frac{(n-1)!}{(n-r)!}s_r(n-r,m-1) & \textit{otherwise.} \end{cases} \tag{4}$$

Proof. Similar to (3), this lemma can be proved by considering the last case. The first term is when we add a new item to a permutation of size $n-1$ with m cycles each of length at least r, resulting in a permutation where item n belongs to a cycle of length greater than r. The second term chooses a cycle of length exactly r that contains n. □

By definition, $s_1(n,m)$ is the original Stirling number of the first kind, while $s_2(n,m)$ only counts derangements with m cycles.

We can compute $s_r(n, m)$ using dynamic programming for a fixed r. Assuming that all factorials have been precomputed, since a single entry for $s_r(n, m)$ only looks up $O(1)$ other entries with smaller indices, the running time is $O(mn) \leq O(n^2)$

Another extension considers the number of occurrences of the shortest cycles. Note that this *partial-type Stirling number of the first kind* basically gives recurrence for computing the number of derangements based on permutation types.

Definition 2. *The* partial-type Stirling number of the first kind*, denoted by $s_k^v(n, m)$, is the number of permutations of n items with m cycles, where the smallest cycles have length exactly k and appear exactly v times.*

The following lemma states the formula for $s_k^v(n, m)$, which is essentially an unpack of the analysis of (2).

Lemma 12.

$$s_k^v(n, m) = \frac{n!}{k^v v!(n-kv)!} s_{k+1}(n-kv, m-v). \tag{5}$$

Proof. The coefficient $\frac{n!}{k^v v!(n-kv)!}$ counts the number of ways one can choose v cycles of length exactly k, and $s_{k+1}(n-kv, m-v)$ counts the number of ways for choosing the rest of the permutation. □

Note that when one have already computed $s_{k+1}(n, m)$ and all the necessary factorials, it only takes $O(1)$ to find $s_k^v(n, m)$. With all intermediate values of $s_{k+1}(\cdot, \cdot)$ from the computation of $s_{k+1}(n, m)$, one can also compute a list of $s_k^v(n, m)$ for all $0 \leq v \leq \lfloor \frac{n}{k} \rfloor$ under the same time limit.

2.2 Permutation Algorithms

We introduce 3 subroutines for permutation and partition generation. The first two functions UNRANKCHOOSE and UNRANKPERIODICCHOOSE are related to partition generation based on $B(n, k)$ shown in (1). The third function is fundamentally the cyclic permutation algorithm of Sattolo (1986). All three subroutines are essentially unranking functions that map integers to their corresponding permutations. We remark that these functions are discussed in (Arndt, 2010; Nijenhuis and Wilf, 2014). Throughout the chapter, we use zero-based numbering when referring to the i-th item in some ordering and also when referring to items in lists or arrays.

We refer to a subset of size k as a k-subset. The first function chooses the i-th k-subset from a list of n items. Figure 8 shows function UNRANKCHOOSE(X, k, i) that takes array X with n unique items, the number k of items to choose, and index i, and returns the i-th k-subset Y of X as a list of items. If array X is sorted, subset Y is the i-th lexicographically smallest subset. For example, let X be an array $[A, B, C, D, E]$. Choosing $k = 3$ items at index $i = 0$ yields $[A, B, C]$, choosing at index $i = 1$ yields $[A, B, D]$, and choosing at the last index $i = \binom{5}{3} - 1$ yields $[C, D, E]$.

Function UNRANKCHOOSE(X, k, i) works in $O(n)$ time, since it recurses for at most $O(n)$ times and each call runs in $O(1)$ time. The term $\binom{n-1}{k-1}$ can be computed in $O(1)$ time, provided that all factorials have been precomputed. We remark that this function can be improved to run in time $O(k \log n)$ using binary search.

The next function UNRANKPERIODICCHOOSE(X, k, i), shown as Figure 8, essentially generates the i-th partition of X into v subsets of size k using function UNRANKCHOOSE. It takes array X with n unique items, the size of subsets k, where $n = kv$, and an index i such that $0 \leq i < B(n, k)$. It returns the i-th partition of X into an unordered list v subsets of size k. Again if array X is sorted, the partition returned is the i-th lexicographically smallest partition. The function works by repeatedly calling UNRANKCHOOSE with appropriate indices.

Function UNRANKPERIODICCHOOSE runs in $O(n^2/k)$ time, because there are $O(n/k)$ recursive steps and each step runs in time $O(n)$. However, if we use the $O(k \log n)$ implementation of UNRANKCHOOSE, this function may takes $O(n \log n)$ running time instead. Note that since items in X and Y are stored in ascending order, finding the array $X \setminus Y$ takes $O(n)$ time.

In each recursive call to UNRANKPERIODICCHOOSE, it chooses one k-subset from X. Recalled the relation for $B(n,k)$ from (1), there are $\binom{n-1}{k-1}$ ways to partition k-subset in this level, leaving $n-k$ items with $B(n-k,k)$ ways to generate the rest of the partition. Therefore to generate the i-th partition, we write i as $\binom{n-1}{k-1} \cdot j + i'$ and find the i'-th subset of size k at this level and recursively find the j-th partition of the rest of the array. This is essentially the implementation of the recursive form of $B(n,k)$ in (1). Note that index i' ensures that UNRANKCHOOSE would choose the first item of X; thus, the subsets in the partition generated is unordered as claimed.

The last subroutine generates the i-th cyclic permutation, which based on Sattolo's random cyclic permutation algorithm (Sattolo, 1986). Figure 8 describes the function UNRANKCYCLICPERMUTE(X, i), which takes array X of n unique items and an index i. It permutes the given items into the i-th cyclic permutation. Unlike previous subroutines, this function does not preserve lexicographical order property. This is a trade-off for ease of implementation, which result in a simple algorithm with $O(n)$ time complexity.

3. The $O(n^3)$ Algorithm

In this section, we describe our $O(n^3)$-time algorithm to find the i-th derangement with m cycles. To find the required random derangement with this algorithm, one first compute $s_2(n,m)$ in time $O(n^2)$, then choose a random index $0 \leq i < s_2(n,m)$ as an input to the unranking function.

We briefly describe our approach based on dynamic programming. Typically,

```
function UnrankChoose(X, k, i)
    if k = 0 then
        return [ ]                                   // empty array
    x ← X[0]                                  // let x be the first item
    X' ← X[1, ...]      // let X' be the rest of the array
    j ← i − C(n−1, k−1)
    if j ≥ 0 then
        return UnrankChoose(X', k, j)                    // Skip x
    return [ x ] + UnrankChoose(X', k−1, i)          // Choose x
function UnrankPeriodicChoose(X, k, i)
    if X = ∅ then
        return [ ]
    (j, i') ← DivMod(i, C(n−1, k−1))
    Y ← UnrankChoose(X, k, i')
    return [ Y ] + UnrankPeriodicChoose(X\Y, k, j)
function UnrankCyclicPermute(X, i)
    X' ← a copy of array X
    for k ∈ [n−1, n−2, ··· , 1] do
        (i, j) ← DivMod(i, k)
        swap X'[j] with X'[k]
    return X'
```

Figure 8 Three base permutation functions

if one could count the number of objects with some property, one could follow the step backward to regenerate the i-th object. Our algorithm follows the same principle. While the 2-associated Stirling number of the first kind $s_2(n, m)$ counts the number of derangements with m cycles, it does not give us enough structure for reconstruction. Our key ingredient is a permutation type $\mathbf{t}$ (see Section 6.2 in Comtet (2012)). Instead of using recurrence for $s_2(n, m)$ to count derangements, we use recurrences for permutation types, more specifically the partial-type $s_k^v(\cdot, \cdot)$, to do so, and by reconstructing backward from the smallest cycles, one can reconstruct the complete i-th derangement. To see this, consider the topmost level of the recursion where we compute $s_2(n, m)$ as

$$s_2(n, m) = s_2^0(n, m) + s_2^1(n, m) + s_2^2(n, m) + \cdots + s_2^m(n, m). \tag{6}$$

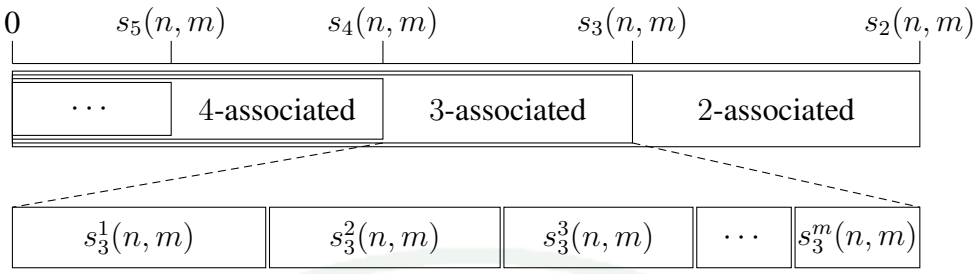


Figure 9 Type structure of the derangement

Suppose that we want to find the i-th derangement. Since it is one of the $s_2(n,m)$ derangements, it is counted in one of the term $s_2^v(n,m)$ in the summation above. If we find v such that

$$\sum_{j=0}^{v-1} s_2^j(n,m) \leq i < \sum_{j=0}^{v} s_2^j(n,m), \tag{7}$$

we know partially that the type $\mathbf{t}$ of the i-th derangements is $\mathbf{t}(2) = v$. We also know that the i-th derangement is the i'-th derangement with $\mathbf{t}(2) = v$, where

$$i' = i - \left(\sum_{j=0}^{v-1} s_2^j(n,m) \right). \tag{8}$$

We define $\mathbf{i}(2)$ of this derangement to be i'. The formal definition of $\mathbf{i}$ will be given in the following subsection.

Using the same idea, recursively, we can find $\mathbf{t}(3)$, $\mathbf{t}(4)$, and so on, together with corresponding indices $\mathbf{i}(3)$, $\mathbf{i}(4)$. This is the first phase, the decomposition phase, of our algorithm described in Subsection *"The Decomposition Phase"*.

The second phase, the reconstruction phase, described in *"The Reconstruction Phase"* actually reconstructs the i-th derangement from the type $\mathbf{t}$ and index $\mathbf{i}$.

Figure 9 illustrated the structure of the type decomposition. Where the top row is a stack of $s_k(n,m)$, and the bottom row is a zoom in of $s_k(n,m)$.

3.1 The Decomposition Phase

In this phase, we use i to find a type signature $\mathbf{t}$. We also decompose the given index i into a subindex $\mathbf{i}(k)$ for the corresponding $\mathbf{t}(k)$. To define $\mathbf{i}$ of a derangement formally, we first have to define how one would order derangements. The derangements are ordered, first, by the lexicographical ordering of their types. For the derangements of the same types, they are ordered lexicographically by the cycles of each length, with priority given to shorter cycles.

Intuitively, the index $\mathbf{i}(k)$ of a derangement π, is the index for choosing and arranging cycles of length k. Let X' be the set of items in π such that its cycles are of length at least k. If we construct a derangement by choosing cycles from the shorter ones, we may choose items from X' to form $\mathbf{t}(k)$ cycles of length k. Thus, $\mathbf{i}(k)$ is the index for the chosen cycles, each of length k, for π among all possible $\binom{|X'|}{k\mathbf{t}(k)} B(k\mathbf{t}(k), k)$ choices.

For the given n, m, and i, we would like to recursively find $\mathbf{t}$ for the corresponding i-th derangement. The recursive problem also include parameter r, the lower bound of the length of the smallest cycle; at the top-most call $r = 2$. More specifically, a function DECOMPOSETYPE takes n,m,r, and i, where i is the index of a derangement on n items with m cycles, with the restriction that the minimum length of each cycle is at least r. It returns the type $\mathbf{t}$ with subindices $\mathbf{i}$ of the corresponding derangement.

To do so, we first find (1) the minimum $k \geq r$ such that $\mathbf{t}(k) > 0$ and also (2) the value of $\mathbf{t}(k)$. To find k, we write $s_r(n, m)$ as

$$\begin{aligned} s_r(n,m) =& \big(s_r(n,m) - s_{r+1}(n,m)\big) \; + \; \big(s_{r+1}(n,m) - s_{r+2}(n,m)\big) \; + \; \cdots \; + \\ & \big(s_{n-1}(n,m) - s_n(n,m)\big). \end{aligned} \tag{9}$$

function FINDSMALLESTCYCLELENGTH(n, m, r, i)
  if $i < s_r(n, m)$ **then**
    return FINDSMALLESTCYCLELENGTH$(n, m, r{+}1, i)$
  return $(r{-}1, i{-}s_r(n, m))$
function FINDNUMSMALLESTCYCLES(n, m, k, i)
  for $v \in [1, 2, \cdots, \lfloor \frac{n}{k} \rfloor]$ **do**
    if $i < s_k^v(n, m)$ **then**
      return (v, i)
    $i \leftarrow i - s_k^v(n, m)$
function DECOMPOSETYPE(n, m, r, i)
  if $m = 0$ **then**
    return $\emptyset$
  $(k, i') \leftarrow$ FINDSMALLESTCYCLELENGTH(n, m, r, i)
  $(\mathbf{t}(k), i'') \leftarrow$ FINDNUMSMALLESTCYCLES(n, m, k, i')
  $(\mathbf{i}(k), j) \leftarrow$ DIVMOD$(i'', s_{k+1}(n{-}k\mathbf{t}(k), m{-}\mathbf{t}(k)))$
  $T \leftarrow (k, \mathbf{t}(k), \mathbf{i}(k))$
  return $[\ T\ ]$ + DECOMPOSETYPE$(n{-}k\mathbf{t}(k), m{-}\mathbf{t}(k), k{+}1, j)$

Figure 10 Type decomposition function

Each term $s_k(n, m) - s_{k+1}(n, m)$ counts the number of derangements whose minimum cycle length is exactly k. One can thus find k such that

$$s_r(n, m) - s_k(n, m) \leq i < s_r(n, m) - s_{k+1}(n, m). \tag{10}$$

Let $i' = i - (s_r(n, m) - s_k(n, m))$ be the ordering of the i-th derangement among the derangements whose type $\mathbf{t}'$ is such that $\mathbf{t}'(j) = 0$ for $j < k$ and $\mathbf{t}'(k) > 0$, i.e., the required i-th derangement is the i'-th derangement in these $s_k(n, m) - s_{k+1}(n, m)$ derangements. To find the number of cycles of length k in the i-th derangement, we turn to partial-type Stirling number of the first kind and note that

$$s_k(n, m) - s_{k+1}(n, m) = s_k^1(n, m) + s_k^2(n, m) + \cdots + s_k^m(n, m). \tag{11}$$

Therefor, the number of cycles of length k is the index ℓ such that

$$\sum_{j=1}^{\ell-1} s_k^j(n,m) \leq i' < \sum_{j=1}^{\ell} s_k^j(n,m). \tag{12}$$

We let $i'' = i' - \sum_{j=1}^{\ell-1} s_k^j(n,m)$.

From the above discussion, we see that to find the minimum cycle length k and the number of cycles of that particular length $\mathbf{t}(k)$ can be done simply by searching the values of s_k and s_k^ℓ. Figure 10 describes the above process, where the function FINDSMALLESTCYCLELENGTH searches for the index k and the offset i' and function FINDNUMSMALLESTCYCLES finds the value for $\mathbf{t}(k)$. Given k and $\mathbf{t}(k)$, during the reconstruction phase we would choose $k\mathbf{t}(k)$ items from the item set and create $\mathbf{t}(k)$ random cycles from them, and the rest of the items should belong to longer cycles. Note that for each choice for choosing these $\mathbf{t}(k)$ cycles, there are

$$s_{k+1}(n-k\mathbf{t}(k), m-\mathbf{t}(k)) \tag{13}$$

choices for the rest of the permutation. Therefore, to continue the process for longer cycles, we can write the index i'' as

$$i'' = \mathbf{i}(k) \cdot s_{k+1}(n - k\mathbf{t}(k), m - \mathbf{t}(k)) + j, \tag{14}$$

where $j < s_{k+1}(n-k\mathbf{t}(k), m-\mathbf{t}(k))$. The index $\mathbf{i}(k)$ is used to construct $\mathbf{t}(k)$ cycles and j is the ordering for the rest of the construction where we build a permutation containing longer cycles by recursively invoking DECOMPOSETYPE$(n-k\mathbf{t}(k), m-\mathbf{t}(k), k+1, j)$.

The next lemma considers the running time of DECOMPOSETYPE.

Lemma 13. DECOMPOSETYPE *in Figure 10 runs in* $O(n^3)$ *time.*

Proof. We first deal with the running time for computing all Stirling numbers and their extensions needed. First of all we can compute $s_r(\cdot,\cdot)$ for every r in time $O(n^3)$ since computing $s_r(n,m)$ for a particular r takes $O(n^2)$ time, as discussed in Section *"Preliminaries"*. Therefore we assume that we have all values of the r-associated Stirling number of the first kinds available when running the recursive procedure. From this, we have that functions FINDSMALLESTCYCLELENGTH runs in time $O(n)$.

After knowing k, the function FINDNUMSMALLESTCYCLES only needs $s_k^v(n,m)$ for every v, which can be computed in time $O(1)$ based on available $s_{k+1}(\cdot,\cdot)$. Thus FINDNUMSMALLESTCYCLES also works in $O(n)$ time.

Since we spend $O(n)$ time for each recursion level and there can be at most n levels of the recursion, we have that the running time for DECOMPOSETYPE is $O(n^2)$, without the preprocessing for $s_k(\cdot,\cdot)$. Combining with the preprocessing time, we have that the whole algorithm runs in $O(n^3)$ time. □

It is useful to have a sharper bound on the number of levels of recursion and consider the running time of DECOMPOSETYPE without the preprocessing needed for the Stirling numbers. The following Lemma will be very useful when deriving a faster algorithm.

Lemma 14. *Suppose that we can access to all* $s_r(\cdot,\cdot)$ *for every* r *in a constant time.* DECOMPOSETYPE *runs in time* $O(n^{1.5})$. *More specifically,* DECOMPOSETYPE *only makes at most* $O(\sqrt{n})$ *recursive calls.*

Proof. For the running time, we only need to bound the number of levels of recursion. Note that each recursive call involves a different value of cycle lowerbound r. Since the sum of cycle lengths in a permutation is at most n and they are all different, there can

be no more than $1 + \lceil 2\sqrt{n} \rceil$ cycle lengths since

$$\sum_{i=1}^{1+\lceil 2\sqrt{n} \rceil} i > n. \tag{15}$$

We conclude that the algorithm never makes more than $1 + \lceil 2\sqrt{n} \rceil = O(\sqrt{n})$ recursive calls. □

3.2 The Reconstruction Phase

In this phase, we take the output type signature $\mathbf{t}$ and subindices $\mathbf{i}$ from the previous phase to build up the final result derangement of desired number of cycles.

Based on the analysis of (2), each of $\mathbf{i}(k)$ corresponding to choosing $k\mathbf{t}(k)$ items from n_k items, the number of remaining items after done arranging all of cycles of length less than k. Then partition selected items and make many cycles of length k. Thus, we must break down $\mathbf{i}(k)$ furthermore, which are $\mathbf{i}'(k)$ index for choosing items; $\mathbf{i}''(k)$ index for partitioning into base cycles; and $\mathbf{i}'''_\ell(k)$ index for cyclic permutation on each cycle of $0 \leq \ell < \mathbf{t}(k)$. Function UNRANKDERANGEMENTCYCLES in Figure 11 describes the details of this assembling process. Also refer to Figure 12 for an example.

We have the following lemma.

Theorem 2. UNRANKDERANGEMENTCYCLES *in Figure 11 runs in time $O(n^3)$. Therefore, there exists an algorithm for uniformly generating a random derangement of n items with m cycles in $O(n^3)$ time.*

Proof. We first consider function UNRANKDERANGEMENTCYCLES. The function DECOMPOSETYPE runs in time $O(n^3)$ and returns a type signature $\mathbf{t}$ with at most $O(\sqrt{n})$ different cycle lengths. Each loop of the main function is dominated by the

function UNRANKDERANGEMENTCYCLES(X, m, i)
  $\pi \leftarrow \emptyset$
  for $(k, \mathbf{t}(k), \mathbf{i}(k)) \in$ DECOMPOSETYPE$(|X|, m, 1, i)$ **do**
    $(\mathbf{i}(k), \mathbf{i}'(k)) \leftarrow$ DIVMOD$(\mathbf{i}(k), \binom{|X|}{k\mathbf{t}(k)})$
    $X^* \leftarrow$ UNRANKCHOOSE$(X, k\mathbf{t}(k), \mathbf{i}'(k))$
    $(\mathbf{i}(k), \mathbf{i}''(k)) \leftarrow$ DIVMOD$(\mathbf{i}(k), B(k\mathbf{t}(k), k))$
    for $X^\dagger \in$ UNRANKPERIODICCHOOSE$(X^*, k, \mathbf{i}''(k))$ **do**
      $(\mathbf{i}(k), \mathbf{i}'''(k)) \leftarrow$ DIVMOD$(\mathbf{i}(k), (k-1)!)$
      $\pi \leftarrow \pi \cdot$ UNRANKCYCLICPERMUTE$(X^\dagger, \mathbf{i}'''(k))$
    $X \leftarrow X \setminus X^*$
  return π
function RANDOMDERANGEMENTCYCLES(n, m)
  $X \leftarrow [0, 1, 2, \cdots, n-1]$
  $i \leftarrow$ uniform random an integer such that $0 \le i < s_2(n, m)$.
  return UNRANKDERANGEMENTCYCLES(X, m, i)

Figure 11 Reconstruction of the derangement function

call to UNRANKPERIODICCHOOSE which runs in time $O(n^2/k) = O(n^2)$; thus given the type information, it takes $O(n^2\sqrt{n}) = O(n^{2.5})$ to reconstruct the derangement.

To randomly generate an instance of derangement, we consider function RANDOMDERANGEMENTCYCLES in Figure 11 that first chooses a random index i and then calls UNRANKDERANGEMENTCYCLES to find the i-th derangement. □

4. The $O(n^{2.5} \log n)$ Algorithm

In this section, we make an improvement on the algorithm shown in Section *"The $O(n^3)$ Algorithm"* to obtain an $O(n^{2.5} \log n)$-time algorithm.

Consider the search loop in FINDSMALLESTCYCLELENGTH. Since there are at most n possible values for k, we can use binary search to speed up the search to only $O(\log n)$ iterations. Not only that this implies a faster running time for this specific function, it implies that we only look at $O(\log n)$ values of k for $s_k(n, m)$. (Figure 13

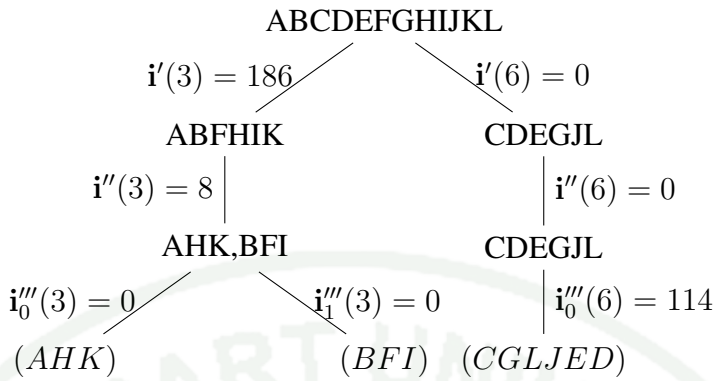


Figure 12 Example reconstruction given type signature and subindices

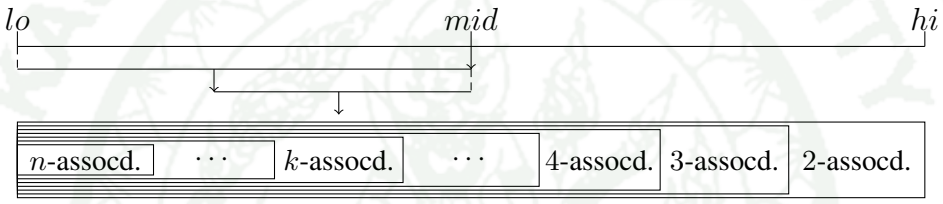


Figure 13 Binary search for k over the type structure

illustrates the idea.) Function FINDSMALLESTCYCLELENGTH in Figure 14 gives the implementation.

The same idea can be applied to function FINDNUMSMALLESTCYCLES to reduce the running time down to $O(\log n)$, provided that values of $s_k^v(\cdot,\cdot)$'s are available. This implies the total running time for a binary-search DECOMPOSETYPE of $O(\sqrt{n}\log n)$, without preprocessing time.

We account for the preprocessing time by noting that when considering the r-associated Stirling numbers of the first kind, we only look at $O(\log n)$ values of k's for each level, with the total of $O(\sqrt{n}\log n)$ values across all recursion levels. Computing those entries, only when needed, takes $O(n^2\sqrt{n}\log n)$ time as each $s_k(n,m)$ requires $O(n^2)$ time.

Since the function for the second phase only runs in time $O(n^{2.5})$, the total

```
function FindSmallestCycleLength(n, m, r, i)
    (lo, hi) ← (r, ⌊n/m⌋ + 1)
    while lo < hi do
        mid ← ⌊(lo+hi)/2⌋
        if i < s_mid(n, m) then
            lo ← mid + 1
        else
            hi ← mid
    k ← lo − 1
    return (k, i − s_{k+1}(n, m))
```

Figure 14 Improvement by binary search

running time for UnrankDerangementCycles is $O(n^{2.5} \log n)$ as stated below.

Theorem 3. *Given an index i of the derangement, there is an algorithm to find the i-th derangement with m cycles in time $O(n^{2.5} \log n)$. This algorithm can be used to uniformly generate a random derangement of n items with m cycles in the same running time.*

We can a have a simple extension where we want to find a derangement with n items with m cycles whose cycle lengths are lower bounded by r.

Corollary 1. *There exists an algorithm for generating a uniform permutation of n items with m cycles, where each cycle have length at least r, in $O(n^{2.5} \log n)$ time.*

5. Dealing with Large Numbers

Previously we work under the RAM model of computation where we can perform simple arithmetic operations in constant time. All running time analysis in previous sections make this assumption when dealing with large integers, especially with factorials.

This is unrealistic as the values of factorials can be very large. For a given n,

we note that $n! \leq n^n$; therefore, it requires an $O(n \log n)$-bit integer. To take this into account, we let $M(N)$ be the running time for an algorithm for multiplying N-bit integers; thus, our algorithm runs in time $O(n^{2.5} \log n \cdot M(n \log n))$ if all factorials are available. Computing all factorials takes only $O(n \cdot M(n \log n))$, which is dominated by the running time of the algorithm.

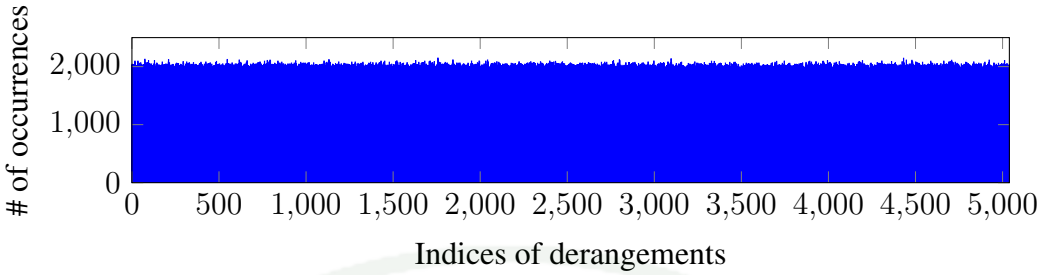
If one employs a straightforward multiplication algorithm, where $M(N) = N^2$, one would get an algorithm with running time $O(n^{4.5} \log^3 n)$, factor of $O(n^2 \log^2 n)$ slowdown in the running time. However, with a more efficient multiplication algorithm such as the one by Harvey and Van Der Hoeven (2021) where $M(N) = O(N \log N)$, one would obtain an $O(n^{3.5} \log^3 n)$-time algorithm, as claimed.

6. Experiments

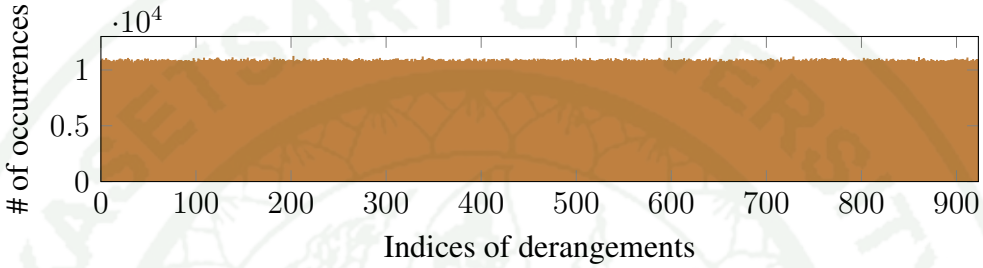
In this section, we present empirical data for the uniformity and the claimed running time of the proposed algorithm in Section *"The $O(n^{2.5} \log n)$ Algorithm"*. The laptop used for this task runs on Intel Core i5-10210U with 16GB RAM. The operating system is Ubuntu 22.04 LTS on Windows 10 WSL2. We implemented the algorithm in Python 3.10 since it uses big-integer as a default, playing a vital role for precise factorial computations. Python also comes with the standard library `random` for generating pseudo-random number, implementing MT19937 (Matsumoto and Nishimura, 1998) as a randomizer. The actual python implementation of the algorithm can be found at `https://github.com/neizod/derangement`.

6.1 Uniform generation

We first demonstrate that the generated derangements are uniformly distributed. Since the number of distinct permutations is very large (i.e., $n!$), we restrict the values of n and m to small numbers. In each case, we run the algorithm for 10^7 times to obtain the data.



(a) Distribution when $n = 8$ and $m = 1$



(b) Distribution when $n = 7$ and $m = 2$

Figure 15 Sampling distribution of generated derangements

For the first case, we deal with derangements with one cycle. We set $n = 8$ and $m = 1$, yielding $s_2(8, 1) = 5040$ distinct derangements. See the histogram in Figure 15a. We observe that the standard deviation of the number of occurrences is 44.752 which is reasonably close to the expected 44.539, calculated with an assumption that each derangement is generated independently.

The second case considers derangements with two cycles. We let $n = 7$ and $m = 2$. There are $s_2(7, 2) = 924$ distinct derangements in this case. See histogram in Figure 15b. We observe that the standard deviation of the number of occurrences is 103.182 (as compared to the expected 103.975, again calculated with an independence assumption).

For larger values of n, we investigate the distribution of $\pi(0)$ instead of the entire space of derangements. If the derangement algorithm works correctly, then $Pr[\pi(0) = u] = 1/(n-1)$ for every $u \neq 0$. We choose $n = 100$ and $m = 10$, then experiment 10^6 times and observe the distribution of $\pi(0)$. The histogram is shown in

Figure 16. We remark that we should see no generated derangements where $\pi(0) = 0$. The sampled standard deviation is 102.41, which is close to the expected 99.99 calculated under an independence assumption.

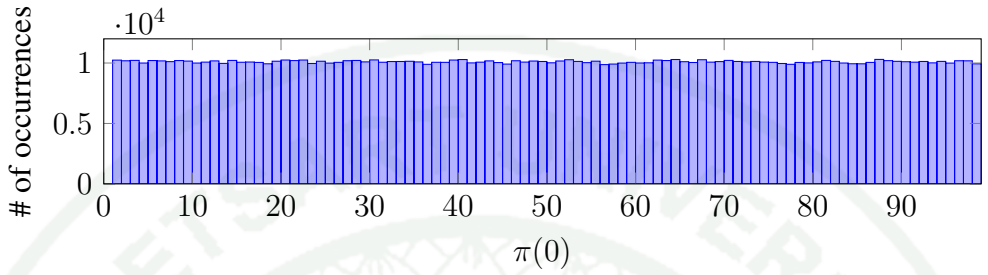


Figure 16 The distribution of $\pi(0)$ when $n = 100$ and $m = 10$

6.2 Running Time

The worst-case running time bound of $O(n^{3.5} \log^3 n)$ makes no assumption on m, the number of cycles. To see the effects of m to the running time, we perform an experiment where $n = 1000$ with many values of m. For each data point, we perform 10 runs. The result in Figure 17 demonstrates that m is an important factor in actual running time of the algorithm.

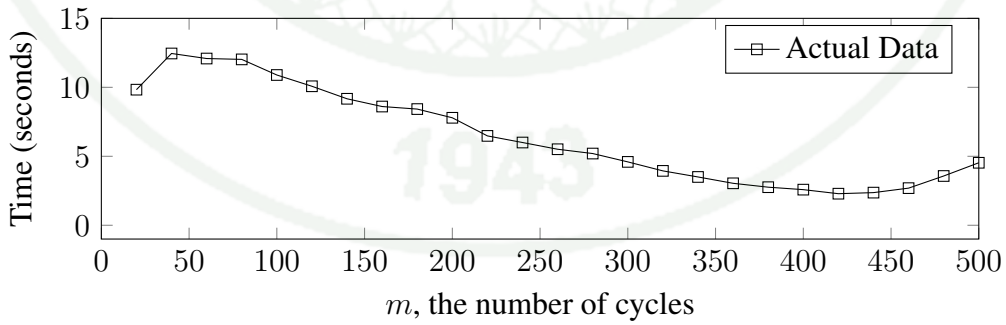
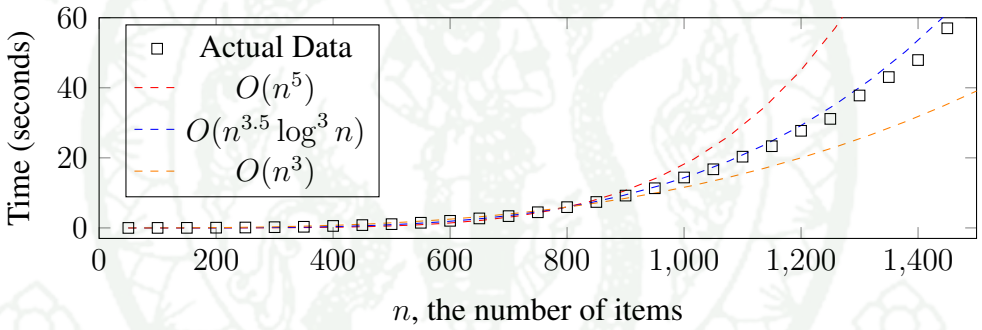


Figure 17 Effects of m to the actual running times

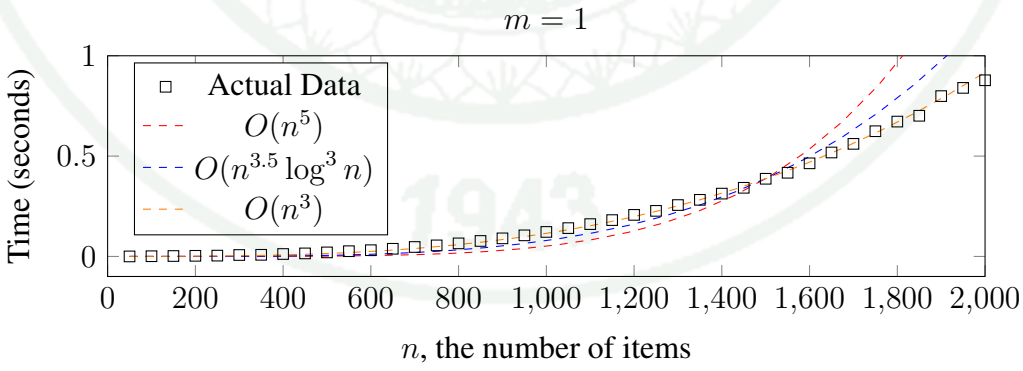
In Figure 17, we suspect that when m is small (m is around $5\%n$), the algorithm must perform many recursions, each allocate large table for dynamic pro-

gramming, resulted in a peak of the running time. When we increase m, each recursion break down the permutation into many small cycles of the same length; thus, the algorithm performs faster. However, when m is near the maximum (m is approximate $n/2$), the overhead of the algorithm shifted from computing dynamic programming table to generating of lots of cycles of the same length. Result in some slow down, but not slower than the peak when m is small.

To demonstrate the running time for various values of n from 50 to $1,450$, we let m be 5% of n. For each configuration, we perform 10 experiments and show the average running times in Figure 18a. For comparison, we draw curves of different asymptotic complexities (normalized to fix small actual values) as well.



(a) When $m = \lfloor 0.05n \rfloor$



(b) When $m = 1$

Figure 18 Sampling of running time

We also consider the case where $m = 1$ shown in Figure 18b. In this case, the experiment hints a faster running time, possibly $O(n^3)$. We suspect that this speed up might come from the fact we only need one table level, $s_2(n, 1)$, when $m = 1$. However, for the case when $m = 1$, one can use Sattolo's algorithm (Sattolo, 1986) which requires only linear time.

Publication 2

Phetmak, N. and J. Fakcharoenphol. 2023a. Folding every point on a polygon boundary to a point. **Algorithms**. 16 (6): 281.

algorithms

MDPI

Article

Folding Every Point on a Polygon Boundary to a Point

Nattawut Phetmak and Jittat Fakcharoenphol *

Faculty of Computer Engineering, Kasetsart University, Bangkok 10900, Thailand; nattawut.p@ku.th
* Correspondence: jtf@ku.ac.th

Abstract: We consider a problem in computational origami. Given a piece of paper as a convex polygon P and a point f located within, we fold every point on a boundary of P to f and compute a region that is safe from folding, i.e., the region with no creases. This problem is an extended version of a problem by Akitaya, Ballinger, Demaine, Hull, and Schmidt that only folds corners of the polygon. To find the region, we prove structural properties of intersections of parabola-bounded regions and use them to devise a linear-time algorithm. We also prove a structural result regarding the complexity of the safe region as a variable of the location of point f, i.e., the number of arcs of the safe region can be determined using the straight skeleton of the polygon P.

Keywords: geometric folding; parabola intersection; computational origami; linear-time algorithms; straight skeletons

Citation: Phetmak, N.; Fakcharoenphol, J. Folding Every Point on a Polygon Boundary to a Point. *Algorithms* **2023**, *16*, 281. https://doi.org/10.3390/a16060281

Academic Editors: Andreas Kanavos and Christos Makris

Received: 1 May 2023
Revised: 29 May 2023
Accepted: 29 May 2023
Published: 31 May 2023

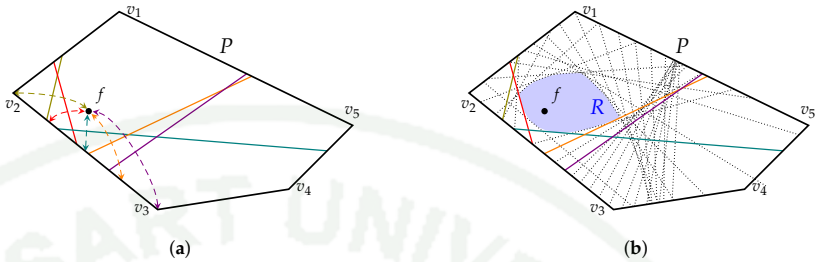
1. Introduction

Paper folding offers rich computational geometry problems with many real-world applications [1]. The topic, typically referred to as *computational origami* or *mathematics of paper folding* [2,3], studies both feasibility problems and also structural problems [4–6] with the aim to illuminate the connections between physical structures/problems and mathematical geometric objects (see, e.g., [7,8]). As geometric construction using straightedge and compass offers elegant connections between algebra and geometry, paper folding, which can be seen as geometric construction with additional operations, may provide beautiful structural properties worth studying (e.g., see [2,9]).

Akitaya, Ballinger, Demaine, Hull, and Schmidt [4] previously considered two folding problems on a convex piece of paper P. Given a query point p inside P, their first problem, originally proposed by Haga [10–12], called *points to a point*, is to find a region containing p bounded by creases after fold-and-unfold of each corner of P onto p. Their second problem, called *lines to a line*, takes as an input a line ℓ inside P and the goal is to find a region containing ℓ bounded by creases after a fold-and-unfold of each side of P onto ℓ. Although these two problems share a similar structure, they find that the outcomes diverge. That is, the region in the first problem resembles a Voronoi cell [13] with the inner point and corners as seeds, while the region in the second problem relies on the straight skeleton of the piece of paper.

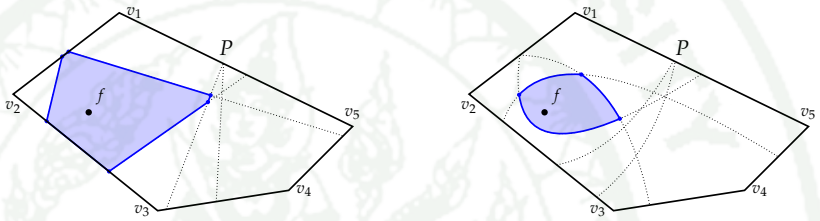
We consider a variant of this folding problem in the same flavor, i.e., we are given a query point f inside P and we are interested in a region containing f bounded by creases after a fold-and-unfold of *every* point on the boundary of the paper δP onto f. We call this result region a *safe region* R since every point $p \in R$ is safe from this folding procedure. Our contributions are an analysis of the shape of R, an efficient algorithm for finding R, and a complexity of the safe region with respect to any query point f.

Figure 1a shows a few creases after folding various points on polygon edge $\overline{v_2v_3}$ onto point f. The resulting safe region is shown in Figure 1b. Figure 2 provides a comparison between point-to-point folding considered in Akitaya et al. [4] (in Figure 2a) and our work (in Figure 2b).



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

Figure 1. The problem of folding every point on the boundary of a polygon P to a point f. (**a**) Sample creases from points on edge $\overline{v_2v_3}$; (**b**) The safe region R.



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

Figure 2. Comparison of the settings of the problem considered in this paper with that of [4]. (**a**) Corner folding in [4]; (**b**) Boundary folding in our paper.

We first show that, although we fold infinitely many points on δP, the result region R can be described finitely. It is a well-known result that a fold-and-unfold of multiple points from a line onto a point produces an envelope of a parabola. Thus, we consider each "side" of R as a parabolic arc instead of a traditional straight line segment. The analysis is presented in Section 2.

Using the analysis of the properties of safe regions, we present a linear-time algorithm for finding the region R, given a piece of paper P as a sorted list of n corners in counterclockwise order. Our algorithm works similarly to Graham's scan for a convex hull. That is, we consider adding one side of P as a parabola arc at a time, maintaining the loop invariant that regulates the region R. During each iteration, we may destroy some of the previously added parabola arcs. The key insight is that the destroyed parabola arc must be the one that is closest to the newly added parabolic region. Thus, it allows us to amortize the cost of destroying, achieving a linear-time algorithm overall. Section 3 explains the algorithm in full detail.

Finally, given any potential query point f, we calculate the precise number of parabola arcs of the region R. They turn out to be dependent on a set of inscribed circles, each of which is centered at a node of the straight skeleton of P. We dedicate Section 4 to focus solely on this property.

We hope that our problem will expand the richness of the family of fold-and-unfold origami problems. Nevertheless, it could serve as a *bridge* joining the previous two problems from [4], since our problem statement is similar to their first problem, but our result is similar to their second problem.

2. Preliminaries

In this section, we provide a formal definition of the problem. We are given a convex polygon P as an ordered list of vertices $V(P) = [v_1, v_2, \ldots, v_n]$ in counterclockwise order. We shall treat the list as a circular list. Given two points a and b, we refer to a line segment whose ends are a and b as a segment $\overline{ab}$. Equivalently, the polygon P is also represented as

a list of edges $E(P) = [e_1, e_2, \ldots, e_n]$, where $e_i = \overline{v_i v_{i+1}}$ is a line segment joining v_i and v_{i+1}. We denote its boundary as δP, i.e., $\delta P = \bigcup E(P)$.

We are also given a point f strictly inside P. Consider a point $u \in \delta P$ on edge $e_i = \overline{v_i v_{i+1}}$. Folding u onto f results in a straight line L, referred to as a *crease* line, which passes through the middle point on the line segment $\overline{uf}$ and is also orthogonal to $\overline{uf}$. We are interested in the set of crease lines when folding every point on e_i onto f. The envelope of this family of crease lines corresponds to a parabola, whose focus is f and directrix is a line resulting from extending the segment e_i to a line. We refer to this parabola as p_i. See Figure 3.

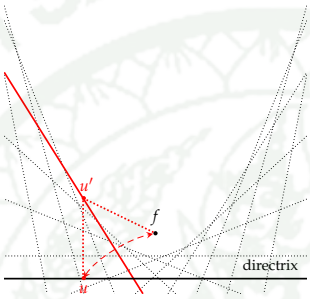


Figure 3. Fold-and-unfold points on a line to a fixed point multiple times, creating an envelope of a parabola.

To see this, let line L' be erected perpendicularly to the directrix at point u, and let $u' = L \cap L'$. We have $|\overline{u'f}| = |\overline{u'u}|$, which indeed obeys the definition of parabolas. In other words, u' is a point on a parabola's curve. Furthermore, for every point $v \in L'$ such that $|\overline{vf}| < |\overline{vu}|$, the point v is "safe" from other creases produced by fold-and-unfold of every other point from this directrix, which is a line extension of the segment e_i.

Since p_i divides the plane into two parabolic half-planes, we are interested in the region containing f. We define a half-space $H(p_i)$ to be the half-plane containing f. More formally, $H(p_i)$ contains all points v such that $|\overline{vf}| \leq |\overline{vu}|$, where u is an orthogonal projection of v on the line extension of e_i.

Our goal is to find:

$$R = \bigcap_{i=1}^{n} H(p_i),$$

defined to be the *safe region*. When focusing on point f, we sometimes refer to the safe region with respect to point f as R_f.

We describe the algorithm in Section 3. Later in this section, we state relevant geometry facts.

2.1. A Parabola as a Projection of a Conic Section

A parabola can be viewed as a conic section, i.e., a curve on a surface of a cone intersecting a cutting plane tilted at the same angle as the cone. Analytically, we consider a Euclidean space $\mathbb{R}^3$. A *cone* is a surface satisfying the equation:

$$x^2 + y^2 - z^2 = 0$$

with an apex of the cone at the origin. A *cutting plane* can be defined with an equation:

$$x \cos\theta + y \sin\theta - z = r,$$

where r is the distance on the xy-plane from the cone's apex to the nearest point of the cutting plane, and θ is the *directional angle* on the xy-plane to that point. By this definition, we have a parabola as a curve on the tilted cutting plane. See Figure 4a,b.

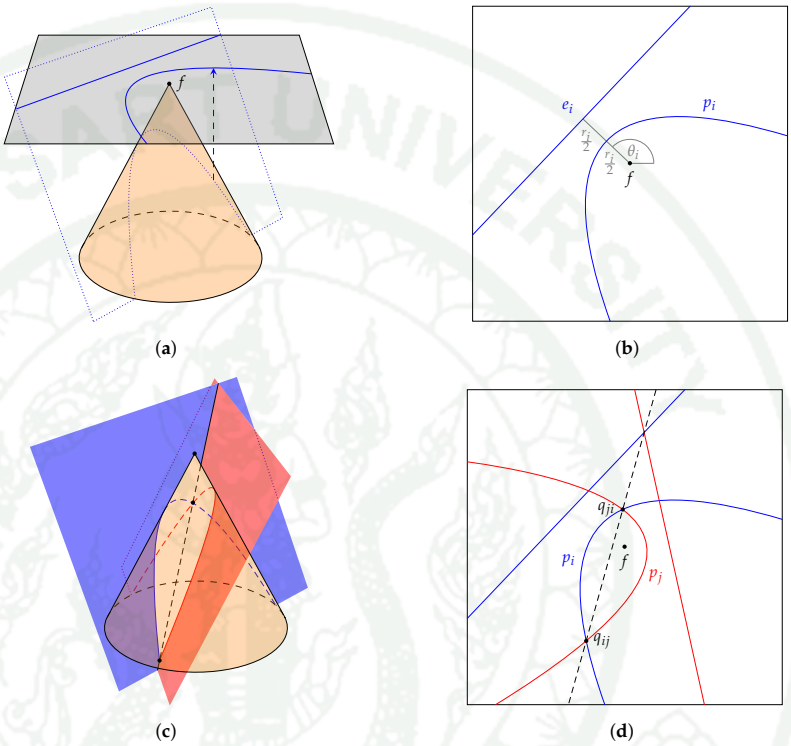


Figure 4. Interpretation of a parabola as a projection of a conic section. (**a**) Mapping a tilted parabola to the xy-plane; (**b**) Projection of a parabola on the xy-plane; (**c**) Two cutting planes; (**d**) Angle bisector and two intersections.

A projected parabola is an orthogonal projection of the tilted parabola onto the xy-plane, which is also a parabola. The projected parabola has the cone's apex as its focus, and an intersect line of the cutting plane with the xy-plane as its directrix. This projection viewpoint was briefly mentioned at the end of [14]. In this paper, we refer to projected parabolas simply as parabolas.

We say that parabola p is in the *upright form* if, by rotating and translating the xy-plane, the parabola possesses an analytical form of $y = tx^2$ where $t > 0$.

2.2. Two Parabolas

We define *semi-confocal parabolas* as a family of parabolas that share the same focus, but their directrixes do not need to have the same directional angle. It follows that semi-confocal parabolas are projected parabolas from multiple cutting planes that cut the same cone. We state two important facts on intersections of two parabolas which, under the conic interpretation, are straightforward.

Lemma 1. *Two semi-confocal parabolas do not intersect iff their directional angles are the same.*

Proof. Two parabolas with the same directional angle are produced from two parallel cutting planes in the conic view, which never intersect. □

Lemma 2. *If two semi-confocal parabolas intersect, then they intersect at two points which also lie on an angle bisector of their directrixes.*

Proof. Take conic section interpretation. Their curves on the surface of the cone must also lie on their cutting planes, in which the intersection of the planes is a straight line. This line piece goes through the cone exactly two times. See Figure 4c,d. □

It follows from Lemma 2 that two intersecting semi-confocal parabolas p_i, p_j produce a safe region $R = H(p_i) \cap H(p_j)$ which is a bounded convex region. Moreover, since we only deal with parabolas whose directrixes are from boundary edges of a convex polygon, the non-intersecting case never occurs.

Given two parabolas p_i and p_j, we can compute their intersections using analytical techniques in $O(1)$ time as follows. We reduce the problem of finding intersection of two parabolas to the problem of finding intersection of a parabola and a line. Without loss of generality, we consider p_i in the upright form. From Lemma 2, we find an angle bisector b such that it divides the inner angle between edges e_i and e_j. Then, we find the resulting intersections of p_i and b using quadratic equations.

We remark particularly on the structure of the safe region R.

Consider each parabola p_i in the upright form. We can partition this parabola using the two intersection points into three arcs: the *left arc*, the *central arc*, and the *right arc*, where the left arc corresponds to the half-parabola unbounded to $-\infty$ and the right arc corresponds to the half-parabola unbounded to $+\infty$, and the central arc lies between the two intersection points (see Figure 5). Under this notation, we note that the left arc of p_i intersects p_j only once at the intersection point where it also intersects the right arc of p_j, and vice versa.

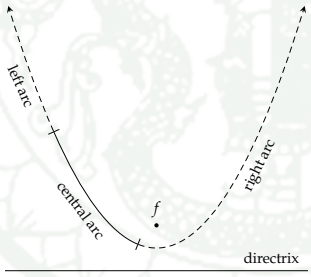


Figure 5. Arc decomposition of a parabola.

In our analysis, where there are parabolas $p_1, p_2, \ldots, p_n$, we refer to the two intersection points between parabolas p_i and p_j as q_{ij} and q_{ji}. To distinguish between these two points, imagine if one traverses counterclockwisely on the boundary of $H(p_i) \cap H(p_j)$, then one would see an arc of p_i, the intersecting point q_{ij}, the arc of p_j, and then the intersection point q_{ji}. The counterclockwise definitions of these points are crucial to our proof of Lemma 4. See Figure 4d.

2.3. Many Parabolas

In this section, we analyze the structure of the intersection of k semi-confocal parabolas, extending the result from the previous section.

Let $p_1, p_2, \ldots, p_k$ be k semi-confocal parabolas with different directional angles. We shall consider the safe region of these parabolas and prove the following lemma.

Lemma 3. *Each parabola touches at most one arc of the safe region.*

Proof. We prove by induction on k for the case where $k = 2$ follows from Lemma 2. Consider k parabolas. Let $R' = \bigcap_{i=1}^{k-1} H(p_i)$. Inductively, each parabola $p_1, p_2, \ldots, p_{k-1}$ touches at most one arc of R'. We consider $R = R' \cap H(p_k)$, we shall show that p_k only touches at most one arc of R. If $R' \subseteq H(p_k)$, then $R = R'$ and p_k does not touch R; hence, the lemma is true. We then assume that $R' \nsubseteq H(p_k)$.

Clearly, p_k touches one arc of $R = R' \cap H(p_k)$. We call that arc a_k. Each endpoint of a_k belongs to some arc of R'.

There are two cases.

Case 1: Both endpoints of a_k belongs to a single arc of p_i. In this case, p_k intersects exactly one arc of R' exactly twice. Then, the safe region R is the intersection of exactly two parabolas, i.e., $R = H(p_i) \cap H(p_k)$. Thus, the lemma follows from Lemma 2.

Case 2: One endpoint of a_k belongs to p_i while the other belongs to p_j. We show that in this case, p_k intersects exactly two arcs, implying the lemma. We consider p_i first, i.e., we look at the intersection $R_i = H(p_k) \cap H(p_i)$. Let q_i be the intersecting point of p_i and p_k. We can partition p_k into three arcs; let b_k be the unbounded arc starting at q_i. From Lemma 2, we know that b_k only intersects R_i once. Since $R \subseteq R_i$, we have that b_k also intersects R at most once at q_i. We follow the same argument for p_j. Thus, p_k only intersects R' exactly twice, as claimed. □

3. The Algorithm

Our algorithm for finding a safe region works similarly to Graham's scan [15] for convex hull. We briefly described the algorithm as a pseudocode in Algorithm 1. Later in this section, we explain the algorithm and prove its correctness.

Algorithm 1: Our algorithm for finding the safe region.

```
function SafeRegion(P : Polygon, f : Point)
    [e_1, e_2, e_3, ..., e_n] ← E(P)
    p_1 ← Parabola(f, e_1)
    p_2 ← Parabola(f, e_2)
    q_12 ← InnerIntersection(p_1, p_2)
    q_21 ← InnerIntersection(p_2, p_1)
    R ← DoublyLinkedList(Arc(p_1, q_21, q_12), Arc(p_2, q_12, q_21))
    for i ∈ [3, 4, 5, ..., n] do
        p_i ← Parabola(f, e_i)
        (p_H, ℓ_H, r_H) ← Head(R)
        (p_T, ℓ_T, r_T) ← Tail(R)
        q_Ti ← InnerIntersection(p_T, p_i)
        q_iH ← InnerIntersection(p_i, p_H)
        if q_iH ∈ LeftArc(p_H, ℓ_H) ∧ q_Ti ∈ RightArc(p_T, r_T) then
            continue                                  // skip this iteration
        while q_iH ∈ RightArc(p_H, r_H) do
            R ← RemoveHead(R)
            (p_H, _, r_H) ← Head(R)
            q_iH ← InnerIntersection(p_i, p_H)
        while q_Ti ∈ LeftArc(p_T, ℓ_T) do
            R ← RemoveTail(R)
            (p_T, ℓ_T, _) ← Tail(R)
            q_Ti ← InnerIntersection(p_T, p_i)
        R ← UpdateHead(R, Arc(p_H, q_iH, r_H))
        R ← UpdateTail(R, Arc(p_T, ℓ_T, q_Ti))
        R ← AppendTail(R, Arc(p_i, q_Ti, q_iH))
    return R
```

Algorithm 1 uses the following subroutines. Subroutine PARABOLA(f, e) creates a representation of a parabola with f as its focus and a line extension of edge e as its directrix. Subroutine INNERINTERSECTION(p_i, p_j) computes the intersection point q_{ij} of parabola p_i and p_j; we recall that from Lemma 2, there are two intersection points and this subroutine returns the "inner" one, i.e., q_{ij}, not q_{ji}.

Since our goal is to find a safe region,

$$R = \bigcap_{i=1}^{n} H(p_i),$$

Algorithm 1 iterates over parabolas p_i producing a partial solution R_i such that:

$$R_i = \bigcap_{j=1}^{i} H(p_j),$$

i.e., R_i is the safe region for the first i parabolas. We maintain R_i as a cyclic list of parabola arcs:

$$a_1, a_2, \ldots, a_k,$$

where each arc a_j is a 3-tuple (p, ℓ, r) which keeps a reference to the parabola $a_j.p$, its left endpoint $a_j.\ell$, and its right endpoint $a_j.r$. We note that with this representation, $a_j.r = a_{j+1}.\ell$ for $1 \leq j < k$, and $a_k.r = a_1.\ell$. We also note that, using the notation defined in Section 2.2, $a_j.r$ is $q_{j(j+1)}$ and $a_j.\ell$ is $q_{(j-1)j}$.

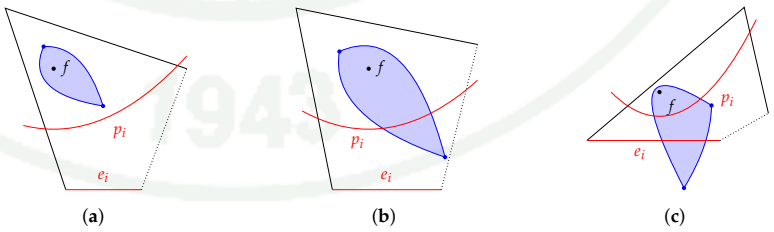
Initially, we start with $R_2 = H(p_1) \cap H(p_2)$. We encode the partial safe region as an ordered list of arcs, $A(R_2) = [a_1, a_2]$, where:

$$a_1 = (p_1, q_{21}, q_{12})$$
$$a_2 = (p_2, q_{12}, q_{21})$$

For each iteration $i > 2$, we consider adding $H(p_i)$ to R_{i-1} to produce $R_i = R_{i-1} \cap H(p_i)$. There are three cases:

- *Case 1*: p_i does not change the region, i.e., $R_{i-1} \cap H(p_i) = R_{i-1}$ and we can discard p_i;
- *Case 2*: p_i clips the region, i.e., all parabola arcs in R_{i-1} remain on the boundary of $R_{i-1} \cap H(p_i)$; or
- *Case 3*: p_i *eclipses* other parabolas in the region, i.e., some parabola arc is entirely outside $R_{i-1} \cap H(p_i)$.

Figure 6 illustrates these three cases. The safe regions R_{i-1} are shown with additional parabolas p_i with e_i as their directrixes.



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

Figure 6. Three cases of adding parabola p_i to the partial safe region R_{i-1}. (**a**) *Case 1*: p_i does not change the region. (**b**) *Case 2*: p_i clips the region. (**c**) *Case 3*: p_i eclipses some parabola.

To distinguish between these cases, our basic procedure is to test if a point lies in $H(p_i)$. The counterclockwise ordering of parabolas ensures that p_i would affect two sequences of arcs, i.e., clockwisely,

$$a_k, a_{k-1}, a_{k-2}, \ldots,$$

to be referred to as the *neighbors to the left* of p_i, and counterclockwisely,

$$a_1, a_2, a_3, \ldots,$$

to be referred to as the *neighbors to the right* of p_i,

We first consider point $q_{k1} = a_k.r$ (which is also $a_1.\ell$). Lemma 4 below ensures that we are in Case 1 if $q_{k1} \in H(p_i)$. See Figure 7.

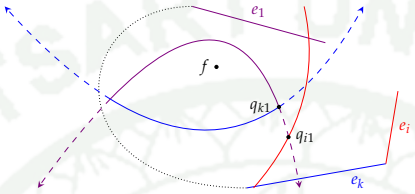


Figure 7. Proof of Lemma 4.

Otherwise, some part of R_{i-1} is below parabola p_i. We in turns consider the points:

$$a_{k-1}.r, a_{k-2}.r, \ldots,$$

from the neighbors to the left of p_i and find the largest index j such that $a_j \cap H(p_i) \neq \emptyset$. In this case, the parabolas $a_k.p, a_{k-1}.p, \ldots, a_{j+1}.p$ are eclipsed by p_i.

We also process the neighbors to the right of p_i similarly by finding the smallest index j' such that $a_{j'} \cap H(p_i) \neq \emptyset$ together with the sequence of eclipsed arcs $a_1, a_2, \ldots, a_{j'-1}$. We note that it can be the case that $j = j'$ when only one arc survives p_i eclipsing.

To construct R_i, we discard eclipsed arcs, add a new arc a_i for p_i, and compute the following:

- The left intersection point $a_i.\ell = a_j.r$, which is the intersection between p_i and $a_j.p$;
- The right intersection point $a_i.r = a_{j'}.\ell$, which is the intersection between p_i and $a_{j'}.p$.

Finally, we re-index the arcs in R_i. We quickly remark that this procedure can be seen as a "twin-headed" Graham scan.

The following lemmas show that this procedure is correct.

Lemma 4. *If $q_{k1} \in H(p_i)$, then $R_{i-1} \subseteq H(p_i)$ and $R_i = R_{i-1}$.*

Proof. Since $R_{i-1} \subseteq H(p_1) \cap H(p_k)$, our goal is to show that $H(p_1) \cap H(p_k) \subseteq H(p_i)$ in this case.

Consider the intersection of p_1 and p_k. Recall that the two intersection points q_{k1} and q_{1k} partition both parabolas into their left arcs, central arcs, and right arcs. Let us call them a_k^ℓ, a_k^c, a_k^r and a_1^ℓ, a_1^c, a_1^r.

We now consider the intersection of p_1 and p_i. We show that q_{i1}, the intersection point of p_i and p_1, is on the left arc a_1^ℓ of p_1. To see this, we start by rotating the plane such that p_i is in the upright form. Then, we find a region $r = H(p_1) \cap H(p_i)$. Since $q_{k1} \in p_1$ and also $q_{k1} \in H(p_i)$, q_{k1} in on the boundary of $H(p_1) \cap H(p_i) = r$. Again, since $q_{k1} \in p_1$ and also on the boundary of r, traversing from q_{k1} on the boundary of r clockwisely with respect to f would reach q_{i1}, by definition of q_{i1}, as claimed.

Using the same argument, we can show that q_{ki} is in the right arc a_k^r of p_k.

Using q_{i1} and q_{ki}, we partition p_i into three arcs—a_1, a_2, and a_3—so that a_1 is an unbounded curve with q_{i1} as its end point, a_2 is a bounded part with q_{i1} and q_{ki} as their end points, and finally, a_3 is an unbounded curve with q_{ki} as its end point (see Figure 7).

Using the structure from Lemma 2, we know that $a_1 \cup a_2$ intersects p_k at only $q_{ki} \in a_k^r$. Thus, p_i does not intersect $a_k^c \cup a_k^\ell$.

Additionally, we know that $a_2 \cup a_3$ intersects p_1 at only $q_{i1} \in a_1^\ell$, implying that p_i does not intersect $a_1^r \cup a_1^c$.

We can conclude that p_i does not intersect with R_{i-1} because R_{i-1} lies between the unbounded curves $a_k^c \cup a_k^\ell$ and $a_1^r \cup a_1^c$. □

We also have a simple contraposition.

Corollary 1. *If $R_{i-1} \not\subseteq H(p_i)$, then $q_{k1} \notin H(p_i)$.*

Let $\hat{R} = \bigcap_{j=1}^{i} H(p_j)$ be the correctly updated solution, we would like to show that R_i constructed above equals $\hat{R}$. We remark that the i-th parabola p_i corresponds to edge e_i that comes counterclockwisely after all other edges that contribute to R_{i-1}.

Lemma 5. *If $R_{i-1} \not\subseteq H(p_i)$, the arcs on R_{i-1}'s boundary which do not belong to $\hat{R}$ form a consecutive sequence:*

$$a_j, a_{j+1}, \ldots, a_k, a_1, a_2, \ldots, a_{j'}.$$

Proof. Lemma 3 ensures that if p_i intersects the boundary of R_{i-1}, p_i touches at most one arc of $\hat{R}$, the result of the intersection of $H(p_i)$ and R_{i-1}. This implies that the arcs of R_{i-1} do not belong to form a (circular) consecutive sequence. To see this, assume otherwise and note that in that case, p_i would touch more than one arc of $\hat{R}$.

Our procedure finds the consecutive sequence starting at q_{k1}, the intersection of p_k and p_1, then iterates through other consecutive points. Thus, the procedure is correct if the starting point is correct, i.e., we start at some intersection point outside $\hat{R}$. This is indeed the case because Corollary 1 guarantees that when $R_{i-1} \not\subseteq H(p_i)$, q_{k1} is outside $\hat{R}$. □

We conclude with our main correctness theorem.

Theorem 1. *Our updating procedure is correct, i.e., $R_{i+1} = \hat{R}$, and the algorithm computes the safe region in linear time.*

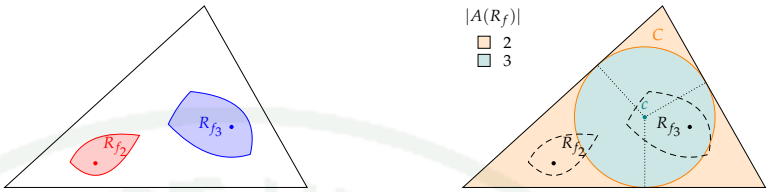
Proof. Regarding the updating procedure, we deal with three possible cases. Lemma 4 ensures that our condition for Case 1 is correct. In other cases, Lemma 5 shows that the procedure for deleting arcs is correct. By induction on n, the algorithm thus produces the required safe region.

To analyze the running time, we first note that, except the two inner while loops, for each i, the algorithm runs in $O(1)$ time. To account for the running time of the inner while loops, observe that each iteration of the loop removes one parabola from the list. Since at most n parabolas are inserted in the list, the deletion can take place at most n times, implying the total running time of $O(n)$ for the loops. □

4. The Number of Arcs of the Safe Region

In this section, we consider the complexity of the boundary of the safe region; in other words, we count the exact number of arcs of the safe region. From previous sections, we derive that a side of the safe region is a parabolic arc with point f as its focus. We also see that some edge of P may not contribute to the resulting safe region, i.e., a parabola associated with it does not touch the safe region. It is natural to ask for the number of arcs of the safe region.

Assuming that the polygon P is fixed, the number of arcs of the safe region depends on the focus f. We denote explicitly by R_f a safe region with point f as its focus. As in [4], this section analyzes the number of arcs of safe region R_f, i.e., $|A(R_f)|$, where $A(R_f)$ is the set of arcs of R_f. Figure 8a shows two safe regions R_{f_2} with query point f_2 and R_{f_3} with query point f_3.



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

Figure 8. Safe regions with 2 and 3 parabola arcs. (**a**) Safe regions with query points f_2 and f_3; (**b**) An event circle determines the numbers of arcs.

Akitaya et al. [4] consider the same problem for the case where each side of P is folded onto a line. They show that the straight skeleton of P plays an important role in determining the number of sides of the resulting region. This is true for our case as well. As an example, Figure 8b shows an inscribed circle C which can be determined using the straight skeleton and two safe regions shown previously. We remark that $f_2 \notin C$ but $f_3 \in C$.

We start by defining useful notations related to straight skeletons and event circles. A *straight skeleton* [16] of a polygon P, denoted by $S(P)$, is a subset of P such that for each point $u \in S(P)$, there exist at least two points on δP with the same distance to u. More intuitively, we may see the skeleton as a Voronoi diagram of line segments where each site is an edge of the polygon. The straight skeleton $S(P)$ partitions P into regions, referred to as *faces*. Thus, under the Voronoi interpretation, each face is bounded by exactly one polygon edge as other edges of $S(P)$. We refer to a face that is bounded by polygon edge e_i as face F_i. We also note that a face is also a convex polygon. See Figure 9 for an illustration.

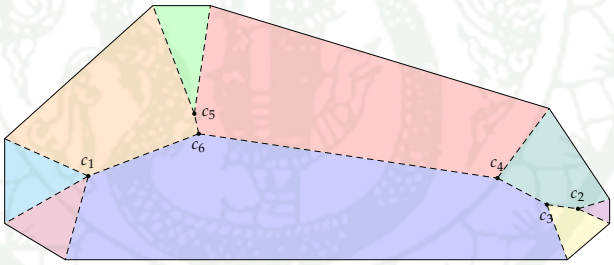


Figure 9. The straight skeleton of polygon P, with each face colored differently. Points $c_1, \ldots, c_6$ are event points.

The skeleton may be viewed as a tree, where each non-leaf node ensures at least three equidistant points on δP. A non-leaf node of $S(P)$ is referred to as an *event point*. A circle centered at an event point and tangential to the nearest edge of the polygon is called an *event circle* of $S(P)$. Let $\mathcal{C}$ be the set of all event circles of $S(P)$, and let $\mathcal{C}_e \subseteq \mathcal{C}$ be a set of event circles tangential to edge e. We also denote by $\text{int}(C)$ an interior of circle C, i.e., the set $\{(x, y) : (x - x_0)^2 + (y - y_0)^2 < r^2\}$ for a circle with center (x_0, y_0) and radius r.

The goal of this section is to show that, under the fixed polygon P, the structure of $A(R_f)$ is governed by event circles of straight skeleton $S(P)$ of P.

We start by analyzing the case when the safe region intersects with the skeleton faces. The following lemma directly follows from the Voronoi interpretation of the straight skeleton.

Lemma 6. *For each event circle C with event point c, c is adjacent to face F_i if and only if its corresponding polygon edge e_i is tangential to C.*

The following two lemmas provide basic properties for our main theorem in this section.

Lemma 7. *For a particular focus f, $\text{int}(R_f)$ intersects with skeleton face F_i adjacent to edge e_i iff the associated parabola p_i is part of the arcs of R_f.*

Proof. ($\Rightarrow$) Assume that $\text{int}(R_f)$ intersects F_i. Since F_i is bounded by a polygon edge and R_f is contained in P, we know that there are parts of the boundary of R_f that intersect F_i. Consider any point u on the boundary of R_f inside F_i. Clearly, u must be on an arc of some parabola, i.e., we have that:

$$\min_j |\overline{uu_j}| = |\overline{uf}|,$$

where u_j is an orthogonal projection of u onto e_j. Since all points in F_i are closer to e_i than other edges, the minimizer of the above term is u_i; thus, u must also lie on p_i, i.e., p_i is part of arcs of R_f.

($\Leftarrow$) We prove by contradiction. Assume that $\text{int}(R_f)$ does not intersect F_i, but p_i is part of the boundary of R_f. Consider any point $u \in p_i$ on the boundary. Since $\text{int}(R_f)$ is disjoint from F_i, u is strictly in some face F_j. In this case, we have that:

$$|\overline{uf}| = |\overline{uu_i}| > |\overline{uu_j}|,$$

where u_i and u_j are orthogonal projections of u onto e_i and e_j, implying that $u \notin H(p_j)$, a contradiction. □

Lemma 8. *For $C \in \mathcal{C}$ with event point c, a safe region R_f strictly contains an event point c, i.e., $c \in \text{int}(R_f)$, iff $f \in \text{int}(C)$.*

Proof. Let r be the radius of an event circle C. Project c orthogonally to a line extension of every edge e_i, named the projected point c_i.

($\Leftarrow$) Assume that $f \in \text{int}(C)$, i.e., $|\overline{cf}| < r$. We show that $r \in H(p_i)$ for every parabola p_i associated with polygon edge e_i. This is the case when r is strictly closer to f than every other edge e_i. Consider each edge e_i tangent to C, we have $|\overline{cf}| < r = |\overline{cc_i}|$. For edge e_i not tangent to C, we have that $|\overline{cc_i}| > r$; thus, $|\overline{cf}| < r < |\overline{cc_i}|$. Hence, c is in the safe region R_f.

($\Rightarrow$) Assume that $f \notin \text{int}(C)$. In this case, we have $|\overline{cf}| \geq r$. Since C is an event circle, there exists edge e_i tangent to C. For that particular edge, we have $|\overline{cc_i}| = r$. Thus, $|\overline{cf}| \geq r = |\overline{cc_i}|$, and c is not strictly contained in the safe region R_f. □

The following theorem gives the number of boundary arcs of R_f as a function of event circles containing f.

Theorem 2. *If f is strictly inside some event circle, i.e., $f \in \text{int}(C)$ for some $C \in \mathcal{C}$, then:*

$$|A(R_f)| = |\{e_i \in E(P) : \textit{there exists } C \in \mathcal{C}_{e_i} \textit{ s.t. } f \in \text{int}(C)\}|$$

Otherwise, $|A(R_f)| = 2$.

Proof. We first assume that $f \in \text{int}(C)$ for some event circle C. Consider each event circle C with event point c such that $f \in \text{int}(C)$. From Lemma 8, we know that $c \in \text{int}(R_f)$. This also means that $\text{int}(R_f)$ intersects every face F_i adjacent to event point c. Lemma 6 ensures that these faces F_i's correspond with edges e_i's tangent to C, the set of edges e_i such that $C \in \mathcal{C}_{e_i}$. Since Lemma 7 ensures that for each face F_i adjacent to edge e_i intersecting with R_f, the parabola p_i appears as an arc of R_f, we have that for each tangent edge e_i of C, its parabola p_i appears as an arc in R_f. The lemma, in this case, follows by taking the union of all boundary edges from every event circle $C \in \mathcal{C}$ that f is strictly inside.

On the other hand, if f is not strictly contained in any event circle $C \in \mathcal{C}$, Lemma 2 ensures that the safe region must touch two parabolas. □

Figure 10 shows an application of Theorem 2. The event circles are shown with their intersections. From Theorem 2, for each point f, the number of arcs of $A(R_f)$ depends

on the number of edges tangent to event circles containing f. Thus, every point f in a particular intersection of event circles has the same value of $|A(R_f)|$. Each intersection in Figure 10 is shown with a different color depending on the value of $|A(R_f)|$ for a query point f inside it. Remark again that for query point f in the area outside any event circles, the value of $|A(R_f)|$ is 2.

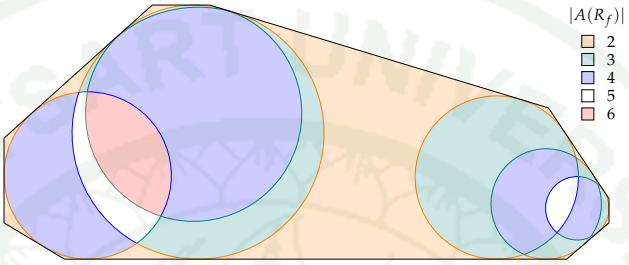


Figure 10. Intersections of event circles from the straight skeleton determine $|A(R_f)|$.

Alternatively, one may view Theorem 2 with the conic section interpretation as follows. The input polygon P induces n cutting planes, forming the straight skeleton and their corresponding faces when projected onto the xy-plane, and the point f is represented as a cone whose apex is at f.

The structural results in this section give another linear-time algorithm for finding safe regions, by first finding straight skeleton in $O(n)$-time using [17], then computing event circles, and finally using this information to find the set of edges contributing to the arcs of the safe region. However, we believe that the results in this section contribute mainly to the structural understanding of the problem and may serve as a guideline for tackling harder problems, especially the non-convex case of the problem. We discuss this in Section 5.1.

5. Conclusions and Open Problems

We present a linear-time algorithm for finding a safe region for folding each point on the boundary of a convex polygon P to point $f \in P$. We also give structural properties related to the number of arcs in a safe region for each focal point f, based on straight skeletons. We note the crucial roles of straight skeletons in our problem as well as other problems in origami design, as can be seen in [18]. An interesting direction for future work is to investigate problems with the similar structures while using straight skeletons as keys.

As mentioned in the introduction, we also hope that our results show interesting connections between the two problems posted by Akitaya et al. [4].

5.1. Remarks on Non-Convex Polygons

Results in Section 4 shed some light on non-convex cases. However, there are issues with the current approach. When dealing with non-convex polygons, there are two related concepts: straight skeletons [16] and medial axes [19] (see also [17,20]). For a given polygon, a medial axis contains points with equal distance to more than one points on δP, while a straight skeleton is a Voronoi diagram where each site is a *line extension* of each edge. They are the same in convex polygons, however, in non-convex polygons, their medial axes contain curved segments.

In our application, since we can only fold every point on each polygon edge, but not points on the line extension of the edge, it makes sense to consider a medial axis. However, each face in the medial axis can be bounded by more than one polygon edges, breaking down one of our key assumptions. We leave the investigation of this approach to non-convex polygons as an important open question.

Author Contributions: Conceptualization, N.P. and J.F.; methodology, N.P. and J.F.; formal analysis, N.P. and J.F.; writing—original draft preparation, N.P.; writing—review and editing, J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Thailand Research Fund grant number RSA-6180074.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors would like to thank organizers and participants of JCDCGGG 2022, especially Hugo Akitaya, for giving feedback on the presentation of this work and pointing out the important aspect. We also thank anonymous reviewers for insightful feedbacks.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lang, R.J. Computational Origami: From Flapping Birds to Space Telescopes. In Proceedings of the Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry (SCG '09), Aarhus, Denmark, 8–10 June 2009; Association for Computing Machinery: New York, NY, USA, 2009; p. 159–162. [CrossRef]
2. Demaine, E.D.; O'Rourke, J. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*; reprint ed.; Cambridge University Press: New York, NY, USA, 2008.
3. Hull, T.C. *Origametry: Mathematical Methods in Paper Folding*; Cambridge University Press: Cambridge, UK, 2020.
4. Akitaya, H.A.; Ballinger, B.; Demaine, E.D.; Hull, T.C.; Schmidt, C. Folding Points to a Point and Lines to a Line. In Proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG 2021), Halifax, NS, Canada, 10–12 August 2021; Dalhousie University: Halifax, NS, Canada, 2021; pp. 271–278.
5. Akitaya, H.A.; Demaine, E.D.; Ku, J.S. Simple Folding is Really Hard. *J. Inf. Process.* **2017**, *25*, 580–589. [CrossRef]
6. Abel, Z.; Demaine, E.D.; Demaine, M.L.; Eppstein, D.; Lubiw, A.; Uehara, R. Flat foldings of plane graphs with prescribed angles and edge lengths. *J. Comput. Geom.* **2018**, *9*, 74–93. [CrossRef]
7. Dambrogio, J.; Ghassaei, A.; Smith, D.; Jackson, H.; Demaine, M.; Davis, G.; Mills, D.; Ahrendt, R.; Akkerman, N.; van der Linden, D.; et al. Unlocking history through automated virtual unfolding of sealed documents imaged by X-ray microtomography. *Nat. Commun.* **2021**, *12*, 1184. [CrossRef] [PubMed]
8. Felton, S.; Tolley, M.; Demaine, E.; Rus, D.; Wood, R. A method for building self-folding machines. *Science* **2014**, *345*, 644–646. [CrossRef] [PubMed]
9. Hull, T.C. Solving cubics with creases: The work of Beloch and Lill. *Am. Math. Mon.* **2011**, *118*, 307–315. [CrossRef]
10. Haga, K. Proposal of a term origamics for plastic origami-workless scientific origami. In Proceedings of the Second International Meeting of Origami Science and Scientific Origami, Otsu, Japan, 29 November–2 December 1994; ABSTRACT A-3, pp. 29–32.
11. Haga, K. *Origamics: Mathematical Explorations through Paper Folding*; World Scientific: Singapore, 2008.
12. Hull, T. *Project Origami: Activities for Exploring Mathematics*, 2nd ed.; A K Peters: Natick, MA, USA; CRC Press: Boca Raton, FL, USA, 2013.
13. Aurenhammer, F.; Klein, R.; Lee, D.T. *Voronoi Diagrams and Delaunay Triangulations*; World Scientific: Singapore, 2013.
14. Fortune, S. A sweepline algorithm for Voronoi diagrams. In Proceedings of the Second Annual Symposium on Computational Geometry, Yorktown Heights, NY, USA, 2–4 June 1986; pp. 313–322.
15. Graham, R. An efficient algorith for determining the convex hull of a finite planar set. *Inf. Process. Lett.* **1972**, *1*, 132–133. [CrossRef]
16. Aichholzer, O.; Aurenhammer, F. Straight skeletons for general polygonal figures in the plane. In Proceedings of the Computing and Combinatorics, Hong Kong, China, 17–19 June 1996; Cai, J.Y., Wong, C.K., Eds.; Springer: Berlin/Heidelberg, Germany, 1996; pp. 117–126.
17. Chin, F.; Snoeyink, J.; Wang, C.A. Finding the Medial Axis of a Simple Polygon in Linear Time. *Discret. Comput. Geom.* **1999**, *21*, 405–420. [CrossRef]
18. Lang, R.J. A computational algorithm for origami design. In Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, 24–26 May 1996; pp. 98–105.
19. Blum, H. A transformation for extracting new descriptors of shape. In *Models for Perception of Speech and Visual Form*; Wathen-Dunn, W., Ed.; MIT Press: Cambridge, MA, USA, 1967.
20. Attali, D.; Boissonnat, J.D.; Edelsbrunner, H. Stability and Computation of Medial Axes—A State-of-the-Art Report. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 109–125. [CrossRef]

CONCLUSIONS

Although counting problem is a basic problem to ask, it might not be an easy question to answer. However, when we did answer to the counting problem systematically, we uncover the structure that has direct impact on the algorithm design.

In our case, the answers to counting problems has different effect on our three problem settings.

For the permutation problem in Phetmak and Fakcharoenphol (2023b), counting is the heart of the algorithm. Without a prior answer to how to count derangements would prohibit us to derive the algorithm in the first place.

In Wongwattanakij *et al.* (2023), the answer to counting problem allows us to speed up the algorithm, even though it is just a bound for the counting.

Finally, in Phetmak and Fakcharoenphol (2023a), the knowledge of counting reveal the structure of the overall problem. Result in a re-interpretation of the problem, which bring us to an easier algorithm.

RECOMMENDATIONS AND FUTURE WORK

Even though we devised faster algorithms with the help of counting problems, there are still many rooms for improvement. Some emerged from the overlooked due to shortage of time/resources during the researches.

In Wongwattanakij *et al.* (2023), although we obtained a bounded solution to the counting problem, it is enough to improves the algorithm to the simplest complexity, i.e., practically "close" the problem. However, if we can devise a simpler way to analyse the counting bound, we may standardize the counting technique and adapt to other problem in the family.

In Phetmak and Fakcharoenphol (2023b), our suggested counting formula is, despite trivial, require many step of computation. Precisely, it is a summation of consecutive incremental terms. If we could find another way to express the formula, maybe in recursion form, it might speed up the algorithm furthermore.

In Phetmak and Fakcharoenphol (2023a), the skeleton structure does not work well with non-convex polygon. We might improve this by using other geometrical structure, with the trade-off between computational time of the more complex structure.

REFERENCES

Abel, Z., E. D. Demaine, M. L. Demaine, D. Eppstein, A. Lubiw and R. Uehara. 2018. Flat foldings of plane graphs with prescribed angles and edge lengths. **Journal of Computational Geometry**. 9 (1): 74–93.

Aichholzer, O. and F. Aurenhammer. 1996. Straight skeletons for general polygonal figures in the plane, pp. 117–126. *In* Cai, J.-Y. and C. K. Wong, eds. **Computing and Combinatorics**. Springer Berlin Heidelberg, Berlin, Heidelberg.

Akitaya, H. A., E. D. Demaine and J. S. Ku. 2017. Simple folding is really hard. **J. Inf. Process.** 25: 580–589.

_____, B. Ballinger, E. D. Demaine, T. C. Hull and C. Schmidt. 2021. Folding points to a point and lines to a line, pp. 271–278. *In* **Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021**. Dalhousie University, Halifax, Nova Scotia, Canada.

Akyildiz, I., W. Su, Y. Sankarasubramaniam and E. Cayirci. 2002. Wireless sensor networks: a survey. **Computer Networks**. 38 (4): 393 – 422.

Ammari, H. M. 2012. On the problem of k-coverage in mission-oriented mobile wireless sensor networks. **Computer Networks**. 56 (7): 1935–1950.

Arndt, J. 2010. **Matters Computational: ideas, algorithms, source code**. Springer Science and Business Media.

Attali, D., J.-D. Boissonnat and H. Edelsbrunner. 2009. Stability and computation of medial axes-a state-of-the-art report. **Mathematical foundations of scientific visualization, computer graphics, and massive data exploration**. pp. 109–125.

Aurenhammer, F., R. Klein and D.-T. Lee. 2013. **Voronoi Diagrams and Delaunay Triangulations**. World Scientific, Singapore.

Baker, B. S. 1994. Approximation algorithms for NP-complete problems on planar graphs. **J. ACM**. 41 (1): 153–180.

Blum, H. A transformation for extracting new descriptors of shape. *In* Wathen-Dunn, W., ed. **Models for Perception of Speech and Visual Form**. MIT Press, Cambridge, MA, 1967.

Bottomley, H. 2001. **Entry A060540 in The On-Line Encyclopedia of Integer Sequences**. Available Source: http://oeis.org/A060540, December 21, 2021.

Charalambides, C. A. 2002. **Enumerative combinatorics**. CRC Press.

Chen, Z., X. Gao, F. Wu and G. Chen. 2016. A PTAS to minimize mobile sensor movement for target coverage problem, pp. 1–9. *In* **IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications**. IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications.

Chin, F., J. Snoeyink and C. A. Wang. 1999. Finding the medial axis of a simple polygon in linear time. **Discrete and Computational Geometry**. 21: 405–420.

Comtet, L. 2012. **Advanced Combinatorics: The art of finite and infinite expansions**. Springer Science and Business Media.

da Silva, P. H., A. Jamshidpey and S. Tavaré. 2022. The feller coupling for random derangements. **Stochastic Processes and their Applications**. 150: 1139–1164.

Dambrogio, J., A. Ghassaei, D. Smith, H. Jackson, M. Demaine, G. Davis, D. Mills, R. Ahrendt, N. Akkerman, D. van der Linden and E. Demaine. 2021. Unlocking history through automated virtual unfolding of sealed documents imaged by x-ray microtomography. **Nature Communications**. 12.

Montmort, deP. R. 1713. **Essay [sic] d'analyse sur les jeux de hazard**. Jacques Quillau.

Demaine, E. D. and J. O'Rourke. 2008. **Geometric Folding Algorithms: Linkages, Origami, Polyhedra**. Reprint edition. Cambridge University Press, USA.

Ding, L., W. Wu, J. Willson, L. Wu, Z. Lu and W. Lee. 2012. Constant-approximation for target coverage problem in wireless sensor networks, pp. 1584–1592. *In* **2012 Proceedings IEEE INFOCOM**. IEEE.

Elhoseny, M., A. Tharwat, X. Yuan and A. E. Hassanien. 2018. Optimizing k-coverage of mobile wsns. **Expert Systems with Applications**. 92: 142–153.

Erlebach, T. and M. Mihalák. 2010. A $(4+\epsilon)$-approximation for the minimum-weight dominating set problem in unit disk graphs, pp. 135–146. *In* Bampis, E. and K. Jansen, eds. **Approximation and Online Algorithms**. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ewens, W. 1972. The sampling theory of selectively neutral alleles. **Theoretical Population Biology**. 3 (1): 87–112.

Felton, S., M. Tolley, E. Demaine, D. Rus and R. Wood. 2014. A method for building self-folding machines. **Science**. 345 (6197): 644–646.

Flajolet, P. and M. Soria. 1990. Gaussian limiting distributions for the number of components in combinatorial structures. **Journal of Combinatorial Theory, Series A**. 53 (2): 165–182.

Fortune, S. 1986. A sweepline algorithm for voronoi diagrams, pp. 313–322. *In* **Proceedings of the second annual symposium on Computational geometry**. Association for Computing Machinery, New York, NY, United States.

Gao, X., Z. Chen, F. Wu and G. Chen. 2017. Energy efficient algorithms for k-sink minimum movement target coverage problem in mobile sensor network. **IEEE/ACM Trans. Netw.** 25 (6): 3616–3627.

Graham, R. 1972. An efficient algorith for determining the convex hull of a finite planar set. **Information Processing Letters**. 1 (4): 132–133.

Guo, J. and H. Jafarkhani. 2019. Movement-efficient sensor deployment in wireless sensor networks with limited communication range. **IEEE Transactions on Wireless Communications**. 18 (7): 3469–3484.

Haga, K. 1994. Proposal of a term origamics for plastic origami-workless scientific origami, pp. 29–32. *In* **Second International Meeting of Origami Science and Scientific Origami, ABSTRACT A-3**. Seian University of Art and Design.

_____. 2008. **Origamics: mathematical explorations through paper folding**. World Scientific.

Harvey, D. and J. Van Der Hoeven. 2021. Integer multiplication in time O(n log n). **Annals of Mathematics**. 193 (2): 563–617.

He, S., J. Chen, X. Li, X. Shen and Y. Sun. 2012. Cost-effective barrier coverage by mobile sensor networks, pp. 819–827. *In* **2012 Proceedings IEEE INFOCOM**. IEEE.

Hull, T. C. 2013. **Project Origami: Activities for Exploring Mathematics, Second Edition**. A K Peters/CRC Press.

_____. 2011. Solving cubics with creases: the work of beloch and lill. **The American Mathematical Monthly**. 118 (4): 307–315.

_____. 2020. **Origametry: Mathematical Methods in Paper Folding**. Cambridge University Press.

Hunt, H. B., M. V. Marathe, V. Radhakrishnan, S. Ravi, D. J. Rosenkrantz and R. E. Stearns. 1998. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. **Journal of Algorithms**. 26 (2): 238 – 274.

Khampeerpat, T. and C. Jaikaeo. 2017. Mobile sensor relocation for nonuniform and dynamic coverage requirements. **IEICE Trans. Inf. Syst.** 100-D (3): 520–530.

Lang, R. J. 1996. A computational algorithm for origami design, pp. 98–105. *In* **Proceedings of the twelfth annual symposium on Computational geometry**. Association for Computing Machinery, New York, NY, United States.

_____. 2009. Computational origami: From flapping birds to space telescopes, page 159–162. *In* **Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry**. Association for Computing Machinery, New York, NY, USA.

Liang, D., H. Shen and L. Chen. 2021. Maximum target coverage problem in mobile wireless sensor networks. **Sensors**. 21 (1).

Liu, B., O. Dousse, J. Wang and A. Saipulla. 2008. Strong barrier coverage of wireless sensor networks, page 411–420. *In* **Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing**. Association for Computing Machinery, New York, NY, USA.

_____, _____, P. Nain and D. Towsley. 2013. Dynamic coverage of mobile sensor networks. **IEEE Transactions on Parallel and Distributed Systems**. 24 (2): 301–311.

Mahboubi, H., K. Moezzi, A. G. Aghdam, K. Sayrafian-Pour and V. Marbukh. 2014. Distributed deployment algorithms for improved coverage in a network of wireless mobile sensors. **IEEE Transactions on Industrial Informatics**. 10 (1): 163–174.

Marathe, M. V., H. Breu, H. B. Hunt, S. S. Ravi and D. J. Rosenkrantz. 1995. Simple heuristics for unit disk graphs. **Networks**. 25 (2): 59–68.

Martínez, C., A. Panholzer and H. Prodinger. 2008. Generating random derangements, pp. 234–240. *In* **2008 Proceedings of the Fifth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)**. SIAM.

Matsumoto, M. and T. Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Transactions on Modeling and Computer Simulation (TOMACS)**. 8 (1): 3–30.

Mendonça, J. R. G. 2020. Efficient generation of random derangements with the expected distribution of cycle lengths. **Computational and Applied Mathematics**. 39 (3): 1–15.

Mikawa, K. and K. Tanaka. 2017. Linear-time generation of uniform random derangements encoded in cycle notation. **Discrete Applied Mathematics**. 217: 722–728.

_____, _____. 2023. Efficient linear-time ranking and unranking of derangements. **Information Processing Letters**. 179: 106288.

Myrvold, W. and F. Ruskey. 2001. Ranking and unranking permutations in linear time. **Information Processing Letters**. 79 (6): 281–284.

Nieberg, T. and J. Hurink. 2006. A PTAS for the minimum dominating set problem in unit disk graphs, pp. 296–306. *In* **Approximation and Online Algorithms: Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers 3**. Springer.

Nijenhuis, A. and H. S. Wilf. 2014. **Combinatorial algorithms: for computers and calculators**. Elsevier.

O'Rourke, J. 1987. **Art Gallery Theorems and Algorithms**. Oxford University Press, Inc., USA.

Phetmak, N. and J. Fakcharoenphol. 2023a. Folding every point on a polygon boundary to a point. **Algorithms**. 16 (6): 281.

Phetmak, N. and J. Fakcharoenphol. 2023b. Uniformly generating derangements with fixed number of cycles in polynomial time. **Thai Journal of Mathematics**. 21 (SI): 1–17.

Riordan, J. 2014. **An introduction to combinatorial analysis**. Princeton University Press.

Sattolo, S. 1986. An algorithm to generate a random cyclic permutation. **Information processing letters**. 22 (6): 315–317.

Leeuwen, vanE. J. 2005. Approximation algorithms for unit disk graphs, pp. 351–361. *In* Kratsch, D., ed. **Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers**. Springer.

_____. 2006. Better approximation schemes for disk graphs, pp. 316–327. *In* Arge, L. and R. Freivalds, eds. **Algorithm Theory – SWAT 2006**. Springer Berlin Heidelberg, Berlin, Heidelberg.

_____. 2009. **Optimization and approximation on systems of geometric objects**. PhD thesis, University of Amsterdam.

Wang, B. 2011. Coverage problems in sensor networks: A survey. **ACM Comput. Surv.** 43 (4): 32:1–32:53.

_____, H. B. Lim and D. Ma. 2009. A survey of movement strategies for improving network coverage in wireless sensor networks. **Computer Communications**. 32 (13): 1427–1436.

Wang, Y.-C. and Y.-C. Tseng. 2008. Distributed deployment schemes for mobile wireless sensor networks to ensure multilevel coverage. **IEEE Transactions on Parallel and Distributed Systems**. 19 (9): 1280–1294.

_____, C.-C. Hu and Y.-C. Tseng. 2008. Efficient placement and dispatch of sensors in a wireless sensor network. **IEEE Transactions on Mobile Computing**. 7 (2): 262–274.

Wongwattanakij, N., N. Phetmak, C. Jaikaeo and J. Fakcharoenphol. 2023. An improved PTAS for covering targets with mobile sensors. **arXiv preprint arXiv:2305.03946**.

Wu, W., Z. Zhang, W. Lee and D.-Z. Du. 2020. **Optimal Coverage in Wireless Sensor Networks**. Number 978-3-030-52824-9 in Springer Optimization and Its Applications. Springer.

Xu, X. and M. Song. 2014. Restricted coverage in wireless networks, pp. 558–564. *In* **IEEE INFOCOM 2014 - IEEE Conference on Computer Communications**. IEEE.

Yick, J., B. Mukherjee and D. Ghosal. 2008. Wireless sensor network survey. **Computer Networks**. 52 (12): 2292 – 2330.

Zhang, Z., X. Gao, W. Wu and D.-Z. Du. 2009. A PTAS for minimum connected dominating set in 3-dimensional wireless sensor networks. **Journal of Global Optimization**. 45 (3): 451–458.

Zou, F., Y. Wang, X.-H. Xu, X. Li, H. Du, P. Wan and W. Wu. 2011. New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. **Theor. Comput. Sci.** 412 (3): 198–208.

Zou, Y. and K. Chakrabarty. 2004. Sensor deployment and target localization in distributed sensor networks. **ACM Trans. Embed. Comput. Syst.** 3 (1): 61–91.