Efficient independent set approximation in unit disk graphs

Gautam Das, Guilherme D. da Fonseca, Ramesh K Jallu

Efficient Independent Set Approximation in Unit Disk Graphs*

Gautam K. Das†[1], Guilherme D. da Fonseca[2], and Ramesh K. Jallu[1]

[1]Department of Mathematics, Indian Institute of Technology Guwahati
[2]Université d'Auvergne and LIMOS, Clermont-Ferrand, France

May 27, 2016

Abstract

We consider the maximum (weight) independent set problem in unit disk graphs. The high complexity of the existing polynomial-time approximation schemes motivated the development of faster constant-approximation algorithms. In this article, we present a 2.16-approximation algorithm that runs in $O(n \log^2 n)$ time and a 2-approximation algorithm that runs in $O(n^2 \log n)$ time for the unweighted version of the problem. In the weighted version, the running times increase by an $O(\log n)$ factor. Our algorithms are based on a classic strip decomposition, but we improve over previous algorithms by efficiently using geometric data structures. We also propose a PTAS for the unweighted version.

Keywords : Maximum Independent Set, Unit Disk Graph, Approximation Algorithms, Polynomial Time Approximation Scheme

1 Introduction

Given a set $\mathcal{P}$ of n points in the plane, a *unit disk graph* (UDG) has vertices $\mathcal{P}$ and an edge pq corresponds to a pair of points $p, q \in \mathcal{P}$, if $\|pq\| \leq 1$, where $\|pq\|$ denotes the Euclidean distance between p and q. Equivalently, a unit disk graph is the intersection graph of disks of unit diameter centered at the points in $\mathcal{P}$. In this geometric setting, an *independent set* consists of a subset of $\mathcal{P}$ with minimum distance greater than 1. In the *unweighted* version, a *maximum independent set* (MIS) is an independent set of maximum cardinality. In the *weighted* version, we are given a weight function $w : \mathcal{P} \to \mathbb{R}^+$ and we want to find a *maximum weight independent set* (MWIS), that is, an independent set that maximizes the sum of the weights of its points. The M(W)IS problem is well studied due to its wide range of applications, including but not limited to map labeling, clustering, wireless ad-hoc networks, and coding theory.

*A preliminary version of this paper appeared in CALDAM, 2016.

E-mail addresses: gkd@iitg.ernet.in, guilherme.dias_da_fonseca@udamail.fr, j.ramesh@iitg.ernet.in
†corresponding author

The MIS problem on UDGs is known to be NP-hard [11]. There are polynomial-time approximation schemes (PTASs) for several optimization problems on unit disk graphs [10, 14, 17, 18, 22, 23], including maximum (weight) independent set. However, the high complexities of the PTASs motivated the recent study of faster constant-approximation algorithms, notably for minimum dominating set [5, 12, 13, 15] and maximum (weight) independent set [13, 14].

The history of constant-approximation algorithms for M(W)IS in unit disk graphs is rich. We begin by reviewing some constant-approximation algorithms. The following algorithms receive the point coordinates as input and execute on the Real RAM and the $O(n)$-time algorithms use constant-time hashing and floor function (otherwise, their running times increase to $O(n \log n)$).

In their seminal work, Marathe et al. [20] presented greedy algorithms which attain a 3-approximation algorithm for the unweighted version, and a 5-approximation for the weighted version. Using efficient data structures, their algorithms can be implemented to run in $O(n)$ time. Using a strip decompositon, Matsui [22] presented an $O(n^2)$-time algorithm with approximation ratio $(1 + (2/\sqrt{3}) + \varepsilon) < 2.16$ for the weighted version. A different strip decomposition yields an $O(n^3)$-time 2-approximation algorithm for the weighted version [14]. The use of coresets give an $O(n)$-time $(4 + \varepsilon)$-approximation algorithm for the weighted version [13].

Decomposing the problem into strips and using the shifting strategy [17], Matsui [22] proposed the first PTAS for the problem. For any positive integer $k \geq 2$, it gives a solution of weight at least $(1 - 1/k)OPT$ in $O(n^{4\lceil 2(k-1)/\sqrt{3} \rceil})$ time, where OPT is the weight of an optimal solution. For any positive integer $k > 1$, another PTAS based on a similar strategy [14] gives a $((1 + \frac{1}{k})^2)$-approximation in $O(n^{\sigma_k \log k})$ time and $O(n)$ space, where $\sigma_k \leq \frac{7k}{3} + 2$. The complexity of the PTASs [18, 23] based on different strategies tend to be even higher.

The problem is also studied in the context of fixed parameter tractable algorithms. Van Leeuwen [25] proposed a fixed parameter tractable algorithm which runs in $O(t^2 2^{2t} n)$ time, where the parameter t is called the *thickness* of the UDG. A UDG is said to have thickness t, if each strip in the slab decomposition (of width 1) of the UDG contains at most t disk centers.

The study of independent sets in geometric intersection graphs is not limited to UDGs, but studied also for other objects such as rectangles, pseudo disks, etc. For a given set R of rectangles of fixed size, Agarwal et al. [1] proposed a 2-factor approximation algorithm for the MIS problem that runs in $O(n \log n)$ time, as well as a PTAS. For a given set of arbitrary rectangles of bounded aspect ratio (or, more generally arbitrary fat convex bodies) in $\mathbb{R}^d$, Chan [6] proposed a PTAS that computes an $(1 + \varepsilon)$-approximation in $O(n^{O(1/\varepsilon^{d-1})})$ time and space. Chan et al. [9] considered the same problem for pseudo disks in the plane.

Our results In this paper, we present four new approximation algorithms that improve the best running time for the same approximation ratios. Table 1 summarizes the constant-approximation algorithms for the problem, including our results. Next, we describe the techniques that allowed such improvements. We also present a PTAS for the unweighted version.

Given two real numbers h and r, a *strip* of *width* h *rooted* at r is the unbounded planar

Maximum Cardinality IS				Maximum Weight IS			
Ratio	Time	Space	Ref.	Ratio	Time	Space	Ref.
3	$O(n)$	$O(n)$	[20]	$4+\varepsilon$	$O(n)$	$O(n)$	[13]
2.16	$O(n \log^2 n)$	$O(n)$	Sec. 3	2.16	$O(n \log^3 n)$	$O(n \log n)$	Sec. 3
2	$O(n^2 \log n)$	$O(n^2)$	Sec. 4	2	$O(n^2 \log^2 n)$	$O(n^2 \log n)$	Sec. 4

Table 1: Approximation ratios and complexities of several constant-approximation algorithms for maximum (weight) independent set in unit disk graphs.

region $\mathbb{R} \times [r, r+h]$. The algorithms from [14, 22] are based on a procedure to solve the problem optimally when all points lie inside a sufficiently small strip. Strips of width $\sqrt{3}/2$ are used in the 2.16-approximation algorithm [22], while strips of width 1 are used in the 2-approximation algorithm [14]. The algorithms presented in this paper are based on the same strategy and strip widths. However, several geometric data structures are used to speed-up the algorithms. The data structures we use are presented in Section 2, followed by the 2.16-approximation algorithm in Section 3, and the 2-approximation algorithm in Section 4. The proposed PTAS is discussed in Section 5.

2 Data Structures

Given a set $\mathcal{P}$ of n *data points* in d-dimensional space, an *emptiness query* consists of determining whether a given *query range* R contains no data point, that is, whether $\mathcal{P} \cap R = \emptyset$, returning an arbitrary point in $\mathcal{P} \cap R$ if it exists. A *maximum query* is defined similarly, but the data points have real weights and the query finds a point of maximum weight in $\mathcal{P} \cap R$ if it exists. In an *anti-disk* query, the *range* R is the complement of a disk of arbitrary radius r centered at q, that is, the region formed by all points with Euclidean distance greater than r from q.

Maximum queries reduce to emptiness queries as follows. As far as we know, this reduction has never been published, but it is likely that other researchers independently noticed the same construction.

Lemma 2.1. *Consider a set of n data points with weights and a query range. A data structure for answering emptiness queries with storage $S(n) = \Omega(n)$, preprocessing time $B(n) = \Omega(n)$, and query time $Q(n)$ yields a data structure for answering maximum queries for the same range with preprocessing time $O(B(n) \log n)$, query time $O(Q(n) \log n)$, and storage $O(S(n) \log n)$.*

Proof. We assume that all weights are distinct, otherwise we break the ties arbitrarily. We build a binary tree where each node contains a data structure for emptiness queries as follows. The root contains the emptiness data structure for all data points. Consider a node which stores the data structure for a set of points $\mathcal{P}$. Let μ be the median weight among the points in $\mathcal{P}$. The left subtree is defined recursively for the points in $\mathcal{P}$ with weight at most μ, and the right subtree is defined analogously for the remaining points of $\mathcal{P}$. The recursion stops when $\mathcal{P}$ has only 1 point.

Notice that the tree has $\lceil \lg n \rceil + 1$ levels, and exactly n data points at each level. The total storage is therefore

$$S'(n) = \sum_{\ell=0}^{\lceil \lg n \rceil + 1} 2^\ell S\left(\frac{n}{2^\ell}\right) \leq \sum_{\ell=0}^{\lceil \lg n \rceil + 1} S(n) = O(S(n) \log n),$$

where the first inequality uses that $S(n) = \Omega(n)$. The preprocessing time is calculated the same way.

To answer a maximum query, we verify at the root node whether the query range is empty. If not, we verify if the query range is empty for the right subtree. If it is empty, we recurse on the left subtree. Otherwise, we recurse on the right subtree. The procedure ends when we reach a leaf. The leaf reached contains the point of maximum weight inside the query range, which we return. The bound on the query time follows from the fact that we performed $O(\log n)$ emptiness queries, one per level of the tree. □

The previous lemma deserves two remarks. (i) It follows from the proof that the $\log n$ factors in the storage, preprocessing time, and query time disappear if the respective complexities are $\Omega(n^{1+\varepsilon})$, $\Omega(n^{1+\varepsilon})$, and $\Omega(n^\varepsilon)$. (ii) Using known data structures for half-space emptiness queries such as [8, 21], we beat the lower bounds in [2, 4] for half-space maximum queries. This is possible because the set of generators depends on the point weights, which is forbidden by the semi-group arithmetic model in which the lower bounds are stated.

Many techniques yield efficient data structures for anti-disk emptiness searching. For example, one can use a farthest-point Voronoi diagram [16, Ch. 7] and point location [16, Ch. 6]. Alternatively, one can use lifting (called inversion in [16, Ch. 8]), duality [16, Ch. 8], and point location [16, Ch. 6].

Lemma 2.2. *Given a set of n planar points, there exists a data structure that answers anti-disk* emptiness *queries in $O(\log n)$ time with $O(n)$ storage and $O(n \log n)$ preprocessing time.*

Combining Lemmas 2.1 and 2.2, we have:

Lemma 2.3. *Given a set of n weighted planar points, there exists a data structure that answers anti-disk* maximum *queries in $O(\log^2 n)$ time with $O(n \log n)$ storage and $O(n \log^2 n)$ preprocessing time.*

Sometimes, the reduction from Lemma 2.1 is an overkill, and a logarithmic factor can be avoided by a simpler approach.

Lemma 2.4. *Given a set of n planar points with integer weights in $\{1, \ldots, n\}$, there exists a data structure that answers anti-disk* maximum *queries in $O((w_{max} - w_{ret}) \log n)$ time with $O(n)$ storage and $O(n \log n)$ preprocessing time, where w_{max} is the maximum weight among the points in the data structure and w_{ret} is the weight of the point returned by the query, or 0 if the query is empty.*

Proof. We keep an array of n data structures $D_1, \ldots, D_n$ from Lemma 2.2 where D_i stores the points of weight i. To answer a query, we start from the maximum i that corresponds to a non-empty D_i, decrementing i until the query returns non-empty. Analyzing the data structure is trivial. □

A search problem is *decomposable* if for any set S and $S' \subseteq S$, the answer to a query $q(S)$ can be calculated in constant time from the results of $q(S')$ and $q(S \setminus S')$. The following Lemma is due to Overmars and Leeuwen [24] (see also [3]).

Lemma 2.5. *A static data structure for a composable problem storing n elements with preprocessing time $B(n)$, query time $Q(n)$, and storage $S(n)$ yields a semi-dynamic data structure supporting insertions for the same problem with insertion time $O(B(n)\log n/n)$, query time $O(Q(n)\log n)$, and storage $O(S(n)+\log n)$.*

The $\log n$ factor in the query and insertion times disappear if the query and preprocessing times are respectively $\Omega(n^{\varepsilon})$ and $\Omega(n^{1+\varepsilon})$. Sometimes, dynamic data structures specifically designed for a particular problem are more efficient than the dynamized version, but this is not known to be the case for half-space emptiness. A specific data structure supporting insertions and deletions is presented in [7].

Combining Lemma 2.5 with Lemmas 2.3 and 2.4 we have the following lemmas.

Lemma 2.6. *Given a set of n weighted planar points, there exists a semi-dynamic data structure that answers anti-disk* maximum *queries in $O(\log^3 n)$ time with $O(n\log n)$ storage and $O(\log^3 n)$ insertion time.*

Lemma 2.7. *Given a set of n planar points with integer weights in $\{1,\ldots,n\}$, there exists a semi-dynamic data structure that answers anti-disk* maximum *queries in $O((w_{max}-w_{ret})\log^2 n)$ time with $O(n)$ storage and $O(\log^2 n)$ insertion time, where w_{max} is the maximum weight among the points in the data structure and w_{ret} is the weight of the point returned by the query, or 0 if the query is empty.*

3 Small strips

In this section, we present a 2.16-approximation algorithm for the maximum (weight) independent set problem on unit disk graphs. Recall that an unit disk graph is represented as a set $\mathcal{P}$ of n points where an edge pq corresponds to a pair of points $p,q \in \mathcal{P}$ with $\|pq\| \leq 1$. We use $x(p_i)$ and $y(p_i)$ to represent the x and y coordinates respectively of $p_i \in \mathcal{P}$. In order to describe multiple results with a unified treatment, we assume that a weight function with $w(p)=1$ for all $p \in \mathcal{P}$ is provided in the unweighted version. Our approximation algorithms uses a decomposition of the unit disk graph into strips.

The following lemma shows why determining the maximum independent set within a strip is useful for approximating the unrestricted problem. The proof is a simple application of the shifting strategy [17], paying attention to the fact that the width h must be a constant rational number for the proof to work (the running time depends on the denominator of h).

Lemma 3.1. *An exact algorithm for the maximum (weight) independent set for a set of m points inside a strip of rational width h with running time $T(m)$ yields an approxilation algorithm for maximum (weight) independent set of n points in arbitrary locations with approximation ratio $1+1/h$ and running time $O(T(n)+n\log n)$.*

Matsui [22] showed that a unit disk graph inside a strip of width $\sqrt{3}/2$ is a co-comparability graph, and therefore the maximum weight independent set can be determined exactly in time linear on the size of the input graph [19]. Since a complete graph

is a unit disk graph within an arbitrarily small strip, building the graph takes quadratic time in the worst case. The crucial property shown by Matsui and used by our algorithm is the following.

Lemma 3.2. *Let H be a strip of width $\sqrt{3}/2$. Consider $p_a, p_b, p_c \in H$ with $x(p_a) < x(p_b) < x(p_c)$. If $\|p_a p_b\| > 1$ and $\|p_b p_c\| > 1$, then $\|p_a p_c\| > 1$.*

To obtain a logarithmic improvement for the unweighted version, we use an additional property of unit disk graphs from [20]. The *left neighborhood* of a point p is the set of points q with $\|pq\| \leq 1$ and $x(q) \leq x(p)$.

Lemma 3.3. *The left neighborhood of a point p has at most 3 independent vertices.*

Let $\mathcal{Q} = \{p_1, p_2, \ldots, p_m\}$ be the set of points lying in a strip H with $p_1, p_2, \ldots, p_m$ in increasing order of their x-coordinates. Given $1 \leq i \leq m$, we define the set $S_i \subseteq \mathcal{Q}$ is as follows: (i) S_i is an independent set, (ii) p_i is the rightmost point of S_i, and (iii) S_i is a maximum weight set satisfying the previous two properties. Let W_i denote the sum of the weights of the points in S_i. For simplicity, the sets S_i can be viewed as x-monotone polygonal chains formed by connecting the points of S_i from left to right (see Figure 1). The objective of our algorithm is to extend these chains as long as possible. A chain containing maximum number of points is reported.

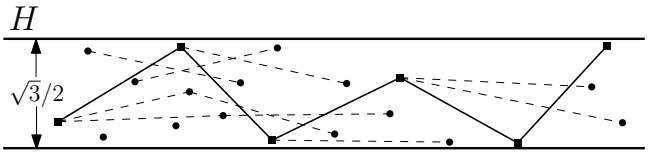

Figure 1: Pictorial representation of the sets S_i in the form of chains, with the maximum independent set marked.

Our algorithm uses dynamic programming to compute S_i for $i = 1, \ldots, m$ as follows. We calculate S_i using S_j for $j < i$ iteratively as $S_i = S_j \cup \{p_i\}$ where j is the index $\ell < i$ that satisfies $\|p_i p_\ell\| > 1$ and maximizes W_ℓ. In case there is no point p_ℓ, satisfying $\|p_i p_\ell\| > 1$ and $\ell < i$, the algorithm sets $S_i = \{p_i\}$. To efficiently determine the index j, we use the data structure from Lemma 2.6 (weighted) or 2.7 (unweighted). The pseudo-code is presented in Algorithm 1.

Theorem 3.4. *Algorithm 1 correctly computes a MWIS for the set $\mathcal{Q} = \{p_1, p_2, \ldots, p_m\}$ inside a strip of width $\sqrt{3}/2$ in $O(m \log^3 m)$ time using $O(m \log m)$ space. In the unweighted version, the complexities decrease to $O(m \log^2 m)$ time and $O(m)$ space.*

Proof. To see that S_i is calculated correctly by the procedure, notice that by Lemma 3.2, $\|p_i p_j\| > 1$ for $j < i$ implies that a point p_k to the left of p_j with $\|p_j p_k\| > 1$ satisfies $\|p_i p_k\| > 1$. By construction, S_i contains p_i and its weight is maximum.

To execute the algorithm in $O(m \log^3 m)$ time, we use the data structure for anti-disk maximum queries supporting insertions from Lemma 2.6 to store p_j for $j < i$. The weight of p_j in the data structure is W_j. The $O(m)$ insertions (line 10) and queries (line 7) take together $O(m \log^3 m)$ time, and the space is $O(m \log m)$ due to the storage of the data

Algorithm 1 Maximum (weight) independent set inside a strip of width $\sqrt{3}/2$

Input: The set $\mathcal{Q}$ of m points $p_1, \ldots, p_m$ ordered by x coordinate inside a strip of width $\sqrt{3}/2$.

Output: A maximum weight independent set of $\mathcal{Q}$.

D = data structure from Lemma 2.6 (weighted) or 2.7 (unweighted)
S = array $[1 \ldots m]$ of integers
W = array $[0 \ldots m]$ of reals (weighted) or integers (unweighted)
$W[0] = 0$, $max = 1$
for $i = 1, \ldots, m$ **do**
    A = complement of a disk of radius 1 centered at p_i
    $j =$ `maxQuery`(D, A) ▷ Returns 0 if no point is found
    $S[i] = j$ ▷ Represents $S_i = S_j \cup \{p_i\}$ with $S_0 = \emptyset$
    $W[i] = W[j] + w(p_i)$
    `insert`$(D, i, p_i, W[i])$ ▷ Insert point p_i with label i and weight $W[i]$ into D
    if $W[i] > W[max]$ **then**
        $max = i$
    end if
end for
$i = max$, $IS = \{\}$
while $i \neq 0$ **do**
    $IS = IS \cup \{p_i\}$
    $i = S[i]$
end while
return IS

structure. The algorithm stores S_i as a linked list from right to left. Part of the linked lists are shared by different sets S_i, forming a DAG.

To reduce the running time to $O(m \log^2 m)$ for the unweighted version, we use the data structure from Lemma 2.7. In the unweighted version, $W_i = |S_i|$ and therefore the weight W_i is an integer between 1 and m. Let max be the index of the point of maximum weight in the data structure at the time when a point of index j is returned by the query. We use Lemma 3.3 to show that $W_{max} - W_j \leq 3$. If $|S_{max}| \leq 3$, the statement follows trivially. Otherwise, let p_a, p_b, p_c, p_{max} be the four rightmost elements in S_{max} from left to right. We have $W_{max} = 1 + W_c = 2 + W_b = 3 + W_a$. By Lemma 3.3, p_a, p_b, p_c, p_{max} cannot be all adjacent to p_i, which shows that $w_{max} - w_{ret} \leq 3$. Therefore, using the data structure from Lemma 2.7, the $O(m)$ insertions and queries take together $O(m \log^2 m)$ time, and the space is $O(m)$. □

Combining Lemma 3.1 and Theorem 3.4, we obtain the following theorem.

Theorem 3.5. *Given a set $\mathcal{P}$ of n weighted points, we can compute an approximation to the maximum weight independent set in $O(n \log^3 n)$ time and $O(n \log n)$ space with approximation ratio $(1 + (2/\sqrt{3}) + \varepsilon) < 2.16$. In the unweighted version, the complexities decrease to $O(n \log^2 n)$ time and $O(n)$ space.*

4 Large Strips

In this section, we present a 2-approximation algorithm for the maximum (weight) independent set problem on unit disk graphs. The algorithm uses a decomposition of the region containing the disk centers into strips of width 1. Das et al. [14] proved the following lemma which is similar to Lemma 3.2 for these larger strips.

Lemma 4.1. *Let H be a strip of width 1. Consider $p_a, p_b, p_c, p_d \in H$ with $x(p_a) < x(p_b) < x(p_c) < x(p_d)$. If both $\{p_a, p_b, p_c\}$ and $\{p_b, p_c, p_d\}$ are independent sets, then $\|p_a p_d\| > 1$.*

Let $\mathcal{Q} = \{p_1, p_2, \ldots, p_m\}$ be the set of points lying in the strip H with $p_1, p_2, \ldots, p_m$ in increasing order of their x-coordinates. Given $1 \leq j < i \leq m$ with $\|p_i p_j\| > 1$, we define the set $S_{i,j} \subseteq \mathcal{Q}$ as follows: (i) $S_{i,j}$ is an independent set, (ii) p_j and p_i are the two rightmost points of $S_{i,j}$, and (iii) $S_{i,j}$ is a maximum weight set satisfying the previous two properties. Furthermore, we define $S_{i,0} = \{p_i\}$.

Let $W_{i,j}$ denote the sum of the weights of the points in $S_{i,j}$. Let $\mathcal{S}_i = \{S_{i,j} \mid 1 \leq j < i \text{ and } \|p_i p_j\| > 1\}$ denote the collections of sets $S_{i,j}$ for fixed i.

Our algorithm uses dynamic programming to compute $S_{i,j}$ for $i = 2, \ldots, m$ and $j = 1, \ldots, i-1$ with $\|p_i p_j\| > 1$ as follows. We calculate $S_{i,j}$ using $S_{j,k}$ for $k < j < i$ iteratively as $S_{i,j} = S_{j,k} \cup \{p_i\}$ where k is the index $\ell < j$ maximizing $W_{j,\ell}$ and satisfying the constraint that $\{p_i, p_j, p_\ell\}$ is an independent set. In case there is no point p_ℓ satisfying the previous constraint, the algorithm sets $S_{i,j} = \{p_i, p_j\}$. To efficiently determine the index $\ell < j$ that satisfies the previous constraint and maximizes W_ℓ we use an array of data structures $D_1, \ldots, D_m$ from Lemma 2.3 (weighted) or 2.4 (unweighted). The data structure D_i stores a point p_j with weight $W_{i,j}$ for each set $S_{i,j} \in \mathcal{S}_i$. Note that the data structure D_j only stores points p_k such that $\|p_j p_k\| > 1$. The pseudo-code is presented in Algorithm 2.

Theorem 4.2. *Algorithm 2 correctly computes a MWIS for the set $\mathcal{Q} = \{p_1, p_2, \ldots, p_m\}$ inside a strip of width 1 in $O(m^2 \log^2 m)$ time using $O(m^2 \log m)$ space. In the unweighted version, the complexities decrease to $O(m^2 \log m)$ time and $O(m^2)$ space.*

Proof. To see that $S_{i,j}$ calculated by the procedure is indeed an independent set, notice that by Lemma 4.1, $\{p_i, p_j, p_k\}$ be independent implies that a point p_s to the left of p_k with $\{p_j, p_k, p_s\}$ independent satisfies $\|p_i p_s\| > 1$. By construction, $S_{i,j}$ contains p_i, p_j as the points with maximum x coordinates and its weight is maximum.

To execute the algorithm in $O(m^2 \log^2 m)$ time, we use m data structures for anti-disk maximum queries from Lemma 2.3. For each point p_i, the data structure D_i stores the point p_j for each set $S_{i,j} \in \mathcal{S}_i$. The weight associated with p_j is $W_{i,j}$. The data structure D_i is static, preprocessed (line 26) with points collected in the set T. The time to preprocess all $O(m)$ data structures from the corresponding sets T is therefore $O(m) \cdot O(m \log^2 m)$. The point p_k of maximum weight satisfying $k < j$ and $\{p_i, p_j, p_k\}$ independent is determined with a single anti-disk maximum query (line 17) on the data structure D_j associated with $\mathcal{S}_j$. All the $O(m^2)$ queries take together $O(m^2) \cdot O(\log^2 m) = O(m^2 \log^2 m)$ time. Space is dominated by the $O(m)$ data structures with $O(m \log m)$ storage each.

To reduce the running time to $O(m^2 \log m)$ and the space to $O(m^2)$ for the unweighted version, we use the data structure from Lemma 2.4, and the same argument as in the proof of Theorem 3.4 shows that $w_{max} - w_{ret} \leq 3$. □

Algorithm 2 Maximum (weight) independent set inside a strip of width 1

Input: The set $\mathcal{Q}$ of m points $p_1, \ldots, p_m$ ordered by x coordinate inside a strip of width 1.

Output: A maximum weight independent set of $\mathcal{Q}$.

D = array $[1 \ldots m]$ of data structure from Lemma 2.3 (weighted) or 2.4 (unweighted)
S = array $[1 \ldots m, 0 \ldots m]$ of integers
W = array $[1 \ldots m, 0 \ldots m]$ of reals (weighted) or integers (unweighted)
$max = (1, 0)$
for $i = 1, \ldots, m$ **do**
    $S[i, 0] = 0$ ▷ Represents $S_{i,0} = \{p_i\}$
    $W[i, 0] = w(p_i)$
    if $W[i, 0] > W[max]$ **then**
        $max = (i, 0)$
    end if
end for
for $i = 2, \ldots, m$ **do**
    $T = \{\}$
    for $j = 1, \ldots, i - 1$ **do**
        if $\|p_i p_j\| > 1$ **then**
            A = complement of a disk of radius 1 centered at p_i
            $k =$ `maxQuery`$(D[j], A)$ ▷ Returns 0 if no point is found
            $S[i, j] = k$ ▷ Represents $S_{i,j} = S_{j,k} \cup \{p_i\}$
            $W[i, j] = W[j, k] + w(p_i)$
            $T = T \cup \{(j, p_j, W[i, j])\}$ ▷ (label, point, weight) for $D[i]$
            if $W[i, j] > W[max]$ **then**
                $max = (i, j)$
            end if
        end if
    end for
    $D[i] =$ `preprocess`(T) ▷ New data structure from set T
end for
$(i, j) = max$, $IS = \{p_i\}$
while $j \neq 0$ **do**
    $IS = IS \cup \{p_j\}$
    $i = j$, $j = S[i, j]$
end while
return IS

Combining Lemma 3.1 and Theorem 4.2, we obtain the following theorem.

Theorem 4.3. *Given a set $\mathcal{P}$ of n weighted points, we can compute a 2-approximation to the maximum weight independent set in $O(n^2 \log^2 n)$ time and $O(n^2 \log n)$ space. In the unweighted version, the complexities decrease to $O(n^2 \log n)$ time and $O(n^2)$ space.*

5 Polynomial-Time Approximation Scheme

We design a polynomial time approximation scheme (PTAS) for the maximum independent set problem on a given UDG, where the geometric representation of the graph has been given i.e., the center of the unit disks are given. We assume that $\mathcal{P}$ be the set of centers of the unit disks associated with the UDG. Also assume that $\mathcal{R}$ be an enclosing rectangle of the point set $\mathcal{P}$. To design a PTAS we use two level shifting strategy, proposed by Hauchbaum and Maass [17]. In the first level of shifting strategy we execute $k+1$ iterations as follows: in the i-th iteration ($0 \leq i \leq k$), we partition the region $\mathcal{R}$ into disjoint vertical slabs such that (i) the first slab is of width i starting from left, (ii) width of each even slab is 1, and (iii) width of other slab is k (note that width of last slab may be less than k). Therefore, solution of different slabs of width k are non-intersecting.

In an iteration of the first level, we consider only those vertical slabs containing at least one point in $\mathcal{P}$, and compute maximum independent set by applying second level shifting strategy by considering horizontal partition of each vertical slab, add up the solutions of all slabs to get the solution of that iteration. The iteration producing maximum size solution is reported. The proof of the following lemma is presented in [14].

Lemma 5.1. *If n_k is the maximum number of mutually non-overlapping unit disks whose centers lie in a strip of width $k > 1$ and intersected by a vertical line ℓ, then $n_k \leq \frac{7k}{3} + 2$.*

5.1 Computing MIS for unit disks centered in a $k \times k$ square

Let $Q \subseteq \mathcal{P}$ be the set of points inside a cell χ of size $k \times k$. Consider a vertical line ℓ_v and a horizontal line ℓ_h that partition χ into four sub-cells each of size $\frac{k}{2} \times \frac{k}{2}$. Let $Q(\ell_v, \ell_h) \subseteq Q$ be the set of points whose distance from ℓ_v or ℓ_h is at most $\frac{1}{2}$, and $Q_1, Q_2, Q_3, Q_4 \subseteq Q$ be the set of points in the four quadrants whose distance from ℓ_v and ℓ_h is greater than $\frac{1}{2}$. To compute a MIS for the set of points in Q, we use the following divide and conquer technique.

Consider all possible subsets $Q' \subseteq Q(\ell_v, \ell_h)$ of size at most $2 \times n_k$, where $n_k = \frac{7k}{3} + 2$ (since $2 \times n_k$ is the maximum possible size of the point set in $Q(\ell_v, \ell_h)$ that can appear in an optimal solution due to Lemma 5.1). For each of Q', we do the following in each quadrant: delete all the points in Q_i ($i = 1, 2, 3, 4$) which are not independent with Q'. Let $Q'_i \subseteq Q_i$ be the remaining set of points. Now compute the optimal solution for Q'_i recursively using the same procedure. If $T(m, k)$ is the time complexity for finding MIS in χ, then $T(m, k) = 4 \cdot T(m, \frac{k}{2}) \times m^{2n_k} = m^{O(k)}$. Thus, we have the following result:

Theorem 5.2. *Given a set $\mathcal{P}$ of n points in the plane and an integer $k > 1$, the proposed algorithm computes an independent set of size at least $\frac{1}{(1+\frac{1}{k})^2}|OPT|$ in $n^{O(k)}$ time, where $|OPT|$ is the optimal size of the solution.*

References

[1] P. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3):209–218, 1998.

[2] S. Arya, D. M. Mount, and J. Xia. Tight lower bounds for halfspace range searching. *Disc. Comput. Geom.*, 47(4):711–730, 2012.

[3] J. L. Bentley and J. B. Saxe. Decomposable searching problems I. static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[4] H. Brönnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Disc. Comput. Geom.*, 10:143–155, 1993.

[5] P. Carmi, G. K. Das, R. K. Jallu, S. C. Nandy, P. R. Prasad, and Y. Stein. Minimum dominating set problem for unit disks revisited. *Inter. J. Comput. Geom. Appl.*, 25:227–244, 2015.

[6] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.

[7] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proc. 17th Annu. ACM-SIAM Symp. Disc. Algorithms (SODA)*, pages 1196–1202, 2006.

[8] T. M. Chan. Optimal partition trees. *Disc. Comput. Geom.*, 47(4):661–690, 2012.

[9] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Disc. Comput. Geom.*, 48(2):373–392, 2012.

[10] X. Cheng, X. Huang, D. Li, W. Wu, and D.-Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42:202–208, 2003.

[11] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1–3):165–177, 1990.

[12] G. D. da Fonseca, C. M. H. de Figueiredo, V. G. Pereira de Sá, and R. C. S. Machado. Efficient sub-5 approximations for minimum dominating sets in unit disk graphs. *Theoret. Comput. Sci.*, 540–541:70–81, 2014.

[13] G. D. da Fonseca, V. G. P. de Sá, and C. M. H. de Figueiredo. Linear-time approximation algorithms for geometric intersection graphs. *arXiv preprint arXiv:1402.4722*, 2014.

[14] G. K. Das, M. De, S. Kolay, S. C. Nandy, and S. Sur-Kolay. Approximation algorithms for maximum independent set of a unit disk graph. *Inform. Process. Lett.*, 115(3):439–446, 2015.

[15] M. De, G. Das, P. Carmi, and S. Nandy. Approximation algorithms for a variant of disc. piercing set problem for unit disks. *Inter. J. Comput. Geom. Appl.*, 6(23):461–477, 2013.

[16] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. 2008.

[17] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image process. and VLSI. *J. ACM*, 32(1):130–136, 1985.

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

[19] E. Köhler and L. Mouatadid. A linear time algorithm to compute a maximum weighted independent set on cocomparability graphs. *Inform. Process. Lett.*, 2015.

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

[21] J. Matoušek. Reporting points in halfspaces. In *Proc. 32nd Annu. Symp. Found. Comput. Sci. (FOCS), 1991.*, pages 207–215, 1991.

[22] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proc. 2nd Japan Conf. Disc. Comput. Geom. (JCDCG)*, volume 1763 of *LNCS*, pages 194–200, 1998.

[23] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4):49:1–49:17, 2008.

[24] M. H. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Inform. Process. Lett.*, 12(4):168–173, 1981.

[25] E. J. van Leeuwen. Approximation algorithms for unit disk graphs. In *Graph-theoretic concepts in computer science*, pages 351–361, 2005.