Dominating Set, Independent Set, Discrete k-Center, Dispersion, and Related Problems for Planar Points in Convex Position

Anastasiia Tkachenko
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points in the plane, its unit-disk graph $G(P)$ is a graph with P as its vertex set such that two points of P are connected by an edge if their (Euclidean) distance is at most 1. We consider several classical problems on $G(P)$ in a special setting when points of P are in convex position. These problems are all NP-hard in the general case. We present efficient algorithms for these problems under the convex position assumption.

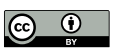
- For the problem of finding the smallest dominating set of $G(P)$, we present an $O(kn \log n)$ time algorithm, where k is the smallest dominating set size. We also consider the weighted case in which each point of P has a weight and the goal is to find a dominating set in $G(P)$ with minimum total weight; our algorithm runs in $O(n^3 \log^2 n)$ time. In particular, for a given k, our algorithm can compute in $O(kn^2 \log^2 n)$ time a minimum weight dominating set of size at most k (if it exists).
- For the discrete k-center problem, which is to find a subset of k points in P (called *centers*) for a given k, such that the maximum distance between any point in P and its nearest center is minimized. We present an algorithm that solves the problem in $O(\min\{n^{4/3} \log n + kn \log^2 n, k^2 n \log^2 n\})$ time, which is $O(n^2 \log^2 n)$ in the worst case when $k = \Theta(n)$. For comparison, the runtime of the current best algorithm for the continuous version of the problem where centers can be anywhere in the plane is $O(n^3 \log n)$.
- For the problem of finding a maximum independent set in $G(P)$, we give an algorithm of $O(n^{7/2})$ time and another randomized algorithm of $O(n^{37/11})$ expected time, which improve the previous best result of $O(n^6 \log n)$ time. Our algorithms can be extended to compute a maximum-weight independent set in $G(P)$ with the same time complexities when points of P have weights.
  - If we are looking for an (unweighted) independent set of size 3, we derive an algorithm of $O(n \log n)$ time; the previous best algorithm runs in $O(n^{4/3} \log^2 n)$ time (which works for the general case where points of P are not necessarily in convex position).
  - If points of P have weights and are not necessarily in convex position, we present an algorithm that can find a maximum-weight independent set of size 3 in $O(n^{5/3+\delta})$ time for an arbitrarily small constant $\delta > 0$. By slightly modifying the algorithm, a maximum-weight clique of size 3 can also be found within the same time complexity.
- For the dispersion problem, which is to find a subset of k points from P for a given k, such that the minimum pairwise distance of the points in the subset is maximized. We present an algorithm of $O(n^{7/2} \log n)$ time and another randomized algorithm of $O(n^{37/11} \log n)$ expected time, which improve the previous best result of $O(n^6)$ time.
  - If $k = 3$, we present an algorithm of $O(n \log^2 n)$ time and another randomized algorithm of $O(n \log n)$ expected time; the previous best algorithm runs in $O(n^{4/3} \log^2 n)$ time (which works for the general case where points of P are not necessarily in convex position).

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases Dominating set, k-center, geometric set cover, independent set, clique, vertex cover, unit-disk graphs, convex position, dispersion, maximally separated sets

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.73

Related Version *Full Version*: http://arxiv.org/abs/2501.00207



42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025).
Editors: Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyễn Kim Thắng;
Article No. 73; pp. 73:1–73:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Let P be a set of n points in the plane. The *unit-disk graph* of P, denoted by $G(P)$, is the graph with P as its vertex set such that two points are connected by an edge if their (Euclidean) distance is at most 1. Equivalently, $G(P)$ is the intersection graph of congruent disks with radius $1/2$ and centered at the points in P (i.e., two disks have an edge if they intersect). This model is particularly useful in applications such as wireless sensor networks, where connectivity is determined by signal ranges, represented by unit disks [6, 19, 42, 43].

1.1 Our results

We consider several classical problems on $G(P)$. These problems are all NP-hard. However, little attention has been given to special configurations of points, such as when the points are in convex position, despite the potential for significant algorithmic simplifications in such cases. In this paper, we systematically study these problems under the condition that the points of P are in convex position (i.e., every point of P appears as a vertex in the convex hull of P) and present efficient algorithms. We hope our results can lead to efficient solutions to other problems in this setting.

Dominating set. A *dominating set* of $G(P)$ is a subset S of vertices of $G(P)$ such that each vertex of $G(P)$ is either in S or adjacent to a vertex in S. The dominating set problem, which seeks a dominating set of the smallest size, is a classical NP-hard problem [19, 29, 31]. In the weighted case, each point of P has a weight and the problem is to find a dominating set of minimum total weight. The dominating set problem has been widely studied, with various approximation algorithms proposed [23, 27, 41, 55].

To the best of our knowledge, we are not aware of any previous work under the convex position assumption. For the unweighted case, we present an algorithm of $O(kn \log n)$ time, where k is the smallest dominating set size of $G(P)$. For the weighted case, we derive an algorithm of $O(n^3 \log^2 n)$ time. In particular, given any k, our algorithm can compute in $O(kn^2 \log^2 n)$ time a minimum-weight dominating set of size at most k.

Discrete k-center. A closely related problem is the *discrete k-center* problem. Given a number k, the problem is to compute a subset of k points in P (called *centers*) such that the maximum distance between any point in P and its nearest center is minimized. The problem, which is NP-hard [49], is also a classical problem with applications in clustering, facility locations, and network design. An algorithm for the dominating set problem can be used to solve the *decision version* of the discrete k-center problem: Given a value r and k, decide whether there exists a subset of k centers such that the distance from any point in P to its nearest center is at most r. Indeed, if we define the unit-disk graph of P with respect to r, then a dominating set of size k in the graph is a discrete k-center of P for r, and vice versa.

For the convex position case, we are not aware of any previous work. We propose an algorithm of $O(\min\{n^{4/3} \log n + kn \log^2 n, k^2 n \log^2 n\})$ time.

Independent set. An *independent set* of $G(P)$ is a subset of vertices such that no two vertices have an edge. The *maximum independent set* problem is to find an independent set of the largest cardinality. The problem of finding a maximum independent set in $G(P)$ is NP-hard [19]. Many approximation algorithms for the problem have been developed in the literature, e.g., [21, 22, 37, 38].

Under the convex position assumption, using the technique of Singireddy, Basappa, and Mitchell [47] for a dispersion problem (more details to be discussed later), one can find a maximum independent set in $G(P)$ in $O(n^6 \log n)$ time. We give a new algorithm of $O(n^{7/2})$ time,[1] and another randomized algorithm of $O(n^{37/11})$ expected time using the recent randomized result of Agarwal, Ezra, and Sharir [1]. Furthermore, our algorithm can be extended to compute a maximum-weight independent set of $G(P)$ within the same time complexity when points of P have weights; specifically, a *maximum-weight independent set* is an independent set whose total vertex weight is maximized. Since the vertices of a graph excluding an independent set form a vertex cover, our algorithm also computes a minimum-weight vertex cover of $G(P)$ in $O(n^{7/2})$ time or in randomized $O(n^{37/11})$ expected time.

Furthermore, we consider a small-size case that is to find an (unweighted) independent set of size 3 in $G(P)$. If P is not necessarily in convex position, the problem has been studied by Agarwal, Overmars, and Sharir [2], who presented an $O(n^{4/3} \log^2 n)$ time algorithm. We consider the convex position case and derive an algorithm of $O(n \log n)$ time. Note that finding an independent set of size 2 is equivalent to computing a farthest pair of points of P, which can be done in $O(n \log n)$ time using the farthest Voronoi diagram [45].

In addition, we consider a more general small-size case that is to find a maximum-weight independent set of size 3 in $G(P)$ when points of P have weights and are not necessarily in convex position. Our algorithm runs in $O(n^{5/3+\delta})$ time; δ refers to an arbitrarily small positive constant. Our technique can also be used to find a maximum-weight clique of size 3 in $G(P)$ within the same time complexity. In addition, we show that a maximum-weight independent set or clique of size 2 can be found in $n^{4/3}2^{O(\log^* n)}$ time. All these algorithms also work for computing the minimum-weight independent set or clique. We are not aware of any previous work on these weighted problems. As mentioned above, the problem of finding an (unweighted) independent set of size 3 can be solved in $O(n^{4/3} \log^2 n)$ time [2]. It is also known that finding an (unweighted) clique of size 3 in a disk graph (not necessarily unit-disk graph) can be done in $O(n \log n)$ time [30].

The dispersion problem. A related problem is the dispersion problem (also called *maximally separated set problem* [2]). Given P and a number k, we wish to find a subset of k points from P so that their minimum pairwise distance is maximized. The problem is NP-hard [50]. An algorithm for the independent set problem of $G(P)$ can be used as a decision algorithm for the dispersion problem: Given a value r, we can decide whether P has a subset of k points whose minimum pairwise distance is larger than r using the independent set algorithm (i.e., by defining an edge for two points in the graph if their distance is at most r).

Under the convex position assumption, Singireddy, Basappa, and Mitchell [47] previously gave an $O(n^4k^2)$ time algorithm for the problem. Using our independent set algorithm as a decision procedure and doing binary search among the interpoint distances of P, we present a new algorithm that can solve the problem in $O(n^{7/2} \log n)$ time, or in randomized $O(n^{37/11} \log n)$ expected time. For a special case where $k = 3$, the algorithm of [2] solves the problem in $O(n^{4/3} \log^3 n)$ time even if the points of P are not in convex position. Our new algorithm, which works on the convex-position case only, runs in $O(n \log^2 n)$ time. This is achieved using parametric search [20, 39] with our independent set algorithm as a decision algorithm. In addition, with our decision algorithm and Chan's randomized technique [11], we can obtain a randomized algorithm of $O(n \log n)$ expected time. We note that a recent work [35] proposed another algorithm of $O(n^2)$ time, apparently unaware of the result in [2].

[1] Throughout the paper, the algorithm runtime is deterministic unless otherwise stated.

1.2 Related work

Unit-disk graphs are a fundamental model in wireless networks, particularly where coverage and connectivity are governed by proximity [6, 19, 42, 43]. However, many classical graph problems, including coloring, vertex cover, independent set, and dominating set, remain NP-hard even when restricted to unit-disk graphs [19]. One exception is that finding a maximum clique in a unit-disk graph can be done in polynomial time [19, 25, 26] and the current best algorithm runs in $O(n^{2.5} \log n)$ time [26] (see [34] for a comment about improving the runtime to $O(n^{7/3+o(1)})$).

The assumption that points are in convex position can simplify certain problems that are otherwise NP-hard for general point sets in the plane. This has motivated the exploration of other computational problems under similar assumptions. For example, the *continuous* k-center problem where centers can be anywhere in the plane is NP-hard for arbitrary points but become polynomial time solvable under the convex position assumption [18]. The convex position constraint was even considered for classical problems that are already polynomial time solvable in the general case. For instance, Aggarwal, Guibas, Saxe, and Shor [4] gave a renowned linear time algorithm for computing the Voronoi diagram for a set of planar points in convex position. Refer to [15, 36, 44] for more work for points in convex position.

The k-center problem under a variety of constraints has received much attention. Particularly, when k, the number of centers, is two and the centers can be anywhere in the plane (referred to as the *continuous 2-center problem*), several near-linear time algorithms have been developed [12, 24, 46, 51], culminating in an optimal $O(n \log n)$ time [17]. The problem variations under other constraints were also considered. For example, the k-center problem can be solved in $O(n \log n)$ time if centers are required to lie on the same line [16, 53] or two lines [10]. The continuous one-center problem is the classical smallest enclosing circle problem and can be solved in linear time [40].

For the convex position case of the continuous k-center problem, Choi, Lee, and Ahn [18] proposed an $O(\min\{k, \log n\} \cdot n^2 \log n + k^2 n \log n)$ time algorithm. For comparison, the worst-case runtime of their algorithm is cubic, while our discrete k-center algorithm runs in near quadratic time.

The discrete 2-center problem also gets considerable attention. Agarwal, Sharir, and Welzl gave the first subquadratic $O(n^{4/3} \log^5 n)$ time algorithm [3]; the logarithmic factor was slightly improved by Wang [52]. As the continuous two-center problem can be solved in $O(n \log n)$ time [17] while the current best discrete two-center algorithm runs in $\Omega(n^{4/3})$ time [3, 52], the discrete problem appears more challenging than the continuous counterpart. This makes our discrete k-center algorithm even more interesting because it is almost a linear factor faster than the continuous k-center algorithm in [18]. Therefore, it is an intriguing question whether the algorithm in [18] can be further improved.

Other variations of the discrete k-center problem for small k were recently studied by Chan, He, and Yu [13], improving over previous results [8, 9, 32].

The dispersion problem and some of its variants have also been studied before. The general planar dispersion problem can be solved by an exact algorithm in $n^{O(\sqrt{k})}$ time [2]. If all points of P lie on a single line, Araki and Nakano [5] gave an algorithm of $O((2k^2)^k \cdot n)$ time (assuming that the points are not given sorted), which is $O(n)$ for a constant k. For a circular case where all points of P lie on a circle and the distance between two points is measured by their distance along the circle, the problem is solvable in $O(n)$ time [48], provided that the points are given sorted along the circle. We note that this implies that the line case problem, which can be viewed as a special case of the circular case, is also solvable in $O(n)$ time after the points are sorted on the line.

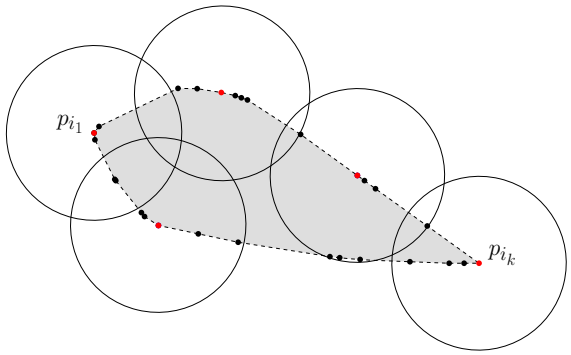


Figure 1 Illustrating the ordering property of S (the centers of the disks).

1.3 Our approach

The weighted dominating set problem reduces to the following problem: Given any k, find a minimum weight dominating set of size at most k. This is equivalent to finding a minimum weight subset of at most k points of P such that the union of the unit disks centered at these points covers P. Let S be an optimal solution for the problem (points of S are called *centers*). If we consider P as a cyclic list of points along the convex hull of P, then for each center $p \in S$, its unit disk D_p may cover multiple maximal contiguous subsequences (called *sublists*) of P. We prove that it is possible to assign at most two such sublists to each center $p \in S$ such that (1) p belongs to at least one of these sublists; (2) the union of the sublists assigned to all centers is P; (3) for every two centers $p_i, p_j \in S$, the sublists of the points assigned to p_i can be separated by a line from the sublists assigned to p_j. Using these properties, we further obtain the following structural property (called *ordering property*; see Figure 1) about the optimal solution S: There exists an ordering of the centers of S as $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ such that (1) p_{i_1} (resp., p_{i_k}) is only assigned one sublist; (2) if a center p_{i_j}, $1 < j < k$, is assigned two sublists, then one of them is on P_1, the portion of P from p_{i_1} to p_{i_k} clockwise, and the other is on P_2, the portion of P from p_{i_1} to p_{i_k} counterclockwise; (3) the order of the centers of the sublists along P_1 (resp., P_2) from p_{i_1} to p_{i_k} is a (not necessarily contiguous) subsequence of the above ordering.

The above ordering property is crucial to the success of our method. Using the property, we develop a dynamic programming algorithm of $O(kn^2 \log^2 n)$ time. Setting $k = n$ leads to an $O(n^3 \log^2 n)$ time algorithm for the original weighted dominating set problem. These properties are also applicable to the unweighted case, which is essentially a special case of the weighted problem. Using an additional greedy strategy, the runtime of the algorithm can be improved by roughly a linear factor for the unweighted case.

To solve the discrete k-center problem, we use the algorithm for the unweighted dominating set problem as the decision problem: Given any value r, determine whether $r \geq r^*$, where r^* is the optimal objective value, i.e., the minimum value for which there exist k centers such that the maximum distance from any point of P to its closest center is at most r^*. Observe that r^* must be equal to the distance of two points of P. As such, by doing binary search on the pairwise distances of points of P and applying the distance selection framework in [54] with our unweighted dominating set algorithm, we can compute r^* in $O(n^{4/3} \log n + kn \log^2 n)$ time. Furthermore, using parametric search [20, 39], we develop another algorithm of $O(k^2 n \log^2 n)$ time, which is faster than the first algorithm when $k = o(n^{1/6}/\sqrt{\log n})$.

For the independent set problem, our algorithm is a dynamic program, which is in turn based on the observation that the Voronoi diagram of a set of points in convex position forms a tree [4]. The (unweighted) size-3 case is solved by new observations and developing

efficient data structures. As discussed above, we tackle the dispersion problem by using the independent set algorithm as a decision procedure. For computing a maximum-weight independent set of size 3 for points in arbitrary position, our algorithm relies on certain interesting observations and a *tree-structured biclique partition* of P. Biclique partition has been studied before, e.g., [33, 54]. However, to our best knowledge, tree-structured biclique partitions have never been introduced before. Our result may find applications elsewhere.

Outline. The rest of the paper is organized as follows. After introducing notation in Section 2, we present our algorithms for the dominating set, the discrete k-center, the independent set, and the dispersion problems for points in convex position in Sections 3, 4, 5 and 6, respectively. As the only problem for points in arbitrary position studied in this paper, the size-3 weighted independent set problem is discussed in Section 7. Due to the space limit, many details and proofs are omitted but can be found in the full paper.

2 Preliminaries

We introduce some notations that will be used throughout the paper, in addition to those already defined in Section 1, e.g., P, n, $G(P)$.

A *unit disk* refers to a disk with radius 1; the boundary of a unit disk is a *unit circle*. For any point p in the plane, we use D_p to denote the unit disk centered at p. For any two points p and q in the plane, we use $|pq|$ to denote their (Euclidean) distance and use $\overline{pq}$ to denote the line segment connecting them. Let $\overrightarrow{pq}$ to denote the directed segment from p to q.

Let $\mathcal{H}(P)$ be the convex hull of P. If the points in P are in convex position, then we can consider P as a cyclic sequence. Specifically, let $P = \langle p_1, p_2, \ldots, p_n \rangle$ represent a cyclic list of the points ordered counterclockwise along $\mathcal{H}(P)$. We use a *sublist* to refer to a contiguous subsequence of P. Multiple sublists are said to be *consecutive* if their concatenation is also a sublist. For any two points p_i and p_j in P, we define $P[i, j]$ as the sublist of P from p_i counterclockwise to p_j, inclusive, i.e., if $i \leq j$, then $P[i, j] = \langle p_i, p_{i+1}, \ldots, p_j \rangle$; otherwise, $P[i, j] = \langle p_i, p_{i+1}, \ldots, p_n, p_1, \ldots, p_j \rangle$. We also denote by $P(i, j]$ the sublist $P[i, j]$ excluding p_i, and similarly for other variations, e.g., $P[i, j)$ and $P(i, j)$.

For simplicity of the discussion, we make a general position assumption that no three points of P are collinear and no four points lie on the same circle. This assumption is made without loss of generality as degenerate cases can be handled through perturbations.

3 The dominating set problem

In this section, we present our algorithms for the dominating set problem on a set P of n points in convex position. The weighted and unweighted cases are discussed in Sections 3.2 and 3.3, respectively. We first prove in Section 3.1 the structural properties that our algorithms rely on and then present these algorithms.

3.1 Structural properties

We examine the structural properties of the dominating sets in the unit-disk graph $G(P)$ for the weighted case, which are also applicable to the unweighted case.

Let $\mathcal{A}$ represent a partition of P into consecutive, nonempty, and disjoint sublists. Suppose $S \subseteq P$ is a dominating set of $G(P)$; points of S are called *centers*. It is not difficult to see that the union of the collection $\mathcal{D}$ of unit disks centered at the points in S covers P.

We say that a collection $\mathcal{D}$ of unit disks *covers* $\mathcal{A}$ if every sublist $\alpha \in \mathcal{A}$ is covered by at least one disk from $\mathcal{D}$. An *assignment* $\phi : \mathcal{A} \to S$ is a mapping from sublists in $\mathcal{A}$ to points in S, such that each sublist α is assigned to exactly one center $p_i \in S$ with $\alpha \subseteq D_{p_i}$. For each $p_i \in S$, we define G_{p_i} as the set of points in the sublists $\alpha \in \mathcal{A}$ that are assigned to p_i; G_{p_i} is called the *group* of p_i. Depending on the context, G_{p_i} may also represent the collection of sublists assigned to p_i. By definition, the groups of two centers of S are disjoint.

An assignment ϕ is said to be *line separable* if, for every two groups of ϕ, there exists a line ℓ that separates the points from the two groups, that is, the points of one group lie on one side of ℓ or on ℓ while those of the other group lie strictly on the other side of ℓ.

As discussed in Section 1.3, our main target is to prove the ordering property. This is achieved by proving a series of lemmas. We start with the lemma that proves a line separable property.

▶ **Lemma 1.** *Let S be a dominating set of $G(P)$. There exist a partition $\mathcal{A}$ of P and a line-separable assignment $\phi : \mathcal{A} \to S$ such that for any center $p_i \in S$, $p_i \in G_{p_i}$, meaning that a sublist of p_i contains p_i.*

For the assignment ϕ from Lemma 1, for each center $p_i \in S$, we refer to the sublist of p_i that contains p_i as the *main sublist* of p_i.

▶ **Lemma 2.** *Let S be a dominating set for $G(P)$. Then there exist a partition $\mathcal{A}$ of P and an assignment $\phi : \mathcal{A} \to S$ with the following properties: (1) ϕ is line separable; (2) each center of S is assigned at most two sublists, one of which is a main sublist.*

▶ **Lemma 3.** *Let S be an optimal dominating set and $\phi : \mathcal{A} \to S$ be the assignment given by Lemma 2. There exists a pair of centers (p_i, p_j) in S, called a* decoupling pair*, such that the following hold: (1) each of p_i and p_j has only one sublist; (2) for any center S that has two sublists, one sublist is in $P(i, j)$ while the other is in $P(j, i)$.*

Let S be an optimal dominating set and $\phi : \mathcal{A} \to S$ be the assignment given by Lemma 2. Let (p_i, p_j) be a decoupling pair from Lemma 3. For any center of $S \setminus \{p_i, p_j\}$, each of its sublist must be either entirely in $P(i, j)$ or in $P(j, i)$, and by Lemma 3, the center has at most one sublist in $P(i, j)$ and at most one sublist in $P(j, i)$. We finally prove in the following lemma the ordering property discussed in Section 1.3.

▶ **Lemma 4.** (The ordering property) *Let S be an optimal dominating set and $\phi : \mathcal{A} \to S$ be the assignment given by Lemma 2. Let (p_i, p_j) be a decoupling pair from Lemma 3. Then, there exists an ordering of all centers of S as $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ with $k = |S|$ such that (see Figure 1)*

1. *$p_i = p_{i_1}$ and $p_j = p_{i_k}$, i.e., p_{i_1} and p_{i_k} are the first and last points in the ordering, respectively.*
2. *The sequence of the centers of S that have at least one sublist in $P[i, j]$ (resp., $P[j, i]$) ordered by the points of their sublists appearing in $P[i, j]$ (resp., $P[j, i]$) from p_i to p_j is a (not necessarily contiguous) subsequence of the ordering.*
3. *For any t, $1 \le t \le k$, the sublists of the first t centers in the ordering are consecutive (i.e., their union, which is $\bigcup_{l=1}^{t} G_{i_l}$, is a sublist of P).*
4. *For any t, $2 \le t \le k$, $\bigcup_{l=1}^{t-1} G_{i_l} \subseteq \bigcup_{l=1}^{t} G_{i_l}$.*
5. *For any t, $1 \le t \le k$, each sublist of p_{i_t} appears at one end of $\bigcup_{l=1}^{t} G_{i_l}$ (if $t = k$, then $\bigcup_{l=1}^{t} G_{i_l}$ becomes the cyclic list of P; for convenience, we view $\bigcup_{l=1}^{t} G_{i_l}$ as a list by cutting it right after the clockwise endpoint of G_{i_k}).*

Proof. First of all, notice that the first two properties imply the last three. Therefore, it suffices to prove the first two properties.

Let S_1 (resp., S_2) be the subset of centers of $S \setminus \{p_i, p_j\}$ that have a sublist in $P(i,j)$ (resp., $P(j,i)$). By Lemma 3, each center of S_1 has at most one sublist in $P(i,j)$ and each center of S_2 has at most one sublist in $P(j,i)$. We add p_i and p_j to both S_1 and S_2. We sort all centers of S_1 as a sequence (called *the sorted sequence* of S_1) by the points of their sublists appearing in $P[i,j]$ from p_i to p_j (and thus p_i is the first center and p_j is the last one in the sequence). Similarly, we sort all centers of S_2 as a sequence (called *the sorted sequence* of S_2) by the points of their sublists appearing in $P[j,i]$ from p_i to p_j (and thus p_i is the first center and p_j is the last one in the sequence). To prove the first two properties of the lemma, it suffices to show the following *statement*: There exists an ordering of S such that (1) p_i is the first one in the ordering and p_j is the last one; (2) the sorted sequence of S_1 (resp., S_2) is a subsequence of the ordering.

We say that two centers $p_{j_1}, p_{j_2} \in S$ are *conflicting* if p_{j_1} appears in front of p_{j_2} in the sorted sequence of S_1 while p_{j_2} appears in front of p_{j_1} in the sorted sequence of S_2. It is not difficult to see that if no two centers of S are conflicting then the above statement holds. Assume to the contrary that there exist two centers $p_{j_1}, p_{j_2} \in S$ that are conflicting. Then, since the two centers are in both S_1 and S_2, each of them has two sublists. Since they are conflicting, by the definition of the sorted sequences of S_1 and S_2 and due to the convexity of P, the group of p_{j_1} cannot be separated from the group of p_{j_2} by a line, a contradiction with the line separable property of ϕ. ◀

3.2 The weighted dominating set problem

For each point $p_i \in P$, let w_i denote its weight. We assume that each $w_i > 0$, since otherwise p_i could always be included in the solution. For any subset $S \subseteq P$, let $w(S)$ denote the total weight of all points of S. We mainly consider the following *bounded size problem*: Given a number k, compute a dominating set S of minimum total weight with $|S| \leq k$ in the unit-disk graph $G(P)$. If we have an algorithm for this problem, then applying the algorithm with $k = n$ can compute a minimum weight dominating set for $G(P)$. Let S^* denote an optimal dominating set for the above bounded size problem. Define $W^* = w(S^*)$.

In what follows, we first describe the algorithm and then discuss how to implement the algorithm efficiently.

Algorithm description. We begin by introducing the following definition.

▶ **Definition 5.** *For two points $p_i, p_j \in P$ ($p_i = p_j$ is possible), define a_i^j as the index of the first point p of P counterclockwise from p_j such that $|p_i p| > 1$, and b_i^j the index of the first point p of P clockwise from p_j such that $|p_i p| > 1$ (if $|p_i p_j| > 1$, then $a_i^j = b_i^j = j$). If $|p_i p| \leq 1$ for all points $p \in P$, then let $a_i^j = b_i^j = 0$.*

For a subset $P' \subseteq P$, let $\mathcal{D}(P')$ denote the union of the unit disks centered at the points of P'. Note that a subset $S \subseteq P$ is a dominating set if and only if $P \subseteq \mathcal{D}(S)$.

Our algorithm has k iterations. In each t-th iteration with $1 \leq t \leq k$, we compute a set $\mathcal{L}_t$ of $O(n^2)$ sublists of P, and each sublist $L \in \mathcal{L}_t$ is associated with a weight $w'(L)$ and a set $S_L \subseteq P$ of at most t points. Our algorithm maintains the following invariant: For each sublist $L \in \mathcal{L}_t$, $w(S_L) \leq w'(L)$ and points of L are all covered by $\mathcal{D}(S_L)$. Suppose that there exists a set $S \subseteq P$ of k points such that $P \subseteq \mathcal{D}(S)$. Then we will show that $\mathcal{L}_k$ contains a sublist L that is P and $w'(L) \leq W^*$. As such, after k iterations, we only need to find all sublists of $\mathcal{L}_k$ that are P and then return the one with the minimum weight.

In the first iteration, for each point $p_i \in P$, we compute the two indices a_i^i and b_i^i; we show in Lemma 7 that this can be done in $O(\log n)$ time after $O(n \log n)$ time preprocessing. Then, let $\mathcal{L}_1 = \{P(b_i^i, a_i^i) \mid p_i \in P\}$. For each sublist $L = P(b_i^i, a_i^i)$ of $\mathcal{L}_1$, we set $S_L = \{p_i\}$ and $w'(L) = w_i$. Clearly, the algorithm invariant holds on all sublists of $\mathcal{L}_1$. This finishes the first iteration. Although $|\mathcal{L}_1| = O(n)$, as will be seen next, $|\mathcal{L}_t| = O(n^2)$ for all $t \geq 2$.

In general, suppose that we have a set $\mathcal{L}_{t-1}$ of $O(n^2)$ sublists and each sublist $L \in \mathcal{L}_{t-1}$ is associated with a weight $w'(L)$ and a set $S_L \subseteq P$ of at most $t-1$ points such that the algorithm invariant holds, i.e., $w(S_L) \leq w'(L)$ and points of L are all covered by $\mathcal{D}(S_L)$. We now describe the t-th iteration of the algorithm.

For each point $p_i \in P$, we perform a *counterclockwise processing procedure* as follows. For each point $p_j \in P$, we do the following. Compute the minimum weight sublist from $\mathcal{L}_{t-1}$ that contains $P[a_i^i, j]$; we call this step a *minimum-weight enclosing sublist query*. We show later that each such query can be answered in $O(\log^2 n)$ time after $O(n^2 \log n)$ time preprocessing on the sublists of $\mathcal{L}_{t-1}$. Let $P[j_{i1}, j_{i2}]$ be the sublist computed above. Then, we compute the index $a_i^{j_{i2}+1}$. By definition, the union of the following three sublists is a sublist of P: $P(b_i^i, a_i^i)$, $P[j_{i1}, j_{i2}]$, and $P(j_{i2}, a_i^{j_{i2}+1})$; denote by L the sublist. We set $S_L = S_{L'} \cup \{p_i\}$ and $w'(L) = w'(L') + w_i$, where $L' = P[j_{i1}, j_{i2}]$. We add L to $\mathcal{L}_t$. We next argue that the algorithm invariant holds for L, i.e., points of L are covered by $\mathcal{D}(S_L)$, $|S_L| \leq t$, and $w(S_L) \leq w'(L)$. Indeed, by definition, all the points of $P(b_i^i, a_i^i) \cup P(j_{i2}, a_i^{j_{i2}+1})$ are covered by the disk D_{p_i}. Since the sublist L' is from $\mathcal{L}_{t-1}$, by our algorithm invariant, L' is covered by $\mathcal{D}(S_{L'})$, $|S_{L'}| \leq t-1$, and $w(S_{L'}) \leq w'(L')$. Therefore, L is covered by $\mathcal{D}(S_{L'} \cup \{p_i\})$ and $|S_L| \leq t$. In addition, we have $w(S_L) \leq w(S_{L'}) + w_i \leq w'(L') + w_i = w'(L)$. As such, the algorithm invariant holds on L.

The above counterclockwise processing procedure for p_i will add $O(n)$ sublists to $\mathcal{L}_t$. Symmetrically, we perform a *clockwise processing procedure* for p_i, which will also add $O(n)$ sublists to $\mathcal{L}_t$. We briefly discuss it. Given $p_i \in P$, for each point $p_j \in P$, we compute the minimum weight sublist from $\mathcal{L}_{t-1}$ that contains $P[j, b_i^i]$. Let $P[j_{i3}, j_{i4}]$ be the sublist computed above. Then, we compute the index $b_i^{j_{i3}-1}$. Let L be the sublist that is the union of the following three sublists: $P(b_i^i, a_i^i)$, $P[j_{i3}, j_{i4}]$, and $P(b_i^{j_{i3}-1}, j_{i3})$. We let $S_L = S_{L'} \cup \{p_i\}$ and $w'(L) = w'(L') + w_i$, where $L' = P[j_{i3}, j_{i4}]$. As above, the algorithm invariant holds on L. We add L to $\mathcal{L}_t$. In this way, the t-th iteration computes $O(n^2)$ sublists in $\mathcal{L}_t$.

After the k-th iteration, we find from all sublists of $\mathcal{L}_k$ that are P the one L^* whose weight $w'(L^*)$ is the minimum. Based on the ordering property in Lemma 4, the next lemma shows that S_{L^*} is an optimal dominating set.

▶ **Lemma 6.** *S_{L^*} is an optimal dominating set and $W^* = w'(L^*)$.*

Time analysis. In each iteration, we perform $O(n^2)$ operations for computing indices a_i^j and b_i^j, and perform $O(n^2)$ minimum-weight enclosing sublist queries. We show later that each query takes $O(\log^2 n)$ time after $O(n^2 \log n)$ time preprocessing. As such, each iteration of the algorithm takes $O(n^2 \log^2 n)$ time and the total time of the algorithm is thus $O(kn^2 \log^2 n)$.

Algorithm implementation. The following lemma provides a data structure for computing the indices a_i^j and b_i^j.

▶ **Lemma 7.** *We can construct a data structure for P in $O(n \log n)$ time such that the indices a_i^j and b_i^j can be computed in $O(\log n)$ time for any two points $p_i, p_j \in P$.*

Given a set $\mathcal{L}$ of m sublists of P, each sublist has a weight. We wish to build a data structure to answer the following minimum-weight enclosing sublist queries: Given a sublist L, compute the minimum weight sublist of $\mathcal{L}$ that contains L. We have the following lemma.

▶ **Lemma 8.** *We can construct a data structure for $\mathcal{L}$ in $O(m \log m)$ time, with $m = |\mathcal{L}|$, so that each minimum-weight enclosing sublist query can be answered in $O(\log^2 m)$ time.*

Theorem 9 summarizes our result. Applying Theorem 9 with $k = n$ leads to Corollary 10.

▶ **Theorem 9.** *Given a number k and a set P of n weighted points in convex position in the plane, we can find in $O(kn^2 \log^2 n)$ time a minimum-weight dominating set of size at most k in the unit-disk graph $G(P)$, or report no such dominating set exists.*

▶ **Corollary 10.** *Given a set P of n weighted points in convex position in the plane, we can compute a minimum-weight dominating set in the unit-disk graph $G(P)$ in $O(n^3 \log^2 n)$ time.*

3.3 The unweighted case

In this section, we consider the unweighted dominating set problem. The goal is to compute the smallest dominating set in the unit-disk graph $G(P)$. Note that all properties for the weighted case are also applicable here. In particular, by setting all point weights to 1 and applying Theorem 9, one can solve the unweighted problem in $O(n^3 \log^2 n)$ time. We provide an improved algorithm of $O(kn \log n)$ time, where k is the smallest dominating set size.

Algorithm description. We follow the iterative algorithmic scheme of the weighted case, but incorporate a greedy strategy using the property that all points of P have the same weight.

In each t-th iteration of the algorithm, $t \geq 1$, we compute a set $\mathcal{L}_t$ of $O(n)$ sublists and each list $L \in \mathcal{L}_t$ is associated with a set $S_L \subseteq P$ of at most t points. Our algorithm maintains the following invariant: For each sublist $L \in \mathcal{L}_t$, all points of L are covered by $\mathcal{D}(S_L)$, i.e., the union of the unit disks centered at the points of S_L. If k is the smallest dominating set size, we show in Lemma 11 that after k iterations, $\mathcal{L}_k$ is guaranteed to contain a sublist that is P. Thus, we can stop the algorithm as soon as the first sublist that is P is computed.

Initially, we compute the indices a_i^i and b_i^i for all points $p_i \in P$. By Lemma 7, this takes $O(\log n)$ time after $O(n \log n)$ time preprocessing. In the first iteration, we have $\mathcal{L}_1 = \{P(b_i^i, a_i^i) \mid p_i \in P\}$. For each sublist $L = P(b_i^i, a_i^i) \in \mathcal{L}_1$, we set $S_L = \{p_i\}$. Clearly, the algorithm invariant holds.

In general, suppose that we have a set $\mathcal{L}_{t-1}$ of $O(n)$ sublists such that the algorithm invariant holds. We assume that no sublist of $\mathcal{L}_{t-1}$ is P. Then, the t-th iteration of the algorithm works as follows. For each point $p_i \in P$, we perform the following *counterclockwise processing procedure*. We first compute the sublist of $\mathcal{L}_{t-1}$ that contains $p_{a_i^i}$ and has the most counterclockwise endpoint. This is done by a *counterclockwise farthest enclosing sublist query*. We show later in Section 3.3 that each such query takes $O(\log n)$ time after $O(n \log n)$ time preprocessing for $\mathcal{L}_{t-1}$. Let $P[j_{i1}, j_{i2}]$ be the sublist computed above. Then, we compute the index $a_i^{j_{i2}+1}$ in $O(\log n)$ time by Lemma 7. Note that the union of the following three sublists is a sublist L of P: $P(b_i^i, a_i^i)$, $P[j_{i1}, j_{i2}]$, and $P(j_{i2}, a_i^{j_{i2}+1})$. We add L to $\mathcal{L}_t$ and set $S_L = S_{L'} \cup \{p_i\}$ with $L' = P[j_{i1}, j_{i2}]$. By our algorithm invariant, points of L' are covered by $\mathcal{D}(S_{L'})$. By definition, points of $P(b_i^i, a_i^i) \cup P(j_{i2}, a_i^{j_{i2}+1})$ are covered by D_{p_i}. Therefore, all points of L are covered by $\mathcal{D}(S_L)$. Hence, the algorithm invariant holds for L. In addition, if L is P, we stop the algorithm and return S_L as an optimal dominating set.

Symmetrically, we perform a *clockwise processing procedure* for p_i. We compute the sublist from $\mathcal{L}_{t-1}$ that contains b_i^i and has the most clockwise endpoint; this is done by a *clockwise farthest enclosing sublist query*. Let $P[j_{i3}, j_{i4}]$ be the sublist computed above. Then, we compute the index $b_i^{j_{i3}-1}$. Let L be the sublist that is the union of the following three

sublists: $P(b_i^i, a_i^i)$, $P[j_{i3}, j_{i4}]$, and $P(j_{i3}, b_i^{j_{i3}-1})$. Let $S_L = S_{L'} \cup \{p_i\}$ with $L' = P[j_{i3}, j_{i4}]$. As above, the algorithm invariant holds on L. We add L to $\mathcal{L}_t$. If L is P, then we stop the algorithm and return S_L as an optimal dominating set.

The following lemma proves the correctness of the algorithm.

▶ **Lemma 11.** *If the algorithm first time computes a sublist L that is P, then S_L is the smallest dominating set of $G(P)$.*

Time analysis. In each iteration, we perform $O(n)$ operations for computing indices a_i^j and b_i^j and $O(n)$ counterclockwise/clockwise farthest enclosing sublist queries. Computing indices a_i^j and b_i^j takes $O(\log n)$ time by Lemma 7. We show in Lemma 12 that each counterclockwise/clockwise farthest enclosing sublist query can be answered in $O(\log n)$ time after $O(n \log n)$ time preprocessing. As such, each iteration runs in $O(n \log n)$ time and the total time of the algorithm is $O(kn \log n)$, where k is the smallest dominating set size.

Algorithm implementation. It remains to describe the data structure for answering counterclockwise/clockwise farthest enclosing sublist queries. We only discuss the counterclockwise case as the clockwise case can be handled analogously. Given a set $\mathcal{L}$ consisting of n sublists of P, the goal is to build a data structure to answer the following counterclockwise farthest enclosing sublist queries: Given a point $p \in P$, find a sublist in $\mathcal{L}$ that contains p with the farthest counterclockwise endpoint from p. We have the following lemma.

▶ **Lemma 12.** *We can construct a data structure for $\mathcal{L}$ in $O(n \log n)$ time such that each counterclockwise farthest enclosing sublist query can be answered in $O(\log n)$ time.*

We conclude with the following theorem and corollary, which will be used in Section 4 to solve the discrete k-center problem.

▶ **Theorem 13.** *Given a set P of n points in convex position in the plane, the smallest dominating set of the unit-disk graph $G(P)$ can be computed in $O(kn \log n)$ time, where k is the size of the smallest dominating set.*

▶ **Corollary 14.** *Given k, r, and a set P of n points in convex position in the plane, one can do the following in $O(kn \log n)$ time: determine whether there exists a subset $S \subseteq P$ of at most k points such that the distance from any point of P to its closest point in S is at most r, and if so, find such a subset S.*

Proof. We redefine the unit-disk graph of P with a parameter r, where two points in P are connected by an edge if their distance is at most r. We then apply the algorithm of Theorem 13. If the algorithm finds a sublist L that is P within k iterations, then we return $S = S_L$; otherwise such a subset S as in the lemma statement does not exist. Since we run the algorithm for at most k iterations, the total time of the algorithm is $O(kn \log n)$. ◀

4 The discrete k-center problem

In this section, we present our algorithm for the discrete k-center problem. Let P be a set of n points in convex position in the plane. Given a number k, the goal is to compute a subset $S \subseteq P$ of at most k points (called *centers*) so that the maximum distance between any point in P and its nearest center is minimized. Let r^* denote the optimal objective value.

Given a value r, the *decision problem* is to determine whether $r \geq r^*$, or equivalently, whether there exist a set of k centers in P such that the distance from any point of P to its closest center is at most r. By Corollary 14, the problem can be solved in $O(kn \log n)$ time. Clearly, r^* is equal to the distance of two points of P, that is $r^* \in R$, where R is defined as the set of all pairwise distances between points in P. If we explicitly compute R and then perform a binary search on R using the algorithm of Corollary 14 as a *decision algorithm*, then r^* can be computed in $O(n^2 + kn \log^2 n)$ time. We can improve the algorithm by using the distance selection algorithms, which can find the k-th smallest value in R in $O(n^{4/3} \log n)$ time for any given k [33, 54]. In fact, by applying the algorithmic framework of Wang and Zhao [54] with our decision algorithm, r^* can be computed in $O(n^{4/3} \log n + nk \log^2 n)$ time.

In the following, we present another algorithm of $O(k^2 n \log^2 n)$ time using the parametric search [20, 39]. This algorithm is faster than the above one when $k = o(n^{1/6}/\sqrt{\log n})$.

We simulate the decision algorithm over the unknown optimal value r^*. The algorithm maintains an interval $(r_1, r_2]$ that contains r^*. Initially, $r_1 = -\infty$ and $r_2 = \infty$. During the algorithm, the decision algorithm is invoked on certain *critical values* r to determine whether $r \geq r^*$; based on the outcome, the interval $(r_1, r_2]$ is shrunk accordingly so that the new interval still contains r^*. Upon completion, we can show that $r^* = r_2$ must hold.

Algorithm overview. For any r, certain variables in our decision algorithm are now defined with respect to r as the radius of unit disks and therefore may be considered as functions of r. For example, we use $a_i^j(r)$ to represent a_i^j when the unit disk radius is r. The algorithm has k iterations. We wish to compute the sublist set $\mathcal{L}_t(r^*)$ in each t-th iteration, $1 \leq t \leq k$. Specifically, the set $\mathcal{L}_1(r^*)$ relies on $a_i^i(r^*)$ and $b_i^i(r^*)$ for all points $p_i \in P$. As such, in the first iteration, we will compute $a_i^i(r^*)$ and $b_i^i(r^*)$ for all $p_i \in P$. The computation process will generate certain critical values r, call the decision algorithm on these values, and shrink the interval $(r_1, r_2]$ accordingly. After that, $\mathcal{L}_1(r^*)$ can be computed.

In a general t-th iteration, our goal is to compute the set $\mathcal{L}_t(r^*)$. We assume that the set $\mathcal{L}_{t-1}(r^*)$ is already available with an interval $(r_1, r_2]$ containing r^*. Then, for each $p_i \in P$, we perform a counterclockwise processing procedure. We first compute the sublist of $\mathcal{L}_{t-1}(r^*)$ with the farthest counterclockwise endpoint and containing $a_i^i(r^*)$. This procedure depends solely on $a_i^i(r^*)$ and $\mathcal{L}_{t-1}(r^*)$, which are already available, and thus no critical values are generated. Suppose that $P[j_{i1}(r^*), j_{i2}(r^*)]$ is the sublist computed above. The next step is to compute $a_i^{j_{i2}(r^*)+1}(r^*)$. This step will again generate critical values and shrink the interval $(r_1, r_2]$. After that, we add to $\mathcal{L}_t(r^*)$ the sublist that is the union of the following three sublists: $P(b_i^i(r^*), a_i^i(r^*))$, $P[j_{i1}(r^*), j_{i2}(r^*)]$, and $P(j_{i2}(r^*), a_i^{j_{i2}(r^*)+1})$. Similarly, we perform a clockwise processing procedure for each point $p_i \in P$. After that, the set $\mathcal{L}_t(r^*)$ is computed. The details can be found in the full version.

In summary, the algorithm can compute r^* in $O(k^2 n \log^2 n)$ time. Combining with the $O(n^{4/3} \log n + kn \log^2 n)$ time algorithm discussed earlier, we obtain the following result.

▶ **Theorem 15.** *Given a set P of n points in convex position in the plane and a number k, we can compute in $O(\min\{n^{4/3} \log n + kn \log^2 n, k^2 n \log^2 n\})$ time a subset $S \subseteq P$ of size at most k, such that the maximum distance from any point of P to its nearest point in S is minimized.*

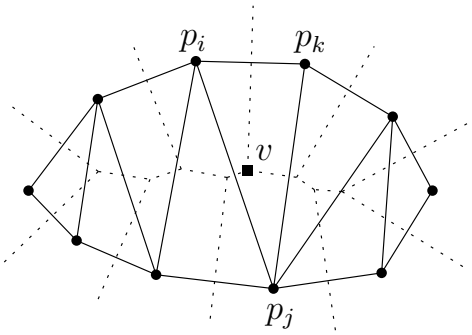


Figure 2 Illustrating $\mathcal{DT}(S)$, the solid segments, and $\mathcal{VD}(S)$, the dotted segments.

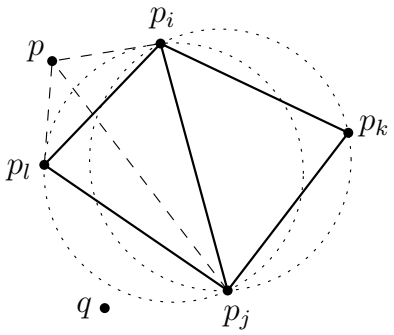


Figure 3 Illustrating Lemma 18.

5 The independent set problem

In this section, we present our algorithms for the independent set problem, assuming that the points of P are in convex position. In Section 5.1, we present the algorithm for computing a maximum-weight independent set. Section 5.2 gives the algorithm for computing an (unweighted) independent set of size 3.

5.1 The maximum-weight independent set problem

Recall that $P = \langle p_1, p_2, \ldots, p_n \rangle$ is a cyclic list ordered along $\mathcal{H}(P)$ in counterclockwise order. For each point $p_i \in P$, let w_i denote its weight. We assume that each $w_i > 0$ since otherwise p_i can be simply ignored, which would not affect the optimal solution. For any subset $P' \subseteq P$, let $w(P')$ denote the total weight of all points of P'.

For any three points p_1, p_2, p_3, let $D(p_1, p_2, p_3)$ denote the disk whose boundary contains them. Thus, $\partial D(p_1, p_2, p_3)$ is the unique circle through these points. For any compact region B in the plane, we use ∂B to denote its boundary and use $\overline{B}$ to denote the complement region of B in the plane. In particular, for a disk D in the plane, ∂D is its bounding circle, and $\overline{D}$ refers to the region of the plane outside D.

In what follows, we first describe the algorithm and explain why it is correct, and then discuss how to implement the algorithm efficiently.

5.1.1 Algorithm description and correctness

To motivate our algorithm and demonstrate its correctness, we first examine the optimal solution structure and develop a recursive relation on which our dynamic program is based.

Let S be a maximum-weight independent set of $G(P)$, or equivalently, S is a maximum-weight subset of P such that the minimum pairwise distance of the points of S is larger than 1. Let $\mathcal{DT}(S)$ denote the Delaunay triangulation of S. If (p, q) is the closest pair of points of S, then $\overline{pq}$ must be an edge of $\mathcal{DT}(S)$ and in fact, the shortest edge of $\mathcal{DT}(S)$ [45]. As such, finding a maximum-weight independent set of $G(P)$ is equivalent to finding a maximum-weight subset $S \subseteq P$ such that the shortest edge of $\mathcal{DT}(S)$ has length larger than 1. The algorithm in [47] is based on this observation, which also inspires our algorithm.

Consider a triangle $\triangle p_i p_j p_k$ of $\mathcal{DT}(S)$ such that the points p_i, p_j, p_k are in the counterclockwise order of P (i.e., ordered counterclockwise on $\mathcal{H}(P)$). Due to the property of Delaunay triangulation, the disk $D(p_i, p_j, p_k)$ does not contain any point of $S \setminus \{p_i, p_j, p_k\}$ [45]. Since the points of S are in convex position, we have the following observation.

▶ **Observation 16.** *$\mathcal{DT}(S)$ does not contain an edge connecting two points from any two different subsets of $\{P(i,j), P(j,k), P(k,i)\}$; see Figure 2.*

Observation 16 implies the following: To find an optimal solution S, if we know that $\triangle p_i p_j p_k$ is a triangle in $\mathcal{DT}(S)$, since no point of $S \setminus \{p_i, p_j, p_k\}$ lies in the disk $D(p_i, p_j, p_k)$, we can independently search $P(i,j) \cap \overline{D(p_i, p_j, p_k)}$, $P(j,k) \cap \overline{D(p_i, p_j, p_k)}$, and $P(k,i) \cap \overline{D(p_i, p_j, p_k)}$, respectively. This idea forms the basis of our dynamic program.

Let W^* denote the total weight of a maximum-weight independent set of $G(P)$.

For any pair of indices (i,j) with $|p_i p_j| > 1$, we call (i,j) a *canonical pair* and define $f(i,j)$ as the total weight of a maximum-weight subset P' of $P(i,j)$ such that $P' \cup \{p_i, p_j\}$ forms an independent set of $G(P)$; if no such subset P' exists, then $f(i,j) = 0$. Computing $f(i,j)$ is a subproblem in our dynamic program. For simplicity, we let $f(i,j) = -(w_i + w_j)$ if (i,j) is not canonical, i.e., $|p_i p_j| \leq 1$. Lemma 17 explains why we are interested in $f(i,j)$.

▶ **Lemma 17.** $W^* = \max_{1 \leq i,j \leq n}(f(i,j) + w_i + w_j)$.

By Lemma 17, to compute W^*, it suffices to compute $f(i,j)$ for all pairs of indices $1 \leq i, j \leq n$ and the one with the largest $f(i,j) + w_i + w_j$ leads to the optimal solution. To compute $f(i,j)$, we define another type of subproblems that will be used in our algorithm.

For any three points p_i, p_j, p_k such that they are ordered counterclockwise in P and their minimum pairwise distance is larger than 1, we call (i,j,k) a *canonical triple*.

For a canonical triple (i,j,k), by slightly abusing the notation, we define $f(i,j,k)$ as the total weight of a maximum-weight subset P' of $P(i,j) \cap \overline{D(p_i, p_j, p_k)}$ such that $P' \cup \{p_i, p_j\}$ is an independent set; if no such subset P' exists, then $f(i,j,k) = 0$. For any canonical pair (i,j), if we consider p_0 a dummy point to the left of $\overrightarrow{p_i p_j}$ and infinitely far from the supporting line of $\overline{p_i p_j}$ so that $D(p_i, p_j, p_0)$ becomes the left halfplane of $\overrightarrow{p_i p_j}$, then $f(i,j,0)$ following the above definition is exactly $f(i,j)$; for convenience, we also consider $(i,j,0)$ a canonical triple. To make the discussion concise, we often use $f(i,j,0)$ instead of $f(i,j)$ since the way we compute $f(i,j,0)$ is consistent with the way we compute $f(i,j,k)$ for $k \neq 0$.

For any canonical triple (i,j,k), define $P_k(i,j) = \{p \mid p \in P(i,j), p \notin D(p_i, p_j, p_k), |pp_i| > 1, |pp_j| > 1\}$. For any canonical pair (i,j), define $P_0(i,j) = \{p \mid p \in P(i,j), |pp_i| > 1, |pp_j| > 1\}$. Note that $P_0(i,j)$ is consistent with $P_k(i,j)$ if we consider p_0 a dummy point as defined above. Observe also that $P_k(i,j) = P_0(i,j) \cap \overline{D(p_i, p_j, p_k)}$ for any canonical triple (i,j,k). By definition, $f(i,j,k)$ (including the case $k = 0$) is the total weight of a maximum-weight independent set $P' \subseteq P_k(i,j)$; this is the reason we introduce the notation $P_k(i,j)$.

The following lemma gives the recursive relation of our dynamic programming algorithm.

▶ **Lemma 18.** *For any canonical triple (i,j,k), including the case $k = 0$, the following holds (see Figure 3):*

$$f(i,j,k) = \begin{cases} \max_{p_l \in P_k(p_i,p_j)}(f(i,l,j) + f(l,j,i) + w_l), & \text{if } P_k(i,j) \neq \emptyset \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

With Lemma 18, it remains to find an order to solve the subproblems so that when computing $f(i,j,k)$, the values $f(i,l,j)$ and $f(l,j,i)$ for all $p_l \in P_k(p_i,p_j)$ are available.

For any two points $p_i, p_j \in P$, we call $\overline{p_i p_j}$ a *diagonal*.

We process the diagonals $\overline{p_i p_j}$ for all $1 \leq i, j \leq n$ in the following way. For each $j = 2, \ldots, n$ in this order, we enumerate $i = j-1, j-2, \ldots, 1$ to process $\overline{p_i p_j}$ as follows. If $|p_i p_j| \leq 1$, then we set $f(i,j) = -(w_i + w_j)$. Otherwise, (i,j) is a canonical pair, and we compute $f(i,j)$, i.e., $f(i,j,0)$, by Equation (1); one can check that the values $f(i,l,j)$ and $f(l,j,i)$ for all $p_l \in P_0(p_i,p_j)$ have already been computed. Next, for each point $p_k \in P(j,i)$

with $|p_ip_k| > 1$ and $|p_jp_k| > 1$, (i, j, k) is a canonical triple and we compute $f(i, j, k)$ by Equation (1); again, the values $f(i, l, j)$ and $f(l, j, i)$ for all $p_l \in P_k(p_i, p_j)$ have already been computed. Finally, by Lemma 17, we can return the largest $f(i, j) + w_i + w_j$ among all canonical pairs (i, j) as W^*. The algorithm only computes the value W^*, but by the standard back-tracking technique a maximum-weight independent set can also be obtained.

5.1.2 Algorithm implementation

We can easily implement the algorithm in $O(n^4)$ time. Indeed, there are $O(n^3)$ subproblems $f(i, j, k)$. Each subproblem can be computed in $O(n)$ time by checking every point $p_l \in P_k(i, j)$. As such, the total time is $O(n^4)$. We give a better algorithm below.

Specifically, we show that for each canonical pair (i, j) we can compute the subproblems $f(i, j, k)$ for all $p_k \in P(j, i)$ in a total of $O(n^{3/2})$ time. To this end, we reduce the problem to an *offline outside-disk range max-cost query* problem. For each point $p_l \in P_k(i, j)$, we define the *cost* of p_l as $cost(p_l) = f(i, l, j) + f(l, j, i) + w_l$. Recall that $P_0(i, j) = \{p \mid p \in P(i, j), |pp_i| > 1, |pp_j| > 1\}$ and $P_k(i, j) = P_0(i, j) \cap \overline{D(p_i, p_j, p_k)}$. As such, computing $f(i, j, k)$ is equivalent to finding the maximum-cost point of $P_0(i, j)$ outside the query disk $D(p_i, p_j, p_k)$. Our goal is to answer all such disk queries for all $p_k \in P(j, i)$. We note that this problem can be solved in $O(n^{15/11})$ expected time by applying the recent randomized algorithm of Agarwal, Ezra, and Sharir [1]. In the full version, we present a deterministic algorithm of $O(n^{3/2})$ time by using cuttings [14]. We thus have the following result.

▶ **Theorem 19.** *Given a set P of n weighted points in convex position in the plane, a maximum-weight independent set in the unit-disk graph of P can be computed in $O(n^{7/2})$ deterministic time, or in $O(n^{37/11})$ randomized expected time.*

Using the randomized result of [1], the problem can be solved in $O(n^{37/11})$ expected time. The following corollary will be used in Section 6 to solve the dispersion problem.

▶ **Corollary 20.** *Given a set P of n points in convex position in the plane and a number $r > 0$, one can find in $O(n^{7/2})$ deterministic time or in $O(n^{37/11})$ randomized expected time a maximum subset of P such that the distance of every two points of the subset is larger than r.*

5.2 Computing an independent set of size 3

To facilitate the discussion in Section 6 for the dispersion problem, we consider the following problem: Given a set P of n points in convex position and a number $r > 0$, find three points from P whose minimum pairwise distance is larger than or equal to r.

We follow the notation in Section 2. In particular, $P = \langle p_1, p_2, \ldots, p_n \rangle$ is a cyclic list ordered along $\mathcal{H}(P)$ counterclockwise. In the following definition, for each point $p_i \in P$, we define a_i similarly to a_i^i in Definition 5 with respect to r (i.e., change "> 1" to "$\geq r$").

▶ **Definition 21.** *For each point $p_i \in P$, define a_i as the index of the first point p of P counterclockwise from p_i such that $|p_ip| \geq r$; similarly, define $b_i \in P$ as the index of the first point p clockwise from P such that $|p_ip| \geq r$. If $|p_ip| < r$ for all points $p \in P$, then let $a_i = b_i = 0$.*

We will make use of the following lemma, which has been proved previously in [35].

▶ **Lemma 22** ([35])**.** *P has three points whose minimum pairwise distance is at least r if and only if there exists a point $p_i \in P$ such that $P[a_i, b_i]$ has two points whose distance is at least r.*

If p_i is a point of P such that $P[a_i, b_i]$ has two points whose distance is at least r, we say that p_i is a *feasible point*. By Lemma 22, it suffices to find a feasible point (if it exists). Our algorithm comprises two procedures. In the first procedure, we compute a_i and b_i for all points $p_i \in P$. This can be done in $O(n \log n)$ time by slightly changing the algorithm of Lemma 7. The second procedure finds a feasible point. In the following, we present an $O(n \log n)$ time algorithm. We start with the following easy but crucial observation.

▶ **Observation 23.** *A point $p_i \in P$ is a feasible point if and only if there is a point $p_k \in P[a_i, b_i]$ such that p_{a_k} is also in $P[a_i, b_i]$ and (p_i, p_k, p_{a_k}) is in counterclockwise order in P.*

For each point $p_i \in P$, note that $a_i \neq i$ must hold; we define $a'_i = \begin{cases} a_i, & \text{if } i < a_i \\ a_i + n, & \text{otherwise.} \end{cases}$ By definition, $i < a'_i$ always holds and $a'_i = a_i$ if $a'_i \leq n$. Note that if $a_i = b_i$, then $P[a_i, b_i]$ has only one point, and therefore p_i cannot be a feasible point. As such, we only need to focus on the points p_i with $a_i \neq b_i$. Our algorithm is based on the following lemma, which in turn relies on Observation 23.

▶ **Lemma 24.** *For each $p_i \in P$, we have the following.*

1. *If $a_i < b_i$, then p_i is a feasible point if and only if $\min_{k \in [a_i, b_i]} a'_k \leq b_i$.*
2. *If $a_i > b_i$, then p_i is a feasible point if and only if $\min\limits_{k \in [a_i, n]} a'_k \leq b_i + n$ or $\min\limits_{k \in [1, b_i]} a'_k \leq b_i$.*

Define an array $A[1 \cdots n]$ such that $A[k] = a'_k$ for each $1 \leq k \leq n$. In light of Lemma 24, for each point $p_i \in P$, we can determine whether p_i is a feasible point using at most two *range-minima* queries of the following type: Given a range $[i, j]$ with $i \leq j$, find the minimum number in the subarray $A[i \cdots j]$. It is possible to answer each range-minima query in $O(1)$ time after $O(n)$ time preprocessing on A [7, 28]. For our problem, since it suffices to have $O(\log n)$ query time and $O(n \log n)$ preprocessing time, we can use a simple solution by constructing an augmented binary search tree. As such, in $O(n \log n)$ time we can find a feasible point or report that no such point exists.

In summary, in $O(n \log n)$ time we can determine whether P has three points whose minimum pairwise distance is at least r. If the answer is yes, then these three points can also be found within the same time complexity according to the proofs of Lemmas 22 and 24. We conclude with the following theorem.

▶ **Theorem 25.** *Given a set P of n points in convex position in the plane and a number r, in $O(n \log n)$ time one can find three points of P whose minimum pairwise distance is at least r or report that no such three points exist.*

6 The dispersion problem

Given a set P of n points in convex position in the plane and a number k, the dispersion problem is to find a subset of k points from P so that the minimum pairwise distance of the points of the subset is maximized.

Let r^* be the minimum pairwise distance of the points in an optimal solution subset. The value $r^* \in R$, where R is the set of pairwise distances of the points of P. Given a value r, the *decision problem* is to determine whether $r < r^*$, or equivalently, whether P has a subset of k points whose minimum pairwise distance is larger than r. By Corollary 20, the decision problem can be solved in $O(n^{7/2})$ time. Using the decision algorithm and doing binary search on the sorted list of R, r^* can be computed in $O(n^{7/2} \log n)$ time.

▶ **Theorem 26.** *Given a set of n points in convex position in the plane and a number k, one can find a subset of k points whose minimum pairwise distance is maximized in $O(n^{7/2}\log n)$ deterministic time, or in $O(n^{37/11}\log n)$ randomized expected time.*

The size-3 case. We now consider the case where $k = 3$. Given a number r, the *decision problem* is to determine whether $r \leq r^*$, that is, whether P has three points whose minimum pairwise distance is at least r. With Theorem 25 as our *decision algorithm*, the decision problem is solvable in $O(n \log n)$ time. To compute r^*, we follow the standard parametric search framework [39] and simulate the decision algorithm on the unknown optimal value r^* with an interval $[r_1, r_2)$ that contains r^*. In fact, Cole's technique [20] can be applied here. In addition, we observe that Chan's randomized technique [11] is applicable here. Consequently, applying the technique with our $O(n \log n)$ time decision algorithm leads to a randomized algorithm of $O(n \log n)$ expected time for the problem.

▶ **Theorem 27.** *Given a set of n points in convex position in the plane, one can find three points whose minimum pairwise distance is maximized in $O(n \log^2 n)$ deterministic time or in $O(n \log n)$ randomized expected time.*

7 The size-3 weighted independent set for points in arbitrary position

Given a set P of n weighted points in the plane in arbitrary position, the problem is to find a maximum-weight independent set of size 3 in $G(P)$. As the size of our target independent set is fixed, we allow points to have negative weights.

Define $\overline{G(P)}$ as the complement graph of $G(P)$. The problem is equivalent to finding a maximum-weight clique of size 3 in $\overline{G(P)}$. We want to partition $\overline{G(P)}$ into bicliques, i.e., complete bipartite graphs. We give the formal definition below.

▶ **Definition 28** (Biclique partition). *Define a* biclique partition *of $\overline{G(P)}$ as a collection of edge-disjoint bicliques $\Gamma(P) = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ such that the following are satisfied:*

1. *For each pair $(a, b) \in A_t \times B_t \in \Gamma$, $|ab| > 1$.*
2. *For any points $a, b \in P$ with $|ab| > 1$, Γ has a unique biclique $A_t \times B_t$ that contains (a, b).*

In our problem, we need a stronger version of the partition, called a *tree-structured biclique partition*, and Lemma 30 explains why we introduce the concept.

▶ **Definition 29** (Tree-structured biclique partition). *A biclique partition $\Gamma(P) = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ is* tree-structured *if all the subsets A_t's form a tree $\mathcal{T}_A$ such that for each internal node A_t, all its children subsets form a partition of A_t.*

▶ **Lemma 30.** *Let $\Gamma(P) = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ be a tree-structured biclique partition of $\overline{G(P)}$ and $\mathcal{T}_A$ is the tree formed by the subsets A_t's. Then, the triple $a, b, c \in P$ forms an independent set in $G(P)$ if and only if $\Gamma(P)$ has a biclique (A_t, B_t) that contains a pair (a, b), and A_t has an ancestor subset $A_{t'}$ in $\mathcal{T}_A$ such that $c \in B_{t'}$ and $|bc| > 1$.*

With this, we obtain the following result; see the full version for the details.

▶ **Theorem 31.** *Given a set P of n weighted points in the plane, one can find a maximum-weight (or minimum-weight) independent set (or clique) of size 3 in the unit-disk graph $G(P)$ in $O(n^{5/3+\delta})$ time, for any arbitrarily small constant $\delta > 0$.*

References

1 Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Semi-algebraic off-line range searching and biclique partitions in the plane. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 4:1–4:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.4.

2 Pankaj K. Agarwal, Mark H. Overmars, and Micha Sharir. Computing maximally separated sets in the plane. *SIAM Journal on Computing*, 36(3):815–834, 2006. doi:10.1137/S0097539704446591.

3 Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. The discrete 2-center problem. *Discrete and Computational Geometry*, 20:287–305, 1998. doi:10.1007/PL00009387.

4 Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989. doi:10.1007/BF02187749.

5 Tetsuya Araki and Shin-ichi Nakano. Max–min dispersion on a line. *Journal of Combinatorial Optimization*, 44(3):1824–1830, 2022. doi:10.1007/s10878-020-00549-5.

6 Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 37(1):1–24, 2005. doi:doi:10.1239/aap/1113402397.

7 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, 2000. doi:10.1007/10719839_9.

8 Sergei Bespamyatnikh and David G. Kirkpatrick. Rectilinear 2-center problems. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG)*, 1999. URL: http://www.cccg.ca/proceedings/1999/fp55.pdf.

9 Sergei Bespamyatnikh and Michael Segal. Rectilinear static and dynamic discrete 2-center problems. In *Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS)*, pages 276–287, 1999. doi:10.1007/3-540-48447-7_28.

10 Binay Bhattacharya, Ante Ćustić, Sandip Das, Yuya Higashikawa, Tsunehiko Kameda, and Naoki Katoh. Geometric p-center problems with centers constrained to two lines. In *Proceedings of the 18th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGG)*, pages 24–36, 2015. doi:10.1007/978-3-319-48532-4_3.

11 Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discrete and Computational Geometry*, 22(4):547–567, 1999. doi:10.1007/PL00009478.

12 Timothy M. Chan. More planar two-center algorithms. *Computational Geometry: Theory and Applications*, 13:189–198, 1999. doi:10.1016/S0925-7721(99)00019-X.

13 Timothy M. Chan, Qizheng He, and Yuancheng Yu. On the fine-grained complexity of small-size geometric set cover and discrete k-center for small k. In *Proceedings of th 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 34:1–34:19, 2023. doi:10.4230/LIPIcs.ICALP.2023.34.

14 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993. doi:10.1007/BF02189314.

15 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, Micha Sharir, and Emo Welzl. Improved bounds on weak ϵ-nets for convex sets. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 495–504, 1993. doi:10.1145/167088.167222.

16 Danny Z. Chen, Jian Li, and Haitao Wang. Efficient algorithms for the one-dimensional k-center problem. *Theoretical Computer Science*, 592:135–142, 2015. doi:10.1016/j.tcs.2015.05.028.

17 Kyungjin Cho, Eunjin Oh, Haitao Wang, and Jie Xue. Optimal algorithm for the planar two-center problem. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 40:1–40:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.40.

18 Jongmin Choi, Jaegun Lee, and Hee-Kap Ahn. Efficient k-center algorithms for planar points in convex position. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 262–274, 2023. doi:10.1007/978-3-031-38906-1_18.

19 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

20 Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34:200–208, 1987. doi:10.1145/7531.7537.

21 Gautam K. Das, Guilherme D. da Fonseca, and Ramesh K. Jallu. Efficient independent set approximation in unit disk graphs. *Discrete Applied Mathematics*, 280:63–70, 2020. doi:10.1016/j.dam.2018.05.049.

22 Gautam K. Das, Minati De, Sudeshna Kolay, Subhas C. Nandy, and Susmita Sur-Kolay. Approximation algorithms for maximum independent set of a unit disk graph. *Information Processing Letters*, 115:439–446, 2015. doi:10.1016/j.ipl.2014.11.002.

23 Minati De and Abhiruk Lahiri. Geometric dominating-set and set-cover via local-search. *Computational Geometry*, 113(C):102007, 2023. doi:10.1016/j.comgeo.2023.102007.

24 David Eppstein. Faster construction of planar two-centers. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 131–138, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314198.

25 David Eppstein. Graph-theoretic solutions to computational geometry problems. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 1–16, 2009. doi:10.1007/978-3-642-11409-0_1.

26 Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In *Proceedings of the 39th International Symposium on Computational Geometry (SoCG)*, pages 30:1–30:17, 2023. doi:10.4230/LIPIcs.SoCG.2023.30.

27 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA)*, pages 243–254, 2010. doi:10.1007/978-3-642-15775-2_21.

28 Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984. doi:10.1137/0213024.

29 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

30 Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Triangles and girth in disk graphs and transmission graphs. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, pages 64:1–64:14, 2019. doi:10.4230/LIPIcs.ESA.2019.64.

31 Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-3-540-68279-0_8.

32 Matthew J. Katz, Klara Kedem, and Michael Segal. Discrete rectilinear 2-center problems. *Computational Geometry*, 15(4):203–214, 2000. doi:10.1016/S0925-7721(99)00052-8.

33 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997. doi:10.1137/S0097539794268649.

34 J. Mark Keil and Debajyoti Mondal. The maximum clique problem in a disk graph made easy. *arXiv:2404.03751*, 2024. URL: https://arxiv.org/abs/2404.03751, doi:10.48550/arXiv.2404.03751.

35 Yasuaki Kobayashi, Shin-ichi Nakano, Kei Uchizawa, Takeaki Uno, Yutaro Yamaguchi, and Katsuhisa Yamanaka. An $O(n^2)$-time algorithm for computing a max-min 3-dispersion on a point set in convex position. *IEICE Transactions on Information and Systems*, 105(3):503–507, 2022. doi:10.1587/transinf.2021FCP0013.

36 Andrzej Lingas. On approximation behavior and implementation of the greedy triangulation for convex planar point sets. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoSG)*, pages 72–79, 1986. doi:10.1145/10515.10523.

37 Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, Sekharipuram S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995. doi:10.1002/net.3230250205.

38 Tomomi Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings of the 2nd Japanese Conference on Discrete and Computational Geometry (JCDCG)*, pages 194–200, 1998. doi:10.1007/978-3-540-46515-7_16.

39 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

40 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983. doi:10.1137/0212052.

41 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010. doi:10.1007/s00454-010-9285-9.

42 Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, pages 234–244, 1994. doi:10.1145/190809.190336.

43 Charles E. Perkins and Pravin Bhagwat. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, 1999. doi:10.1109/MCSA.1999.749281.

44 Dana S. Richards and Jeffrey S. Salowe. A rectilinear steiner minimal tree algorithm for convex point sets. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 447, pages 201–212, 1990. doi:10.1007/3-540-52846-6_90.

45 Michael I. Shamos and Dan Hoey. Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975. doi:10.1109/SFCS.1975.8.

46 Micha Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete and Computational Geometry*, 18:125–134, 1997. doi:10.1007/PL00009311.

47 Vishwanath R. Singireddy, Manjanna Basappa, and Joseph S.B. Mitchell. Algorithms for k-dispersion for points in convex position in the plane. In *Proceedings of the 9th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 59–70, 2023. doi:10.1007/978-3-031-25211-2_5.

48 Kuo-Hui Tsai and Da-Wei Wang. Optimal algorithms for circle partitioning. In *Proceedings of the Third Annual International Computing and Combinatorics Conference (COCOON)*, pages 304–310, 1997. doi:10.1007/BFb0045097.

49 Vijay V. Vazirani. *Approximation Algorithms*. Springer, Berlin Heidelberg, 1st edition, 2001.

50 Da-Wei Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information processing letters*, 28(6):281–286, 1988. doi:10.1016/0020-0190(88)90174-3.

51 Haitao Wang. On the planar two-center problem and circular hulls. *Discrete and Computational Geometry*, 68:1175–1226, 2022. doi:10.1007/S00454-021-00358-5.

52 Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14(1):343–394, 2023. doi:10.20382/jocg.v14i1a13.

53 Haitao Wang and Jingru Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry and Application*, 26(3):185–210, 2016. doi:10.1142/S0218195916600049.

54 Haitao Wang and Yiming Zhao. Improved algorithms for distance selection and related problems. In *Proceedings of the 31st Annual European Symposium on Algorithms (ESA)*, pages 101:1–101:14, 2023. doi:10.4230/LIPIcs.ESA.2023.101.

55 Frank Ángel Hernández Mira, Ernesto Parra Inza, José María Sigarreta Almira, and Nodari Vakhania. A polynomial-time approximation to a minimum dominating set in a graph. *Theoretical Computer Science*, 930:142–156, 2022. doi:10.1016/j.tcs.2022.07.020.