Computational Geometry: Theory and Applications 109 (2023) 101925

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Experiments with unit disk cover algorithms for covering massive pointsets ☆

Rachel Friederich, Anirban Ghosh *, Matthew Graham, Brian Hicks, Ronald Shevchenko

School of Computing, University of North Florida, United States of America

ARTICLE INFO

Article history:
Received 16 April 2021
Received in revised form 10 July 2022
Accepted 8 August 2022
Available online 18 August 2022

Keywords:
Unit disk
Geometric covering
Experiments

ABSTRACT

Given a set of n points in the plane, the Unit Disk Cover (UDC) problem asks to compute the minimum number of unit disks required to cover the points, along with a placement of the disks. The problem is NP-hard and several approximation algorithms have been designed over the last three decades. In this paper, we have engineered and experimentally compared practical performances of some of these algorithms on massive pointsets. The goal is to investigate which algorithms run fast and give good approximation in practice.

We present a simple 7-approximation algorithm for UDC that runs in $O(n)$ expected time and uses $O(s)$ extra space, where s denotes the size of the generated cover. In our experiments, it turned out to be the speediest of all. We also present two heuristics to reduce the sizes of covers generated by it without slowing it down by much.

To our knowledge, this is the first work that experimentally compares algorithms for the UDC problem. Experiments with them using massive pointsets (in the order of millions) throw light on their practical uses. We share the engineered algorithms via GitHub[1] for broader uses and future research in the domain of geometric optimization.

1. Introduction

Geometric covering is a well-researched family of fascinating optimization problems in computational geometry and has been studied for decades. To date, research has been confined mostly to the theoretical arena only. Among these problems, the Unit Disk Cover (UDC) problem has turned out to be one of the fundamental covering problems. Given a set P of n points $p_1, \ldots, p_n$ in the Euclidean plane, the UDC problem asks to compute the minimum number of possibly intersecting unit disks (closed disks of unit radius) required to cover the points in P, along with a placement of the disks. See Fig. 1 for an example. Since the algorithms for UDC can be easily scaled for covering points using disks of any fixed radius $r > 0$, for the sake of brevity, we use $r = 1$.

☆ A preliminary version of this paper appeared in the Proceedings of the International Symposium on Experimental Algorithms, Springer, Cham, 2019 [18]. Research on this paper was fully supported by the University of North Florida Academic Technology Grant, and partially by the NSF Grant CCF-1947887.

* Corresponding author.
E-mail addresses: n01140328@unf.edu (R. Friederich), anirban.ghosh@unf.edu (A. Ghosh), n00612546@unf.edu (M. Graham), n00133251@unf.edu (B. Hicks), n01385011@unf.edu (R. Shevchenko).

[1] https://github.com/ghoshanirban/UnitDiskCoverAlgorithms

https://doi.org/10.1016/j.comgeo.2022.101925
0925-7721/

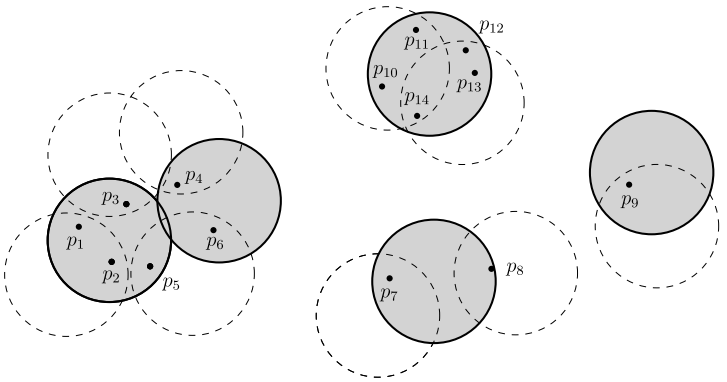


Fig. 1. Any optimal solution for the pointset $P = \{p_1, \ldots, p_{14}\}$ contains exactly 5 disks; an optimal solution for P is shown using gray disks. A non-optimal solution is shown using a set of 9 dashed disks.

The UDC problem has interesting applications in wireless networking, facility location, robotics, image processing, and machine learning. For instance, P can be perceived as a set of clients or locations of interest seeking service from service providers, which can be modeled using a set of fixed-radius disks. The goal is to provide service or cover these locations using the minimum number of service providers.

The UDC problem has a long history. Back in 1981, UDC was shown to be NP-hard by Fowler [15]. The first known approximation algorithm for UDC is a PTAS designed by Hochbaum and Maass [21] that runs in $O(\ell^4(2n)^{4\ell^2+1})$ time having an approximation factor of $(1+\frac{1}{\ell})^2$, for any integer $\ell \geq 1$. Gonzalez [19] presented two approximation algorithms; a $2(1+\frac{1}{\ell})$-approximation algorithm that runs in $O(\ell^2 n^7)$ time, where ℓ is a positive integer and another 8-approximation algorithm with a runtime of $O(n \log |\text{OPT}|)$, where $|\text{OPT}|$ is the number of disks in an optimal cover. Charikar, Chekuri, Feder, and Motwani [9] devised a 7-approximation algorithm for the UDC problem (the authors used the name Dual Clustering for this problem). A $O(1)$-approximation algorithm with a runtime of $O(n^3 \log n)$ is presented by Brönnimann and Goodrich [7]. Franceschetti, Cook, and Bruck [16] developed an algorithm with an approximation factor of $3(1+\frac{1}{\ell})^2$ having a runtime of $O(Kn)$, where ℓ is a positive integer and K is a constant that depends on ℓ. A 2.8334-approximation algorithm is designed by Fu, Chen, and Abdelguerfi [17] that runs in $O(n(\log n \log\log n)^2)$ time. Liu and Lu [25] designed a 25/4-approximation algorithm having a runtime of $O(n \log n)$. Biniaz, Liu, Maheshwari, and Smid [6] devised a 4-approximation algorithm that has a runtime of $O(n \log n)$. Recently, Dumitrescu, Ghosh, and Tóth [14] have designed an online 5-approximation[2] algorithm for the problem.

In the era of Big Data, the sizes of spatial data sets are growing exponentially. Thus, finding good quality solutions efficiently for NP-hard geometric optimization problems has posed a new challenge for algorithm engineers. In this regard, because of the practical importance of the UDC problem, we believe that it is worthwhile to investigate which algorithms designed for UDC are the best for processing massive pointsets in practice.

Covering problems involving points and disks are well-studied in computational geometry; see for instance, [2,3,5,10–13, 20,22,23,26]. Bus, Mustafa, and Ray designed a practical algorithm for the geometric hitting set problem; see [8]. The UDC problem has also been considered in the streaming setup by Liaw, Liu, and Reiss [24].

Our contributions For our experiments, we have implemented the following algorithms; appropriate abbreviations using the authors' names and dates of publication are used for naming purposes.

- G-1991 by Gonzalez (1991) [19]
- CCFM-1997 by Charikar, Chekuri, Feder, and Motwani (1997) [9]
- LL-2014 by Liu and Lu (2014) [25]
- BLMS-2017 by Biniaz, Liu, Maheshwari, and Smid (2017) [6]
- DGT-2018 by Dumitrescu, Ghosh, and Tóth (2018) [14]

We have refrained from implementing the algorithms from [7,16,17,21] since they are not practical and mainly of theoretical interest.

We present a simple 7-approximation algorithm named FastCover that runs in $O(n)$ expected time; see Section 2.6. In our experiments, we found FastCover to be the fastest of all. We also present two heuristics that effectively help to reduce the sizes of covers generated by it. FastCover with the first heuristic included is named FastCover+ and the one in which both the heuristics are included is named FastCover++. To our surprise, we found that in some cases FastCover++ could beat some of the sophisticated algorithms in speed and solution quality simultaneously. These three versions of FastCover

[2] In the literature of online algorithms, the term *competitive ratio* is used instead of approximation factor.

behave like three optimization levels for the algorithm, where FastCover being the fastest and FastCover++ the slowest in practice. Wherever possible, FastCover++ produces the smallest covers among these three.

In our experiments, we have used both synthetic and real-world massive pointsets. The largest pointset used in the experiments contains ≈ 10.8 million points. The algorithms are implemented in C++17 using the CGAL 5.3 library [27]. For broader uses of these algorithms, we share our code via GitHub.

In our knowledge, this is the first work that experimentally compares the existing algorithms for UDC. Experiments with them using massive pointsets throw light on their practical uses.

In Section 2, we discuss the algorithms implemented in this paper along with the FastCover algorithm. In Sections 3, we present our experimental results including tables and plots. In Section 4, we present our recommendations and conclusions.

Notations and terminology We denote a point $p \in \mathbf{R}^2$ using a pair of real numbers (a, b). By p_x and p_y, we denote its x and y-coordinates, respectively. A *unit ball* is a closed ball of unit radius in $\mathbf{R}^d$. In the plane, we use the term *unit disk*.

We define the *point density* of P as the ratio of its size to that of the area of its bounding box.

2. Algorithms engineered

In this section, we briefly describe the algorithms we have engineered and provide their pseudocodes along with their asymptotic runtimes. To see how the algorithms behave differently, refer to Section 6, where we present the covers generated by the algorithms engineered in this work when run on a 60-element pointset drawn randomly from a 20×20 square.

2.1. G-1991: Gonzalez (1991)

Gonzalez [19] presented two algorithms for UDC in d-space. One of these two algorithms is a PTAS that uses the shifting strategy introduced in [21]. This PTAS has an approximation factor of $2(1+\frac{1}{\ell})^{d-1}$ and runs in $O(\ell^{d-1}d(2\sqrt{d})(\ell\sqrt{d})^{d-1} \times n^{d(2\sqrt{d})^{d-1}+1})$ time, for every integer $\ell \geq 1$. In the plane, this algorithm has an approximation factor of $2(1+\frac{1}{\ell})$ and runs in $O(\ell^2 n^{4\sqrt{2}+1})$ time. We did not implement this algorithm due to its high asymptotic runtime.

Algorithm 1 : G-1991(P)

```
1: Disk-Centers ← ∅;
2: Partition P w.r.t. i_y(p) into sets P_1, ..., P_t;
3: for i ← 1 to t do
4:    Partition P_i w.r.t i_x(p) into sets S := S_1, ..., S_k;
5:    R ← S_1 ∪ S_2;
6:    j ← 2;
7:    while R ≠ ∅ do
8:       q ← min{p_x | p ∈ R};
9:       Let Q be the set of points in R at a distance ≤ √2 (w.r.t x only) from q;
10:      R ← R \ Q;
11:      Let c be the center of the unit-disk that covers the √2 × √2 square whose left boundary includes q and whose top boundary coincides with the
         top boundary of the slab having height √2 that contains q;
12:      Disk-Centers ← Disk-Centers ∪ {c};
13:      while j < k and R contains elements from at most one of the sets in S do
14:         j ← j + 1;
15:         R ← R ∪ S_j;
16:      end while
17:   end while
18: end for
19: return Disk-Centers;
```

The other algorithm G-1991, as we call it, has an approximation factor of $2^{d-1}(\lceil\sqrt{d}\rceil)^d$ and runtime of $O(dn + n\log|\mathrm{OPT}|)$, where $|\mathrm{OPT}|$ is the number of disks in an optimal cover. In the plane, G-1991 gives 8-approximation and runs in $O(n\log|\mathrm{OPT}|)$ time. See Algorithm 1 for a high-level description of G-1991. We use the following notations in the algorithm. Let $p \in P$, then $i_x(p) = \lfloor p_x/\sqrt{2} \rfloor$ and $i_y(p) = \lfloor p_y/\sqrt{2} \rfloor$.

The author presented this algorithm for covering points using axis-parallel squares of fixed size and claimed that the same can be used for UDC. In our implementation, we have used squares of length $\sqrt{2}$ to cover the points and then placed a unit disk at the center of every square. Since a square of length $\sqrt{2}$ can be inscribed inside a unit disk, every point in the input is covered using this approach.

2.2. CCFM-1997: Charikar, Chekuri, Feder, and Motwani (1997)

The algorithm CCFM-1991 by Charikar et al. [9] was originally designed for the online version of UDC. The authors used name Dual Clustering in their paper. In d-space, CCFM-1991 gives an approximation factor of $O(2^d d\log d)$. In 2-space, CCFM-1997 has an approximation factor of 7. Refer to Algorithm 2 for a high-level description of CCFM-1991. No comment was made about its runtime or implementation.

Algorithm 2 : CCFM-1997(P)

Let `active-centers` and `inactive-centers` be two empty sets;
for $p \in P$ **do**
  if the distance to the nearest disk center in `active-centers` > 1 **then**
    if `inactive-centers` is empty **then**
      Add p to `active-centers` and add the following six points to `inactive-centers`: $(p_x+\sqrt{3}, p_y)$, $(p_x+\sqrt{3}/2, p_y+1.5)$, $(p_x+\sqrt{3}/2, p_y-1.5)$, $(p_x-\sqrt{3}/2, p_y+1.5)$, $(p_x-\sqrt{3}, p_y)$, $(p_x-\sqrt{3}/2, p_y-1.5)$;
      continue;
    end if
    if the distance to the nearest disk center q in `inactive-centers` ≤ 1 **then**
      Delete q from `inactive-centers` and add q to `active-centers`;
    else
      Add p to `active-centers` and add the following six points to `inactive-centers`: $(p_x+\sqrt{3}, p_y)$, $(p_x+\sqrt{3}/2, p_y+1.5)$, $(p_x+\sqrt{3}/2, p_y-1.5)$, $(p_x-\sqrt{3}/2, p_y+1.5)$, $(p_x-\sqrt{3}, p_y)$, $(p_x-\sqrt{3}/2, p_y-1.5)$;
    end if
  end if
end for
return `active-centers`;

2.3. LL-2014 and LL-2014-1P: Liu and Lu (2014)

In LL-2014 [25], the plane is divided into vertical strips of width $\sqrt{3}$ each. Inside each strip, we obtain an approximate solution by sorting the points in non-increasing order according to their y-coordinates. The next uncovered point inside a strip is covered by placing a disk as low as possible. These disks are placed by centering them on the vertical line that splits the strip into two. The final solution is constructed by taking the union of all the solutions obtained for the strips. This strip system is shifted five times to the right by a distance of $\sqrt{3}/6$ every time. For every shift, we obtain a solution as described above along with one solution for the initial strip configuration. The algorithm returns the best one (having the least number of disks) out of these six solutions. Refer to Algorithm 3 for an algorithmic description of LL-2014. The authors show that LL-2014 has an approximation factor of $25/6 \approx 4.17$ and runs in $O(n \log n)$ time.

Algorithm 3 : LL-2014(P)

DISK-CENTERS $\leftarrow \emptyset$;
`min` $\leftarrow n+1$;
Sort P w.r.t x-coordinate in $O(n \log n)$ time;
for $i \in \{0, 1, 2, 3, 4, 5\}$ **do**
  `current` $\leftarrow 1$, $C \leftarrow \emptyset$, `right` $\leftarrow P[1]_x + \frac{i\sqrt{3}}{6}$;
  while `current` $\leq n$ **do**
    `index` $\leftarrow$ `current`;
    while P[`current`]$_x <$ right **and** `current` $\leq n$ **do**
      `current` $\leftarrow$ `current` $+ 1$;
    end while
    x-`of-restriction-line` $\leftarrow$ `right` $-\sqrt{3}/2$, `segments` $\leftarrow \emptyset$;
    for $j \leftarrow$ `index` **to** `current`-1 **do**
      $d \leftarrow P[j]_x -$ x-`of-restriction-line`, $y \leftarrow \sqrt{1-d^2}$;
      Create a segment s having the endpoints (x-`of-restriction-line`, $P[j]_y + y$) and (x-`of-restriction-line`, $P[j]_y - y$) and insert it into `segments`;
    end for
    Sort `segments` in non-ascending order based on y-coordinates of their tops. Greedily stab them by choosing the stabbing point as low as possible, while still stabbing the topmost unstabbed segment. Put the stabbing points (the disk centers) in C;
    Increment `right` by a multiple of $\sqrt{3}$ such that P[`current`] $-$ `right` $\leq \sqrt{3}$;
  end while
  if $|C| <$ `min` **then**
    DISK-CENTERS $\leftarrow C$,
    `min` $\leftarrow |C|$;
  end if
end for
return DISK-CENTERS;

In our experiments, we also consider a one-pass version of this algorithm since in practice we find there is barely any advantage of using six passes instead of one. The authors have used six passes to reduce the approximation factor from 5 to 25/6. We named this one-pass version LL-2014-1P. Obviously, this one-pass version is much faster in practice too. Since there is no substantial difference between LL-2014 and LL-2014-1P, we do not present separate pseudocode for this one-pass version.

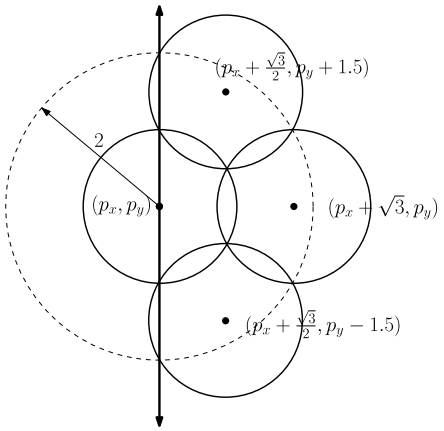


Fig. 2. The four disks placed by BLMS-2017 when p is processed. The figure illustrates the situation when the nearest point in C is more than 2 units away from p.

2.4. BLMS-2017: Biniaz, Liu, Maheshwari, and Smid (2017)

The algorithm BLMS-2017 by Biniaz et al. [6] gives 4-approximation and runs in $O(n \log n)$ time. Refer to Algorithm 4 for a high-level description of the algorithm.

Although the algorithm has a low approximation factor, placing four disks in advance sometimes introduces empty disks in the solutions. For instance, if the distance between any two points in P is greater than 2, BLMS-2017 places exactly 4 times the optimal number of disks (in this case, $|\text{OPT}| = n$). In comparison, other algorithms such as G-1991 or DGT-2018 place an optimal number of disks. However, in our implementation, we have managed to eliminate such empty disks placed by this algorithm. BLMS-2017 uses the popular sweep line technique in computational geometry by carefully maintaining a binary search tree of disk centers. For further details, we refer the reader to the original paper [6].

Algorithm 4 : BLMS-2017(P)

```
1: Disk-Centers ← ∅;
2: C ← ∅,
3: Sort P from left to right in O(n log n) time;
4: for p ∈ P do
5:     if the nearest point in C is more than 2 units away from p then
6:         Place four disks centered at (p_x, p_y), (p_x + √3, p_y), (p_x + √3/2, p_y + 1.5), (p_x + √3/2, p_y − 1.5) as shown in Fig. 2 and add these four points to Disk-Centers;
7:         C ← C ∪ {p};
8:     end if
9: end for
10: return Disk-Centers;
```

2.5. DGT-2018: Dumitrescu, Ghosh, and Tóth (2018)

DGT-2018 [14][3] is a simple online algorithm that gives 5-approximation in the plane. In d-space, the algorithm has an approximation factor of $O(1.321^d)$ which is an improvement over CCFM-1997 (an online algorithm). Refer to Algorithm 5 for a high-level description of this algorithm.

Algorithm 5 : DGT-2018(P)

```
1: Disk-Centers ← ∅;
2: for p ∈ P do
3:     if the distance from p to the nearest point in Disk-Centers is > 1 then
4:         Disk-Centers ← Disk-Centers ∪ {p};
5:     end if
6: end for
7: return Disk-Centers;
```

[3] A preliminary version of the paper appeared in the proceedings of the 12th Annual International Conference on Combinatorial Optimization and Applications, 2018 (COCOA 2018), and as such we use the same year in the algorithm abbreviation.

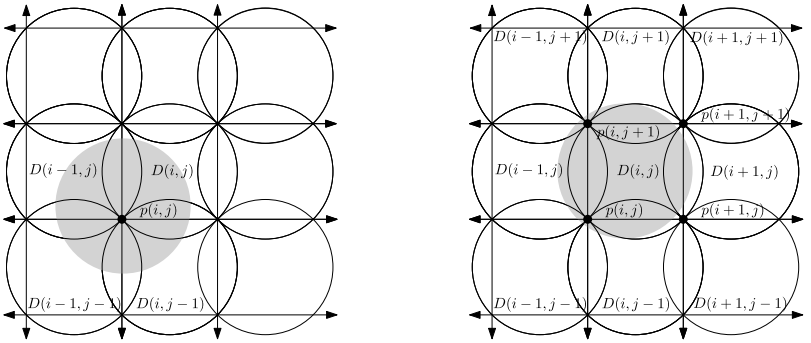


Fig. 3. D is shown in gray. Left: D contains exactly one grid point. Right: D contains exactly two grid points.

2.6. FastCover, FastCover+, and FastCover++

In this section, we present a simple 7-approximation algorithm FastCover that runs in $O(n)$ expected time. We use a $\sqrt{2}$-sized square grid Γ. A cell in Γ is denoted by $\sigma(i, j)$ where $i, j \in \mathbf{Z}$. Formally, the cell $\sigma(i, j)$ is the intersection of the four half-planes: $x \geq \sqrt{2}i, x < \sqrt{2}(i+1), y \geq \sqrt{2}j, y < \sqrt{2}(j+1)$. For every cell $\sigma(i, j) \in \Gamma$, there exists a unit disk $D(i, j)$ that circumscribes $\sigma(i, j)$. We say that $D(i, j)$ is the *grid-disk* of $\sigma(i, j)$. Clearly, $D(i, j)$ is unique.

The points in the input P are considered sequentially without any kind of pre-processing. Let $\sigma(i, j)$ be the cell to which the current point p belongs, for some $i, j \in \mathbf{Z}$. Place $D(i, j)$ if not placed previously.

Since every grid-disk can be represented using a pair of integers, we use a hash-table $\mathcal{H}$ of integer-pairs for storing the placed grid-disks. The main motivation of using a hash-table is fast lookups and insertions in practice. The cell in which p lies is $\sigma(i, j)$ where $i = \lfloor p_x/\sqrt{2} \rfloor$ and $j = \lfloor p_y/\sqrt{2} \rfloor$. The center of the grid-disk $D(i, j)$ is located at $(\sqrt{2}i + \frac{1}{\sqrt{2}}, \sqrt{2}j + \frac{1}{\sqrt{2}})$. See Algorithm 6 for a pseudocode.

Algorithm 6 : FastCover(P)

```
1: H ← ∅; Disk-Centers ← ∅;
2: for p ∈ P do
3:     i ← ⌊p_x/√2⌋; j ← ⌊p_y/√2⌋;
4:     if (i, j) ∉ H then
5:         insert (i, j) into H and (√2 i + 1/√2, √2 j + 1/√2) into Disk-Centers;
6:     end if
7: end for
8: return Disk-Centers;
```

In the following, we show that FastCover is a 7-approximation algorithm for UDC.

Theorem 1. FastCover *is a 7-approximation algorithm for the unit disk cover problem that runs in $O(n)$ expected time using $O(s)$ extra space where s is the size of the solution generated by it. Furthermore, for every integer $n \geq 1$, there exists an $7n$-element pointset for which* FastCover *places seven times the optimal number of disks.*

Proof. The union of all grid-disks in the plane gives $\mathbf{R}^2$. Hence, it is enough to consider the grid-disks for covering the points in P.

Given $i, j \in \mathbf{Z}$, we denote the intersection point of the two lines $x = \sqrt{2}i$ and $y = \sqrt{2}j$ by $p(i, j)$. We refer to these intersection points as *grid points*. Since the distance between any two grid points is at least $\sqrt{2}$, any unit disk D can contain at most four grid points in $\{p(i, j), p(i+1, j), p(i, j+1), p(i+1, j+1)\}$, for some $i, j \in \mathbf{Z}$. Now observe that D cannot contain exactly three grid points since in that case, D would circumscribe $\sigma(i, j)$ and consequently, D would contain the four grid points $\{p(i, j), p(i+1, j), p(i, j+1), p(i+1, j+1)\}$.

Consider any disk D from an optimal solution that covers P. It suffices to show that to cover the points $D \cap P$, FastCover places at most seven grid-disks. We show this using a proof by cases on the number of grid points contained by D.

Assume that D contains exactly one grid point $p(i, j)$, for some $i, j \in \mathbf{Z}$. See Fig. 3(left). In this case, D intersects the four grid-disks $D(i, j), D(i-1, j), D(i-1, j-1), D(i, j-1)$ only and as a result, FastCover places at most four of these grid-disks to cover the points $D \cap P$.

Now assume that D contains exactly two grid points. Without loss of any generality, let the two grid points be $p(i, j)$ and $p(i, j+1)$, for some $i, j \in \mathbf{Z}$. Furthermore, we safely assume that the center of D is in $\sigma(i, j)$. See Fig. 3(right). The case where its center is in $\sigma(i-1, j)$ is analogous and thus omitted. Let $\mathcal{D} := \{D(s, t) : (s, t) \in \{i-1, i, i+1\} \times \{j-1, j, j+1\}\}$. Since the two grid points $p(i+1, j)$ and $p(i+1, j+1)$ are not in D, it follows that $D \cap D(i+1, j+1) = \emptyset$ and $D \cap D(i+1, j-$

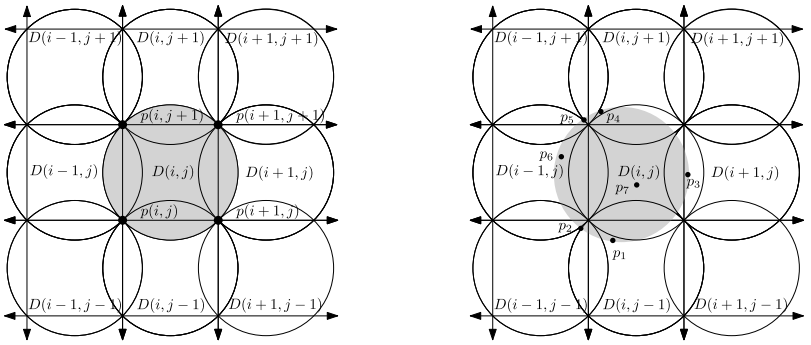


Fig. 4. Left: D contains exactly four grid points. Right: A sequence of seven points $p_1, \ldots, p_7$ for which FastCover places exactly seven disks but they can covered optimally using a single unit disk.

$1) = \emptyset$. Leaving aside these two disks, D can intersect with at most seven grid-disks in $\mathcal{D} \setminus \{D(i+1, j+1), D(i+1, j-1)\}$. As a result, in this case, FastCover will place at most seven disks to cover the points $D \cap P$.

In the final case, we assume that D contains exactly four grid points $p(i, j), p(i+1, j), p(i+1, j+1), p(i, j+1)$, for some $i, j \in \mathbf{Z}$. See Fig. 4(left). Clearly, in this case, D itself is a grid-disk and $D = D(i, j)$. It is enough to consider the nine disks $\mathcal{D} := \{D(s, t) : (s, t) \in \{i-1, i, i+1\} \times \{j-1, j, j+1\}\}$ in this case. Among these nine disks, $D(i-1, j-1)$ will not be used just to cover a point in $D \cap P$ since in that case the only possible point $p(i, j) \in D \cap P$ also belongs to $\sigma(i, j)$. Clearly, in this case the disk $D(i, j)$ will be placed to cover the point. Similarly, the disks $D(i+1, j-1)$ and $D(i-1, j+1)$ will not be placed just to cover $p(i+1, j-1)$ and $p(i, j+1)$, respectively. This rules out three disks in $\mathcal{D}$. Thus, in this case, FastCover will place at most six disks to cover the points in $D \cap P$.

Since FastCover will at most 7 disks to cover the points $D \cap P$, we conclude that it gives 7-approximation.

For every point in P, we perform exactly one look-up and at most one insertion in $\mathcal{H}$, each taking $O(1)$ expected time; refer to Algorithm 8. Hence, FastCover runs in $O(n)$ expected time. Note that we insert a disk into $\mathcal{H}$ only when it belongs to the solution generated by the algorithm. This implies FastCover needs additional $O(s)$ space, where s denotes the number of disks placed by the algorithm.

Next, we present a sequence of points $Q = \{p_1, \ldots, p_7\}$ for which FastCover places exactly seven disks but an optimal algorithm will place exactly one disk to cover them. Refer to Fig. 4(right). Our algorithm places the seven disks: $D(i, j-1), D(i-1, j-1), D(i+1, j), D(i, j+1), D(i-1, j+1), D(i-1, j), D(i, j)$.

Given an integer $n \geq 1$, consider n copies of Q and place them sufficiently apart. Let these n copies be $Q_1, \ldots, Q_n$, where $Q_1 = Q$. For $2 \leq i \leq n$, Q_i is obtained from the Q_{i-1} by adding 3 to the x-coordinate of every point in Q_{i-1}. In this case, the $7n$ points in $\cup_{i=1}^{n} Q_i$ can be covered optimally using n disks, but Fast-Cover will place exactly $7n$ disks to cover the n points. □

Remark. If the bounding-box of P is known in advance and sufficient space is available, FastCover can be implemented to run in $O(n)$ worst-case time using a matrix for storing the disk centers. In our experiments, we have assumed that the bounding box is unknown.

Next, we present two heuristics which are capable of improving the solutions computed by FastCover by decreasing the number of disks placed while still covering P.

Heuristic 1. We observe that if $p \in \sigma(i, j)$, then p may also be covered by one of the four adjacent grid-disks $N := D(i, j+1), S := D(i, j-1), E := D(i+1, j), W := D(i-1, j)$; refer to Fig. 5. So, if $D(i, j)$ is already placed before, we do not take any action. Otherwise, we check if p is covered by any one of the above four neighboring disks placed before. If not, we place $D(i, j)$. We find this simple heuristic to be effective in reducing cover sizes in practice.

Now consider the axis-parallel square $\alpha(i, j)$ that lies inside $\sigma(i, j)$ and touches the four grid-disks N, S, E, W. If p lies in the interior of $\alpha(i, j)$, one can safely conclude that p is not covered by any of the four disks N, S, E, W. Note that this simple checking does not require any distance calculation.

Let d be the distance between $\alpha(i, j)$ and the boundary of $\sigma(i, j)$. Observe that $d = |BC| = |AB| = |AO| - |BO| = 1 - (\sqrt{2}/2)$.

Before verifying whether $p \in E$ using a distance calculation, we first check if $p_x \geq \sqrt{2}(i+1.5) - 1$ since the right boundary of $\alpha(i, j)$ has the x-coordinate $\sqrt{2}(i+1) - d = \sqrt{2}(i+1) - (1 - \sqrt{2}/2) = \sqrt{2}(i+1.5) - 1$. If not, we can safely conclude that $p \notin E$. Similarly, before checking whether $p \in N$, we first verify if $p_y \geq \sqrt{2}(j+1.5) - 1$. For the disks W and S, we use the conditions $p_x \leq \sqrt{2}(i-0.5) + 1$ and $p_y \leq \sqrt{2}(j-0.5) + 1$, respectively. These comparisons along with at most four checks to verify if p is covered by one of the four neighboring disks help in reducing the number of disks placed in practice without slowing down the algorithm by much.

We propose an improved version of FastCover named FastCover+ that includes the Heuristic 1. Refer to Algorithm 7 for a pseudocode of FastCover+.

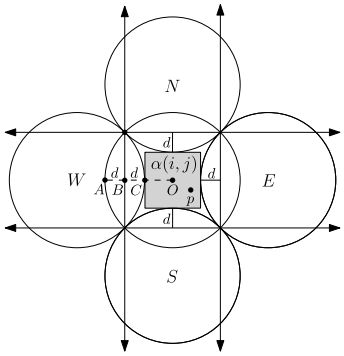


Fig. 5. If $p \in \alpha(i, j)$ (shown in gray), then p is not covered by any disk in $\{N, S, E, W\}$; $d = 1 - \frac{\sqrt{2}}{2}$.

Algorithm 7 : FastCover+(P)

```
1: H ← ∅; Disk-Centers ← ∅;
2: for p ∈ P do
3:     i ← ⌊p_x/√2⌋; j ← ⌊p_y/√2⌋;
4:     if (i, j) ∈ H then
5:         update B(i, j) using p; {p is already covered by D(i, j)}
6:     else if p_x ≥ √2(i + 1.5) − 1 and (i + 1, j) ∈ H and
           distance(p, (√2(i + 1) + 1/√2, √2 j + 1/√2)) ≤ 1 then
7:         continue; {p is covered by the grid-disk E placed before}
8:     else if p_x ≤ √2(i − 0.5) + 1 and (i − 1, j) ∈ H and
           distance(p, (√2(i − 1) + 1/√2, √2 j + 1/√2)) ≤ 1 then
9:         continue; {p is covered by the grid-disk W placed before}
10:    else if p_y ≥ √2(j + 1.5) − 1 and (i, j + 1) ∈ H and
           distance(p, (√2 i + 1/√2, √2(j + 1) + 1/√2)) ≤ 1 then
11:        continue; {p is covered by the grid-disk N placed before}
12:    else if p_y ≤ √2(j − 0.5) + 1 and (i, j − 1) ∈ H and
           distance(p, (√2 i + 1/√2, √2(j − 1) + 1/√2)) ≤ 1 then
13:        continue; {p is covered by the grid-disk S placed before}
14:    else
15:        insert (i, j) into H and (√2 i + 1/√2, √2 j + 1/√2) into Disk-Centers;
16:    end if
17: end for
18: return Disk-Centers;
```

Heuristic 2. This heuristic tries to lower down the number of disks placed by FastCover or FastCover+ using a coalescing technique. If the points covered by two adjacent grid-disks $D(i, j), D(k, \ell)$ where $|i - k| \leq 1, |j - \ell| \leq 1$, can be covered by a single disk D, then we eliminate $D(i, j), D(k, \ell)$ from the solution and include D instead. The pairs of grid-disks are chosen arbitrarily for coalescing. Now, the question remains to be answered is, how to confirm the existence of such a disk D. Certainly, one way is to find the minimum enclosing disk D of the points covered by the disks $D(i, j), D(k, \ell)$ and check if its radius is at most a unit. But, every algorithm that finds a minimum enclosing disk of an n-element pointset runs in $\Omega(n)$ time.

We propose a constant-time approximate method using bounding-boxes. For every grid-disk $D(i, j)$ placed, we maintain a bounding-box $B(i, j)$ of the points covered by $D(i, j)$. Now, for every point $p \in P$, a grid-disk $D(i, j)$ is always found that covers p. If $p \in B(i, j)$, then $B(i, j)$ remains unaltered. Otherwise, $B(i, j)$ is updated to include p in it. This update can be done in $O(1)$ time.

The respective bounding-boxes $B(i, j), B(k, \ell)$ of the disks $D(i, j), D(k, \ell)$ can be used to determine in $O(1)$ time if they can be coalesced into one unit disk D. We compute the bounding-box B by taking the union of $B(i, j)$ and $B(k, \ell)$. This union can be easily computed in $O(1)$ time by considering the maximum and minimum x, y-coordinates of the two bounding-boxes.

If the diagonal of B has length at most 2, we report that there is a disk D that covers the points in $P \cap (D(i, j) \cup D(k, \ell))$. We use the center of B as the center of D. Refer to Fig. 6 for an illustration.

In this heuristic, every grid-disk is considered at most eight times. Since searching and deletion in $\mathcal{H}$ takes $O(1)$ expected time, this heuristic can be implemented to run in $O(n)$ expected time.

Now we present another improved version of FastCover named FastCover++ that includes both Heuristics 1 and 2. See Algorithm 8 for a pseudocode of FastCover++.

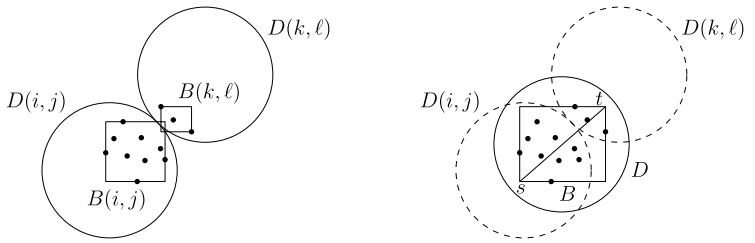


Fig. 6. Left: The case is shown where $k = i+1, \ell = j+1$. Right: The bounding-boxes $B(i,j), B(k,\ell)$ are merged to form B. In this case, the diagonal-length of B is $|st| \leq 2$ and hence the unit disk D, centered at the midpoint of st, is used to cover the points in B. As a result, the disks $D(i,j), D(k,\ell)$ are eliminated from the final solution and D is included instead.

Algorithm 8 : FastCover++(P)

$\mathcal{H} \leftarrow \emptyset$; Disk-Centers $\leftarrow \emptyset$;
for $p \in P$ **do**
  $i \leftarrow \lfloor p_x/\sqrt{2} \rfloor$; $j \leftarrow \lfloor p_y/\sqrt{2} \rfloor$;
  if $(i,j) \in \mathcal{H}$ **then**
    update $B(i,j)$ using p; {p is already covered by $D(i,j)$}
  else if $p_x \geq \sqrt{2}(i+1.5) - 1$ **and** $(i+1,j) \in \mathcal{H}$ **and** distance$(p, (\sqrt{2}(i+1) + \frac{1}{\sqrt{2}}, \sqrt{2}j + \frac{1}{\sqrt{2}})) \leq 1$ **then**
    update $B(i+1,j)$ using p; {p is covered by the grid-disk E placed before}
  else if $p_x \leq \sqrt{2}(i-0.5) + 1$ **and** $(i-1,j) \in \mathcal{H}$ **and** distance$(p, (\sqrt{2}(i-1) + \frac{1}{\sqrt{2}}, \sqrt{2}j + \frac{1}{\sqrt{2}})) \leq 1$ **then**
    update $B(i-1,j)$ using p; {p is covered by the grid-disk W placed before}
  else if $p_y \geq \sqrt{2}(j+1.5) - 1$ **and** $(i,j+1) \in \mathcal{H}$ **and** distance$(p, (\sqrt{2}i + \frac{1}{\sqrt{2}}, \sqrt{2}(j+1) + \frac{1}{\sqrt{2}})) \leq 1$ **then**
    update $B(i,j+1)$ using p; {p is covered by the grid-disk N placed before}
  else if $p_y \leq \sqrt{2}(j-0.5) + 1$ **and** $(i,j-1) \in \mathcal{H}$ **and** distance$(p, (\sqrt{2}i + \frac{1}{\sqrt{2}}, \sqrt{2}(j-1) + \frac{1}{\sqrt{2}})) \leq 1$ **then**
    update $B(i,j-1)$ using p; {p is covered by the grid-disk S placed before}
  else
    insert (i,j) into $\mathcal{H}$ and initialize $B(i,j)$ using p;
  end if
end for
while there is a grid-disk $(i,j) \in \mathcal{H}$ that is not considered yet **do**
  if there is a grid disk $(k,\ell) \in \mathcal{H}$ such that $|i-k| \leq 1, |j-\ell| \leq 1$ and the diagonal-length of the bounding-box $B := B(i,j) \cup B(k,\ell)$ is at most 2 **then**
    remove (i,j) and (k,ℓ) from $\mathcal{H}$ and add the center of B to Disk-Centers;
  end if
end while
for every grid-disk $(i,j) \in \mathcal{H}$ **do**
  insert $(\sqrt{2}i + \frac{1}{\sqrt{2}}, \sqrt{2}j + \frac{1}{\sqrt{2}})$ into Disk-Centers;
end for
return Disk-Centers;

3. Engineering and experiments

In this section, we present our experimental results. The algorithms have been implemented in GNU C++17 using the CGAL library [27]. The machine used for our experiments is equipped with a Ryzen 5 1600 (3.2 GHz) processor, 24 GB of main memory, and runs Ubuntu Linux 20.04 LTS. During compilation, the `g++` compiler was invoked with `-O3` optimization flag for fastest real-world execution times.

Implementation details For better real-world performance, containers and functions from the C++ STL (Standard Template Library) are used wherever needed. The `CGAL::Cartesian<double>` from CGAL is used for geometric computations. We have tried our best to tune our codes to run fast. For instance, constant expressions used in the code have been pre-calculated and stored in variables to avoid repeated calculations. For measuring time, we have used `std::chrono::high_resolution_clock`. Wherever needed, for computing distance between two points we have used the `CGAL::squared_distance` function.

- G-1991: Instead of partitioning the horizontal strips into two groups as stated in the algorithm, we have used one group and then processed the strips sequentially for faster speed. Since ordering is needed, `std::map` is used for storing the strips along with the points.
- CCFM-1997: As evident from its pseudocode, this algorithm needs nearest neighbor searching for every point. Although CGAL has support for such queries, we have used a grid-based approach akin to FastCover for speed. We maintain the non-empty grid cells using a `std::unordered_map` with `boost::hash<std::pair<int,int>>` as the hash

function for fast real-world speed. For every cell, we maintain a list of disk centers inside it. In this way the nearest neighbor query for the current point under consideration can be executed fast since inside the cell in which the point lies, there can be a constant number of disk centers placed by CCFM-1997. The same observation holds for the neighboring cells. For every point, we need to probe into at most nine cells and consequently, nearest neighbor queries implemented in this fashion run very fast for this algorithm.

- LL-2014 and LL-2014-1P: We have used `std::sort` for sorting. Recall that LL-2014 uses six passes for computing covers. In our experiments, we find that there is barely any difference in cover sizes when one pass is used as opposed to six passes. Obviously, the one-pass version is much faster too. We named this one pass version LL-2014-1P.
- BLMS-2017: As stated in Section 2, this algorithm may place empty disks sometimes. In order to eliminate such disks from the solution, we keep track of the empty disks. As proposed by the authors of BLMS-2017, we have used a binary search tree to implement this algorithm. This tree is implemented using `std::set`.
- DGT-2018: Similar to CCFM-1997, this algorithm is also dependent on nearest neighbor queries. As such we have used the grid-based setup as used in CCFM-1997.
- FastCover and FastCover+: We have used `std::unordered_set` to implement the required hash-table for this algorithm with `boost::hash<std::pair<int,int>>` as the hash function in order to maintain the set of placed grid-disks.
- FastCover++: In this variation of FastCover we have used `std::unordered_map` with `boost::hash<std::pair<int,int>>` as the hash function to maintain the set of placed grid-disks and their corresponding bounding-boxes.

In our experiments, we have used both synthetic and real-world pointsets. The execution times of most UDC algorithms vastly depend on the density of pointsets. Clearly, covers of high-density pointsets always have small sizes. Interestingly, this enhances the speed of most UDC algorithms since they consult the set of disks already placed to verify if the current point under consideration is already covered by at least one of those disks. This motivates us to use pointsets having varied densities.

Synthetic pointsets For generating synthetic pointsets, we have used the following three random pointset generators: `CGAL::Random_points_in_square_2` (generates pointsets inside a square), `CGAL::Random_points_in_disc_2` (generates pointsets inside a disk), and `CGAL::random_convex_set_2` (generates convex pointsets inside a square). Besides, we also have used pointsets drawn from an annulus. For generating points inside annulus, we have used `std::default_random_engine` and `std::uniform_real_distribution<double>` to generate point coordinates. In our experiments, the sizes of the pointsets are in the order of millions. They range from 1 to 10 million. For every value of n, we have drawn 5 samples for our experiments. The reported times and cover sizes are averaged over five samples. The plots and tables for synthetic pointsets have been consolidated and moved to Section 5 for an easy reference.

- **Points drawn from a square.** For this class of pointsets, we have used four bounding boxes having areas $10^7, 10^6, 10^5, 10^4$ to have pointsets of varying densities. Refer to the Figs. 9, 10, 11, and 12. In Fig. 9, the point densities are in the range $0.1, 0.2, \ldots, 1$; in Fig. 10, they are in the range $1, 2, \ldots, 10$; in Fig. 11, the range is $10, 20, \ldots, 100$, and in Fig. 12, the range is $100, 200, \ldots, 1000$. Thus, our pointsets have densities that vary from as low as 0.1 to as high as 1000.
  As evident from Figs. 9, 10, 11, and 12, LL-2014 turned out to be the slowest of all. The main reason for this slowdown is the number of passes it makes. It makes 6 passes to bring down the approximation factor to 25/6 from 5. However, we find that 6 passes are not that helpful in reducing the number of disks placed. Indeed, as can be seen in our experimental data, there is a negligible difference between LL-2014 and LL-2014-1P (the one-pass version of LL-2014) when it comes to cover size. On the other hand, in terms of speed, LL-2014-1P is much faster than LL-2014 since it is making just one pass. LL-2014-1P did very well in generating low-sized covers. In fact, its performance is one of the best overall in this regard. In Fig. 10, we find it is way ahead of others in placing fewer disks. In terms of speed, LL-2014-1P remained very competitive everywhere.
  BLMS-2017 and CCFM-1997 are some of the slowest in this case. They are also lagging behind others in terms of the disks they place. Interestingly, G-1991 performed really well both in terms of speed and cover size, especially for high-density pointsets; see Figs. 11 and 12. This is surprising because it has an approximation factor of 8, the highest among the algorithms we have engineered. In Fig. 12, we find that it placed the least number of disks while being very fast. We find DGT-2018 to be less efficient than most other algorithms regarding the number of disks placed.
  FastCover turned out to be the fastest of all because of its sheer simplicity and no use of any pre-processing such as sorting. FastCover+ (Heuristic 1 included) really helped to bring down the number of disks placed without any considerable slowdown. FastCover++ with Heuristic 2 reduced the number of disks further without slowing down too much. Refer to Fig. 9 for instance. In most of the cases, FastCover+ and FastCover++ could decrease the number of disks placed considerably. When $n = 10M$, FastCover has placed $4,323,694$ disks on average, FastCover+ has placed $3,752,103$ disks ($\approx 13\%$ reduction in cover size), and FastCover++ has placed $2,794,374$ disks ($\approx 35\%$ reduction in cover size compared to FastCover). The effect of these heuristics seems to vanish with the increase in point density since for high-density pointsets almost every disk placed by FastCover is required. In other words, for every disk that is placed, it is highly likely that there is at least one point that is covered by that disk only. Our experimental data says that the coalescing strategy used in Heuristic 2 is not slowing down the algorithm heavily. But this slowdown is

much less noticeable for higher density pointsets since, with the increase in density, the number of disks maintained by the algorithm decreases. As a result, there is a lesser number of disks to be considered for coalescing. This makes FastCover++ very speedy for high-density points, see Figs. 10, 11, 12.

- **Points drawn from a disk.** Just like our previous setup (points drawn from a square), we have used disks having areas $10^7, 10^6, 10^5, 10^4$. Refer to the Figs. 13, 14, 15, and 16. The results obtained are quite similar to the previous setup, as can be observed in the figures and therefore we omit the discussion.
- **Convex pointsets drawn from a square.** We have drawn random convex sets from squares whose areas are in $\{10^7, 10^6, 10^5, 10^4\}$. Interestingly, FastCover++ always placed the minimum number of disks. In terms of speed, FastCover turned out to be the fastest but FastCover++ was very close to it everywhere. To see this, refer to the Figs. 17, 18, 19, and 20. To our surprise, we find that unlike the previous two classes of pointsets, LL-2014-1P did not perform well in this case both in terms of speed and the number of disks placed. BLMS-2017 and CCFM-1997 turned out to be inferior both in terms of speed and cover size. G-1991 and DGT-2018 were very speedy in this case but lagged behind the FastCover++ in minimizing the number of disks placed. Our heuristics 1 and 2 together could substantially decrease cover sizes for this class of pointsets too. Refer to Fig. 17 for instance. For $n = 1M$, FastCover has placed 8941 disks on average. In contrast, FastCover++ has placed 5964 disks on average. This is a $\approx 33\%$ reduction in the number of disks placed. Note that the average runtime of FastCover in this case is 0.15 second and that of FastCover++ is 0.44 second. The heuristic 1 alone could not achieve much for this class of pointsets. This is evident from the number of disks placed by FastCover vs FastCover+.
- **Points drawn from an annulus.** In this setup, we have fixed the radius r_2 of the outer circle to 10^3 and have varied the radius r_1 of the inner circle. Refer to the Figs. 21, 22, and 23. In our experiments, $r_1 \in \{0.95 \cdot 10^3, 0.75 \cdot 10^3, 0.5 \cdot 10^3\}$. For this class of pointsets, LL-2014 and LL-2014-1P came out as the clear winner considering the cover sizes. G-1991 captured the second place in this regard and was competitive in speed. But in terms of speed, FastCover, FastCover+, and FastCover++ turned out to be the fastest of all and remained competitive in solution quality. The Heuristics 1 and 2 did a decent job of reducing the cover sizes but they were not effective as in the cases of other three classes of pointsets. For instance, refer to Fig. 21, $n = 1M$, FastCover has placed $155,984.80$ disks on average. In contrast, FastCover+ has placed $150,294.80$ disks (4% less number of disks), and FastCover++ has placed $141,845.80$ disks (9% less number of disks compared to FastCover) on average. BLMS-2017, CCFM-1997, and DGT-2018 were not only slower but also turned out to be less efficient in minimizing the cover sizes.

Real-world pointsets We have used the following eleven real-world pointsets for our experiments. The main reason behind the use of such pointsets is that they do not follow the popular synthetic distributions. Experimenting with them is beneficial to see how the algorithms perform on them both in speed and the number of disks placed. Further, the UDC algorithms can be used on real-world pointsets to tackle practical coverage problems such as facility location and tower placements in wireless networking. Hence, experiments with real-world pointsets throw light on the real-world efficacy of these algorithms. See Fig. 7 for the experimental results obtained for these pointsets. Note that these data sets have varied bounding box sizes.

- `birch3` [8]: An $100,000$-element pointset representing random sized clusters at random locations. Area of the bounding box: 8.84532×10^{11}. Point density: 1.13×10^{-7}
- `monalisa` [1,4]: A $100,000$-city TSP instance representing a continuous-line drawing of the Mona Lisa. Area of the bounding box: 3.98681×10^8. Point density: 0.00025
- `usa` [1,4]: A $115,475$-city TSP instance representing (nearly) all towns, villages, and cities in the United States. Area of the bounding box: 1.43141×10^9. Point density: 0.00008
- `KDDCU2D` [4,8]: An $145,751$-element pointset representing the first two dimensions of a protein data-set. Area of the bounding box: 8564.16. Point density: 17.019
- `europe` [4,8]: An $169,308$-element pointset representing differential coordinates of the map of Europe. Area of the bounding box: 10^{12}. Point density: 1.69308×10^{-7}
- `wildfires`[4]: An $1,880,465$-element pointset representing wildfire locations in USA. Area of the bounding box: 5948.76. Point density: 316.11
- `world` [1,4]: A $1,904,711$-city TSP instance consisting of all locations in the world that are registered as populated cities or towns, as well as several research bases in Antarctica. Area of the bounding box: 58938.8. Point density: 32.32
- `china` [4,8]: An $1,636,613$-element pointset representing locations in China. Area of the bounding box: 3188.92. Point density: 513.219
- `nyctaxi`[5]: An $2,917,288$-element pointset representing NYC taxi pickup and drop-off locations. Area of the bounding box: 1193.77. Point density: 2443.76

[4] https://www.kaggle.com/rtatman/188-million-us-wildfires/home.
[5] https://www.kaggle.com/wikunia/nyc-taxis-combined-with-dimacs/home.

Pointset	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
birch3	99990, 0.07	99993, 0.40	99989, 0.18	99991, 0.03	99994, 0.05	99993, 0.07	99996, **0.02**	99995, **0.02**	**99980**, 0.08
monalisa	**100000**, 0.05	**100000**, 0.41	**100000**, 0.15	**100000**, 0.03	**100000**, 0.11	**100000**, 0.08	**100000**, **0.02**	**100000**, **0.02**	**100000**, 0.08
usa	**115475**, 0.06	**115475**, 0.46	**115475**, 0.17	**115475**, 0.04	**115475**, 0.12	**115475**, 0.09	**115475**, **0.02**	**115475**, 0.03	**115475**, 0.10
KDDCU2D	1288, 0.04	1459, 0.09	**1147**, 0.19	1152, 0.04	1692, 0.10	1626, **0.01**	1418, **0.01**	1374, **0.01**	1257, **0.01**
europe	168087, 0.10	168079, 0.73	168253, 0.35	168271, 0.06	168088, 0.19	168069, 0.16	168333, **0.03**	168277, 0.04	**167811**, 0.20
wildfires	623, 0.57	710, 0.75	622, 3.74	622, 0.88	842, 1.21	787, 0.12	663, **0.04**	637, **0.04**	**620**, 0.06
world	7098, 0.40	8601, 0.73	**6667**, 2.84	6680, 0.51	9145, 0.79	10980, 0.15	7874, **0.03**	7576, 0.04	6967, 0.07
china	**562**, 0.36	639, 0.69	565, 3.74	567, 0.83	771, 0.96	723, 0.10	599, **0.03**	585,**0.03**	573, 0.06
nyctaxi	31, 0.59	29, 0.81	**25**, 13.84	26, 3.09	32, 2.90	31, 0.13	34, **0.05**	31, **0.05**	**25**, 0.10
uber	4, 0.79	5, 1.03	**3**, 21.06	**3**, 4.75	5, 4.03	5, 0.19	5, **0.06**	4, **0.06**	4, 0.16
hail2015	867, 6.22	998, 5.32	888, 39.83	889, 9.82	1193, 11.19	1128, 0.74	901, **0.28**	860, **0.28**	**847**, 0.42

Fig. 7. Experimental results for the real-world pointsets. A pair x, y in a cell denotes the number of disks placed and the running time in seconds, respectively, averaged over five runs of the same pointset. For every pointset, the smallest sized cover(s) and the fastest execution time(s) are shown in bold.

- uber[6] [4]: An 4, 534, 327-element pointset representing Uber pickup locations in New York City. Area of the bounding box: 7.04065. Point density: 644021.078
- hail2015[7]: An 10, 824, 080-element pointset representing hail storm cell locations based on NEXRAD radar data obtained in 2015. Area of the bounding box: 17921.8. Point density: 603.96

As evident from our experimental results in Fig. 7 FastCover, FastCover+, and FastCover++ performed really well on the real-world pointsets both in terms of speed and cover size. The Heuristics 1 and 2 performed reasonably well wherever possible. For the KDDCU2D pointset, FastCover, FastCover+, and FastCover++ have placed 1418, 1374, and 1257 disks, respectively. In this case, the Heuristic 1 could reduce cover size by approximately 3% and the heuristics 1 and 2 together could reduce the size by approximately 11%. For the world pointset, these percentages are ≈ 4 and 11, respectively. For these real-world pointsets, there is a negligible difference between FastCover and FastCover++ in terms of speed. To our surprise, we found that for some of the pointsets such as wildfires, and hail2015, FastCover++ has returned the smallest covers and quickly ran to completion. The other algorithms such as LL-2014-1P, G-1991, BLMS-2017, CCFM-1997, DGT-2018 turned out to be reasonably fast and returned competitive solutions on all the pointsets. G-1991 has placed the lowest number of disks for the china pointset. As in the case of synthetic pointsets, we found LL-2014 to be slower than LL-2014-1P. Further, LL-2014 could not reduce cover sizes substantially compared to its 1-pass version LL-2014-1P.

4. Our recommendations and conclusions

If the sizes of covers are of more interest than real-world running time, we recommend using LL-2014-1P. However, we also observe that in some cases, FastCover++ beats LL-2014 both in terms of speed and cover size.

If running time is much more important than cover size, we recommend using FastCover. Otherwise, we recommend using FastCover++ since it is very fast in practice and at the same time can generate low-sized covers. If it is known that the input pointsets are always convex, we recommend FastCover++ since in this case, it beats every other algorithm both in terms of speed and cover size.

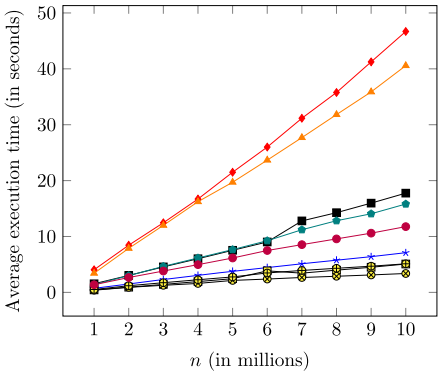
Overall, we find that either LL-2014-1P or FastCover++ is leading when it comes to the number of disks placed. So, in situations, where running time is not so important but the cover size is, one can run both LL-2014-1P and FastCover++ and take the best of the two covers.

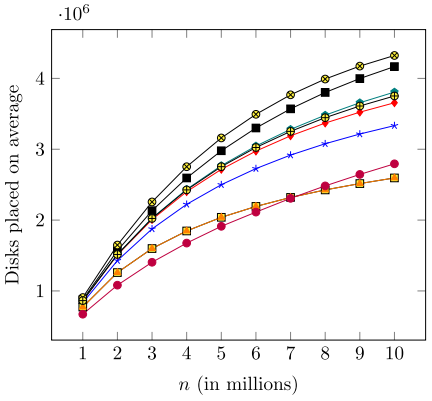
5. Plots and tables

In the interest of space, we avoid legend tables everywhere in our plots. Since the legends are used uniformly throughout this work, we present them here for an easy reference; refer to Fig. 8. In the tables, we mark the fastest execution time and the smallest covers in bold.

Fig. 8. The plot legends.

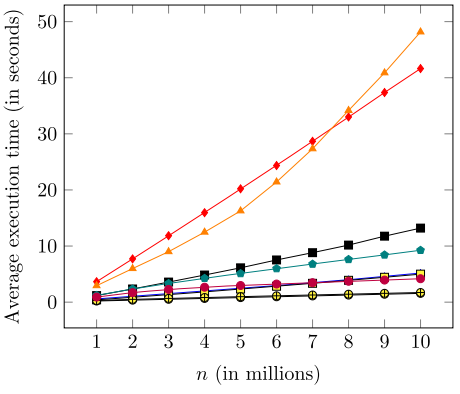
[6] https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city.
[7] https://www.kaggle.com/noaa/severe-weather-data-inventory.

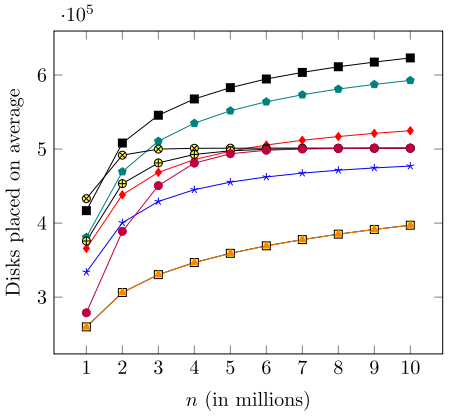




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	833679.00, 0.72	862355.00, 4.05	772740.00, 3.42	772740.00, 0.44	875865.00, 1.54	863576.00, 1.38	906791.00, **0.39**	863633.00, 0.52	**671552.00**, 1.37
$2M$	1428402.00, 1.50	1508485.00, 8.44	1259503.00, 7.88	1259503.00, 0.93	1569025.00, 3.03	1516204.00, 3.00	1647869.00, **0.90**	1514736.00, 1.19	**1081452.00**, 2.66
$3M$	1875574.00, 2.29	2008084.00, 12.45	1598021.00, 12.00	1598743.00, 1.42	2133334.00, 4.56	2027370.00, 4.65	2256218.00, **1.26**	2023099.00, 1.73	**1404776.00**, 3.82
$4M$	2222556.00, 3.05	2400938.00, 16.71	1846366.00, 16.22	1846520.00, 1.93	2594608.00, 6.02	2434139.00, 6.16	2753330.00, **1.60**	2426191.00, 2.26	**1675416.00**, 4.96
$5M$	2499513.00, 3.78	2716215.00, 21.50	2039186.00, 19.71	2039186.00, 2.45	2978678.00, 7.53	2768616.00, 7.65	3159626.00, **2.14**	2753980.00, 2.77	**1912206.00**, 6.15
$6M$	2726832.00, 4.45	2973789.00, 26.00	2192751.00, 23.66	2192751.00, 3.85	3300020.00, 9.02	3045992.00, 9.24	3493144.00, **2.39**	3024817.00, 3.48	**2112692.00**, 7.49
$7M$	2917755.00, 5.10	3189618.00, 31.16	2318895.00, 27.69	2319858.00, 3.44	3570201.00, 12.79	3280443.00, 11.21	3768341.00, **2.66**	3252319.00, 3.92	**2306924.00**, 8.54
$8M$	3077704.00, 5.77	3369697.00, 35.77	**2425896.00**, 31.81	**2425896.00**, 3.98	3801326.00, 14.26	3481160.00, 12.82	3990925.00, **2.88**	3445698.00, 4.31	2482657.00, 9.57
$9M$	3214782.00, 6.39	3523199.00, 41.24	**2515687.00**, 35.87	**2515687.00**, 4.49	3997989.00, 15.97	3656459.00, 14.07	4173678.00, **3.12**	3610278.00, 4.69	2644615.00, 10.60
$10M$	3333725.00, 7.10	3656068.00, 46.69	**2595126.00**, 40.57	**2595126.00**, 5.08	4167035.00, 17.76	3807011.00, 15.81	4323694.00, **3.38**	3752103.00, 5.11	2794374.00, 11.77

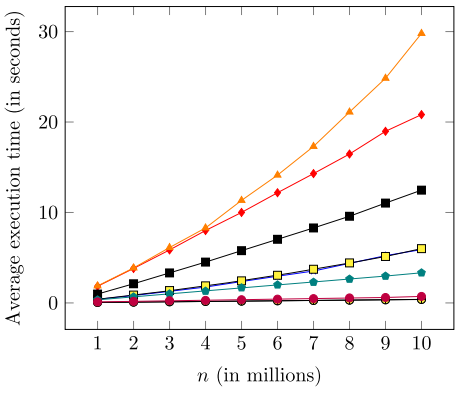
Fig. 9. Points drawn from a square of area 10^7. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

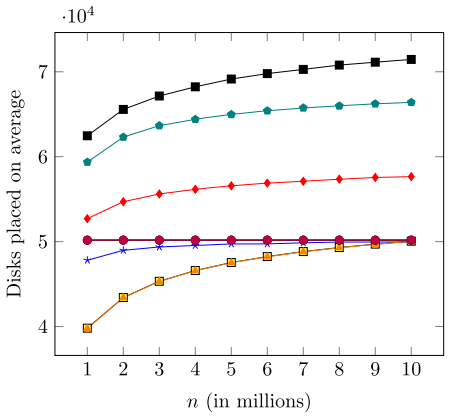




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	333744.60, 0.58	365777.20, 3.65	**259664.40**, 2.93	259842.60, 0.41	416818.00, 1.17	380899.40, 1.21	433044.80, **0.20**	375825.80, 0.29	278895.20, 0.91
$2M$	400520.40, 1.05	438231.60, 7.71	**306192.40**, 5.99	306263.40, 0.87	508165.80, 2.35	469466.60, 2.35	491765.60, **0.36**	453316.40, 0.50	388859.00, 1.70
$3M$	429268.00, 1.54	468538.80, 11.84	**330307.00**, 9.00	330413.40, 1.34	545693.00, 3.58	510634.20, 3.34	499944.60, **0.51**	481315.40, 0.67	450632.40, 2.27
$4M$	445134.60, 2.01	485794.60, 15.95	**346557.20**, 12.48	346757.80, 1.84	567712.40, 4.84	534944.40, 4.25	501056.40, **0.66**	492758.40, 0.82	480987.60, 2.66
$5M$	455346.00, 2.45	497153.20, 20.20	**359054.60**, 16.27	359209.00, 2.33	582947.00, 6.12	551741.80, 5.11	501233.60, **0.81**	497661.20, 0.98	493710.40, 2.97
$6M$	462332.40, 2.95	505495.80, 24.35	**369154.80**, 21.42	369339.60, 2.83	594661.20, 7.52	563888.60, 5.99	501256.40, **0.96**	499694.80, 1.13	498363.40, 3.23
$7M$	467528.20, 3.46	511872.20, 28.67	**377662.40**, 27.33	377709.00, 3.36	603573.00, 8.82	573362.80, 6.79	501262.20, **1.11**	500594.00, 1.27	500115.00, 3.47
$8M$	471437.00, 4.03	516913.40, 33.01	**384974.20**, 34.16	385081.40, 3.89	611188.40, 10.17	581090.40, 7.62	501263.20, **1.26**	500969.20, 1.42	500784.60, 3.73
$9M$	474539.00, 4.60	521240.00, 37.36	**391388.00**, 40.87	391485.20, 4.43	617495.60, 11.76	587309.40, 8.42	501264.00, **1.41**	501130.20, 1.57	501042.00, 3.94
$10M$	477060.40, 5.19	524835.80, 41.63	**397087.00**, 48.16	397181.40, 4.99	623091.20, 13.20	592748.00, 9.25	501264.00, **1.56**	501205.60, 1.72	501161.20, 4.18

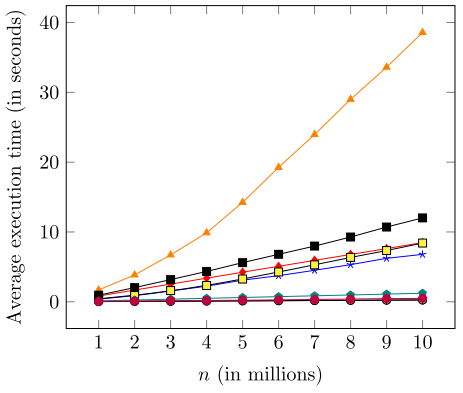
Fig. 10. Points drawn from a square of area 10^6. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

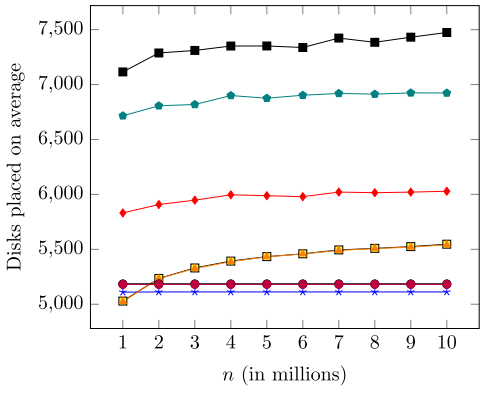




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	47803.80, 0.40	52700.60, 1.84	**39766.00**, 1.88	39806.20, 0.40	62463.80, 0.98	59362.20, 0.34	50176.00, **0.05**	50170.40, 0.05	50168.60, 0.11
$2M$	48975.20, 0.82	54698.00, 3.81	**43377.40**, 3.87	43411.20, 0.87	65568.60, 2.12	62308.60, 0.67	50176.00, **0.08**	50176.00, 0.09	50176.00, 0.17
$3M$	49366.40, 1.28	55609.00, 5.86	**45304.80**, 6.12	45327.00, 1.37	67146.20, 3.32	63657.40, 1.01	50176.00, **0.12**	50176.00, 0.13	50176.00, 0.25
$4M$	49545.20, 1.75	56155.60, 8.00	**46568.60**, 8.29	46594.20, 1.89	68229.80, 4.51	64399.80, 1.31	50176.00, **0.16**	50176.00, 0.16	50176.00, 0.28
$5M$	49725.20, 2.36	56574.80, 10.00	**47512.00**, 11.34	47535.60, 2.44	69146.00, 5.78	64971.20, 1.68	50176.00, 0.20	50176.00, **0.20**	50176.00, 0.36
$6M$	49731.80, 2.95	56889.20, 12.18	**48210.80**, 14.13	48236.00, 3.06	69785.20, 7.03	65407.40, 2.00	50176.00, 0.24	50176.00, **0.24**	50176.00, 0.42
$7M$	49859.00, 3.53	57105.40, 14.30	**48797.00**, 17.30	48826.20, 3.72	70279.20, 8.29	65747.80, 2.30	50176.00, 0.28	50176.00, **0.27**	50176.00, 0.48
$8M$	49947.40, 4.38	57357.40, 16.46	**49274.80**, 21.10	49298.20, 4.42	70789.40, 9.58	65984.00, 2.64	50176.00, 0.32	50176.00, **0.31**	50176.00, 0.56
$9M$	49951.80, 5.23	57554.40, 18.97	**49679.80**, 24.84	49700.20, 5.14	71127.60, 11.04	66225.80, 2.97	50176.00, 0.35	50176.00, **0.34**	50176.00, 0.60
$10M$	**49952.00**, 5.90	57651.40, 20.82	50031.20, 29.79	50051.40, 6.00	71452.40, 12.47	66396.60, 3.33	50176.00, 0.39	50176.00, **0.38**	50176.00, 0.72

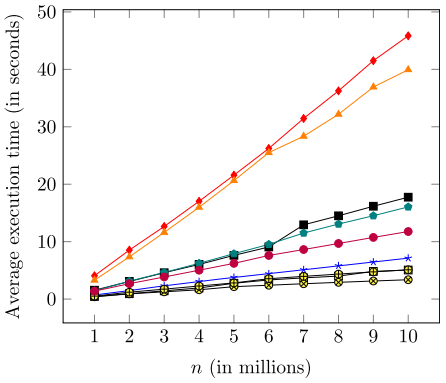
Fig. 11. Points drawn from a square of area 10^5. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

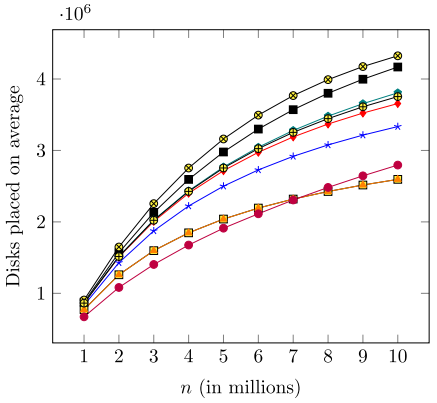




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	5110.20, 0.38	5831.20, 0.82	**5024.20**, 1.70	5028.80, 0.42	7115.00, 0.94	6715.60, 0.13	5184.00, **0.03**	5184.00, 0.03	5180.20, 0.05
$2M$	**5112.00**, 0.87	5907.80, 1.68	5233.20, 3.84	5235.00, 0.95	7287.60, 2.03	6806.80, 0.25	5184.00, 0.05	5184.00, **0.05**	5180.00, 0.10
$3M$	**5112.00**, 1.55	5947.40, 2.53	5325.60, 6.71	5330.20, 1.60	7310.60, 3.18	6818.40, 0.37	5184.00, 0.08	5184.00, **0.08**	5180.20, 0.15
$4M$	**5112.00**, 2.24	5996.40, 3.38	5387.60, 9.88	5392.80, 2.33	7350.80, 4.34	6900.20, 0.49	5184.00, 0.11	5184.00, **0.10**	5180.00, 0.18
$5M$	**5112.00**, 3.10	5988.20, 4.24	5431.80, 14.24	5433.40, 3.26	7351.40, 5.61	6875.60, 0.62	5184.00, 0.13	5184.00, **0.13**	5180.40, 0.24
$6M$	**5112.00**, 3.74	5979.20, 5.06	5457.80, 19.25	5459.00, 4.27	7337.40, 6.80	6902.80, 0.74	5184.00, 0.16	5184.00, **0.15**	5180.00, 0.28
$7M$	**5112.00**, 4.51	6021.00, 5.94	5490.00, 23.95	5494.40, 5.29	7423.20, 7.98	6919.20, 0.86	5184.00, 0.19	5184.00, **0.17**	5180.20, 0.31
$8M$	**5112.00**, 5.32	6015.60, 6.78	5507.00, 29.00	5508.80, 6.36	7384.60, 9.28	6912.80, 0.97	5184.00, 0.21	5184.00, **0.20**	5180.00, 0.39
$9M$	**5112.00**, 6.22	6020.80, 7.62	5520.00, 33.60	5524.60, 7.34	7431.40, 10.70	6924.40, 1.10	5184.00, 0.24	5184.00, **0.23**	5180.20, 0.45
$10M$	**5112.00**, 6.79	6028.60, 8.48	5541.40, 38.57	5546.60, 8.41	7473.80, 12.02	6923.20, 1.22	5184.00, 0.26	5184.00, **0.25**	5180.20, 0.48

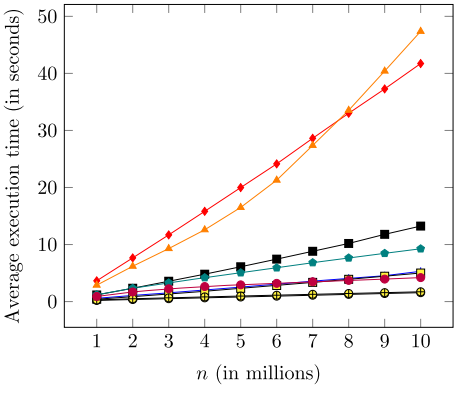
Fig. 12. Points drawn from a square of area 10^4. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

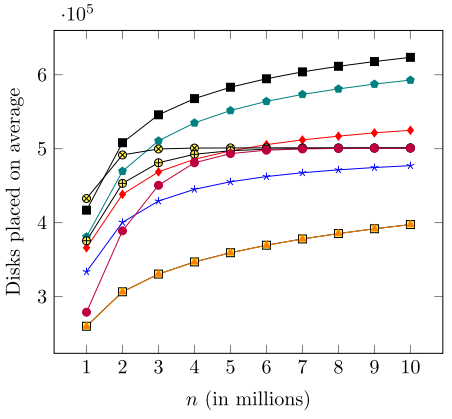




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	833514.20, 0.71	862124.00, 4.07	772335.00, 3.29	772550.00, 0.45	875777.00, 1.54	863466.60, 1.43	906414.80, **0.40**	863156.60, 0.52	**671410.60**, 1.38
$2M$	1428553.80, 1.48	1509081.20, 8.53	1259743.60, 7.35	1260130.80, 0.94	1569100.80, 3.07	1516469.60, 3.11	1648598.40, **0.91**	1515119.20, 1.20	**1082127.60**, 2.68
$3M$	1874756.20, 2.31	2006987.80, 12.65	1597173.40, 11.61	1597632.80, 1.44	2131436.60, 4.59	2025876.20, 4.64	2255540.80, **1.26**	2022218.20, 1.73	**1404007.20**, 3.85
$4M$	2222618.00, 3.06	2400905.20, 17.02	1846483.40, 15.99	1846920.80, 1.94	2594345.40, 6.03	2434661.60, 6.23	2753982.40, **1.62**	2426913.00, 2.28	**1675862.20**, 5.02
$5M$	2500075.60, 3.79	2716239.60, 21.58	2038871.80, 20.63	2039637.20, 2.75	2978434.40, 7.61	2768185.00, 7.90	3160523.20, **2.16**	2754663.60, 2.81	**1911830.20**, 6.22
$6M$	2727673.40, 4.40	2974065.40, 26.23	2192752.00, 25.50	2193439.60, 3.33	3299265.00, 9.07	3044865.20, 9.52	3494245.40, **2.41**	3025422.20, 3.54	**2113411.20**, 7.60
$7M$	2917478.60, 5.11	3189142.60, 31.45	2319005.60, 28.35	2319573.40, 3.69	3570456.20, 12.93	3279889.00, 11.50	3768003.60, **2.67**	3253149.20, 3.96	**2306633.00**, 8.63
$8M$	3077294.40, 5.81	3369785.20, 36.26	**2425023.40**, 32.18	2425540.40, 4.00	3800287.40, 14.50	3480128.60, 13.07	3990991.40, **2.92**	3445295.00, 4.35	2482418.60, 9.68
$9M$	3214640.20, 6.46	3523038.00, 41.51	**2515925.40**, 36.92	2516260.60, 4.82	3997048.40, 16.16	3654549.40, 14.51	4174186.40, **3.15**	3611159.00, 4.74	2644630.40, 10.74
$10M$	3334151.40, 7.13	3656139.00, 45.84	**2594921.40**, 39.94	2595210.20, 5.06	4167852.60, 17.74	3806702.60, 16.05	4324587.00, **3.37**	3753906.20, 5.10	2795167.60, 11.76

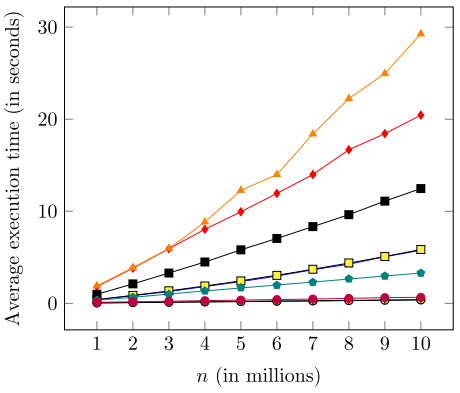
Fig. 13. Points drawn from a disk of area 10^7. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

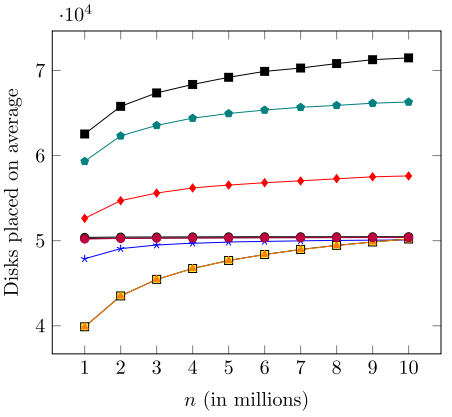




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	333541.00, 0.57	365924.60, 3.64	**259587.20**, 2.87	259803.00, 0.41	416797.00, 1.16	380704.80, 1.20	432543.20, **0.20**	375422.20, 0.29	278807.80, 0.90
$2M$	400380.00, 1.08	438209.00, 7.66	**306214.80**, 6.19	306435.00, 0.86	508011.00, 2.34	469493.40, 2.34	491462.20, **0.36**	453014.20, 0.49	388862.20, 1.69
$3M$	429152.20, 1.56	468564.00, 11.69	**330297.20**, 9.29	330372.20, 1.33	545754.20, 3.57	510548.40, 3.29	499606.00, **0.51**	480920.40, 0.66	450442.40, 2.25
$4M$	445014.80, 2.02	485716.60, 15.82	**346645.80**, 12.58	346790.60, 1.82	567664.40, 4.80	534829.00, 4.20	500767.60, **0.66**	492369.80, 0.82	480887.40, 2.64
$5M$	455171.00, 2.54	496922.20, 19.98	**359130.60**, 16.51	359254.60, 2.32	583029.00, 6.11	551526.40, 5.06	500981.40, **0.80**	497184.80, 0.97	493284.40, 2.93
$6M$	462246.80, 3.04	505437.60, 24.14	**369268.00**, 21.28	369397.00, 2.82	594514.60, 7.44	563937.80, 5.93	501066.00, **0.95**	499220.80, 1.12	497893.00, 3.19
$7M$	467428.60, 3.56	511824.00, 28.62	**377747.80**, 27.37	377887.00, 3.37	603819.40, 8.82	573347.40, 6.83	501096.00, **1.12**	500121.20, 1.28	499599.60, 3.48
$8M$	471330.80, 4.10	516869.80, 33.01	**385136.40**, 33.53	385167.00, 3.91	611350.00, 10.19	580776.40, 7.66	501120.60, **1.26**	500519.00, 1.42	500246.20, 3.72
$9M$	474456.60, 4.54	521327.60, 37.28	**391524.40**, 40.40	391672.00, 4.45	617758.40, 11.79	587292.80, 8.46	501143.60, **1.41**	500681.20, 1.57	500498.60, 3.94
$10M$	476996.00, 5.31	524810.20, 41.73	**397231.60**, 47.37	397349.80, 5.01	623442.00, 13.23	592680.00, 9.26	501169.60, **1.56**	500768.40, 1.72	500621.20, 4.20

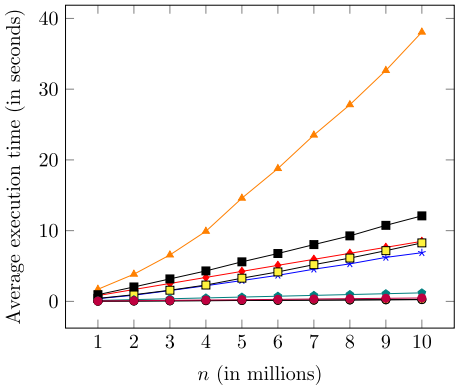
Fig. 14. Points drawn from a disk of area 10^6. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

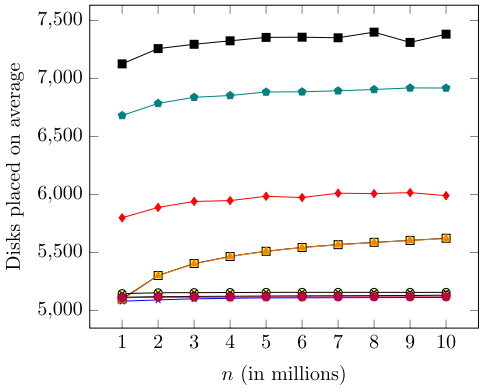




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	47875.20, 0.41	52625.20, 1.82	**39861.80**, 1.89	39881.60, 0.40	62535.20, 0.98	59322.40, 0.34	50387.40, **0.05**	50240.60, 0.05	50186.40, 0.11
$2M$	49065.40, 0.84	54700.20, 3.82	**43491.20**, 3.86	43507.00, 0.87	65784.80, 2.12	62315.80, 0.66	50423.00, **0.08**	50288.60, 0.09	50260.60, 0.17
$3M$	49487.60, 1.29	55596.80, 5.93	**45452.40**, 5.94	45452.80, 1.36	67363.00, 3.29	63544.40, 0.99	50436.60, **0.12**	50322.80, 0.13	50300.60, 0.23
$4M$	49704.40, 1.83	56192.60, 8.05	**46721.40**, 8.84	46728.40, 1.89	68356.80, 4.49	64396.80, 1.35	50446.20, **0.16**	50335.80, 0.16	50315.40, 0.29
$5M$	49829.20, 2.37	56538.60, 9.94	**47661.00**, 12.27	47673.20, 2.44	69197.40, 5.81	64940.20, 1.70	50450.80, 0.20	50347.40, **0.20**	50329.80, 0.35
$6M$	49919.20, 2.96	56816.40, 11.94	**48371.60**, 13.99	48373.20, 3.04	69894.20, 7.05	65349.60, 1.97	50458.60, 0.24	50355.00, **0.23**	50340.60, 0.41
$7M$	49988.00, 3.66	57017.00, 13.99	**48953.20**, 18.40	48969.00, 3.71	70285.00, 8.34	65671.60, 2.30	50459.20, 0.28	50360.60, **0.27**	50349.60, 0.47
$8M$	50027.60, 4.25	57292.00, 16.69	**49427.00**, 22.24	49450.00, 4.39	70810.80, 9.63	65896.60, 2.67	50462.20, 0.32	50366.00, **0.31**	50355.40, 0.54
$9M$	50059.20, 5.07	57505.40, 18.44	**49843.60**, 24.96	49845.20, 5.08	71267.20, 11.10	66152.00, 2.97	50461.60, 0.35	50373.80, **0.34**	50364.80, 0.62
$10M$	**50086.60**, 5.76	57607.60, 20.44	50193.20, 29.27	50199.80, 5.85	71479.20, 12.47	66294.80, 3.29	50467.00, 0.39	50376.60, **0.38**	50367.60, 0.66

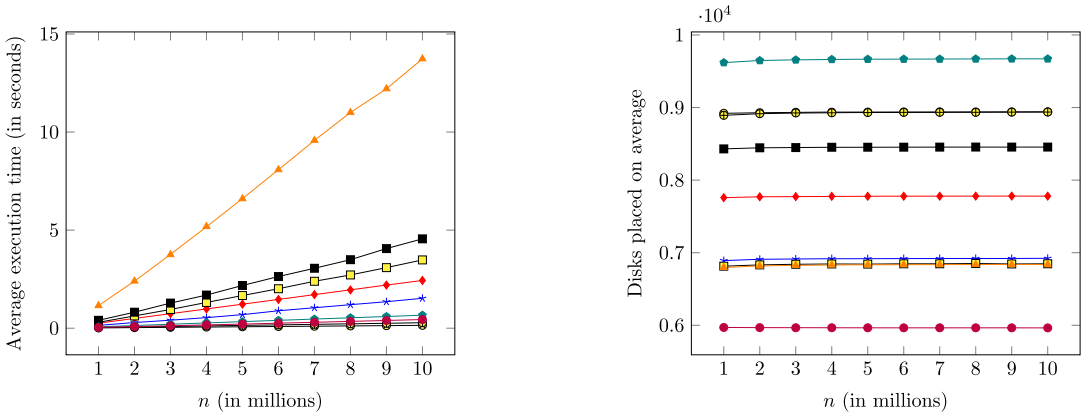
Fig. 15. Points drawn from a disk of area 10^5. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.





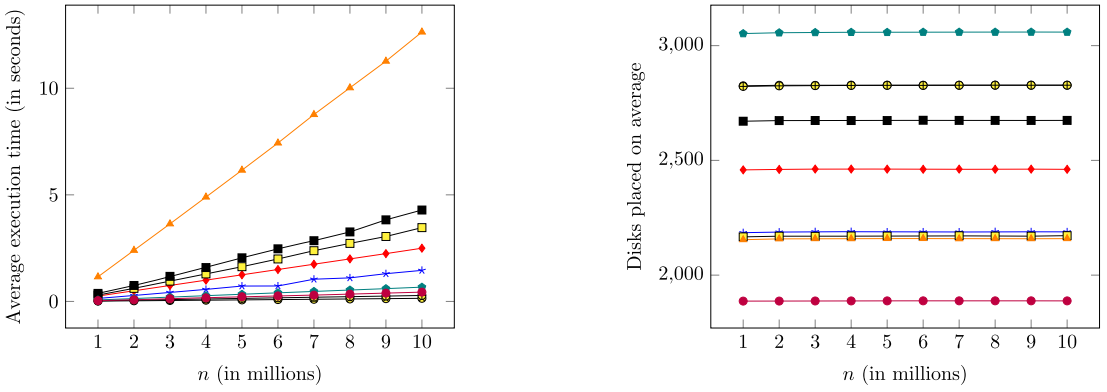
n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	**5079.80**, 0.38	5799.00, 0.82	5090.80, 1.71	5099.00, 0.42	7125.60, 0.95	6680.80, 0.13	5145.20, **0.03**	5113.40, 0.03	5109.20, 0.05
$2M$	**5091.00**, 0.87	5888.40, 1.68	5300.20, 3.84	5301.20, 0.95	7257.40, 2.04	6785.00, 0.25	5152.40, 0.05	5118.80, **0.05**	5114.20, 0.10
$3M$	**5099.80**, 1.53	5939.60, 2.52	5401.40, 6.56	5404.40, 1.58	7294.60, 3.18	6837.60, 0.37	5153.60, 0.08	5122.00, **0.08**	5115.40, 0.15
$4M$	**5106.20**, 2.21	5947.00, 3.38	5463.80, 9.91	5465.20, 2.31	7324.60, 4.31	6853.00, 0.49	5154.80, 0.11	5121.20, **0.10**	5115.20, 0.19
$5M$	**5107.80**, 2.95	5984.00, 4.24	5509.40, 14.59	5510.00, 3.28	7354.00, 5.59	6883.20, 0.61	5155.80, 0.13	5124.40, **0.13**	5116.00, 0.24
$6M$	**5109.20**, 3.69	5973.40, 5.10	5540.00, 18.78	5542.80, 4.17	7355.80, 6.77	6885.00, 0.73	5156.00, 0.16	5125.60, **0.15**	5115.80, 0.28
$7M$	**5110.00**, 4.55	6010.80, 5.95	5565.60, 23.51	5567.00, 5.19	7350.60, 8.03	6893.60, 0.85	5156.00, 0.19	5126.20, **0.18**	5116.40, 0.34
$8M$	**5111.40**, 5.30	6007.00, 6.80	5586.00, 27.80	5587.40, 6.12	7399.40, 9.26	6904.80, 0.97	5156.00, 0.21	5128.40, **0.20**	5116.40, 0.37
$9M$	**5111.60**, 6.22	6015.80, 7.64	5603.00, 32.64	5603.80, 7.17	7310.80, 10.75	6918.40, 1.09	5156.00, 0.24	5128.60, **0.22**	5116.60, 0.43
$10M$	**5111.80**, 6.88	5989.80, 8.49	5619.80, 38.06	5622.60, 8.27	7381.80, 12.09	6917.60, 1.20	5155.80, 0.27	5131.80, **0.25**	5116.60, 0.48

Fig. 16. Points drawn from a disk of area 10^4. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.



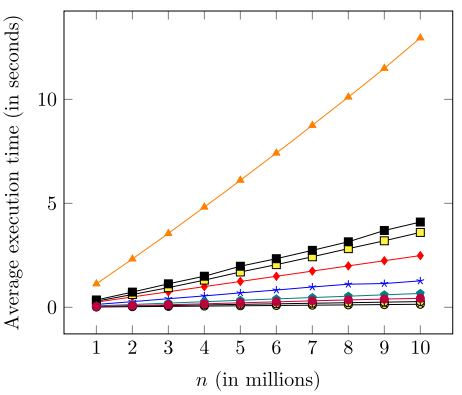
n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	6891.60, 0.15	7758.40, 0.26	6796.60, 1.15	6814.40, 0.30	8430.40, 0.40	9619.20, 0.07	8920.40, **0.02**	8892.80, 0.03	**5970.00**, 0.04
$2M$	6909.40, 0.29	7770.40, 0.50	6817.80, 2.40	6832.20, 0.63	8445.40, 0.82	9648.00, 0.14	8928.60, **0.03**	8915.20, 0.06	**5967.80**, 0.08
$3M$	6913.20, 0.41	7773.20, 0.74	6825.80, 3.75	6840.60, 0.96	8448.40, 1.28	9656.80, 0.20	8934.60, **0.04**	8925.60, 0.09	**5966.20**, 0.13
$4M$	6918.20, 0.54	7776.60, 0.98	6831.20, 5.19	6844.80, 1.31	8452.80, 1.69	9663.00, 0.27	8936.40, **0.06**	8927.60, 0.11	**5966.00**, 0.17
$5M$	6917.20, 0.69	7777.40, 1.23	6833.20, 6.60	6842.80, 1.67	8453.40, 2.18	9666.40, 0.33	8937.40, **0.07**	8931.00, 0.14	**5965.60**, 0.21
$6M$	6919.00, 0.89	7778.00, 1.47	6834.80, 8.08	6847.00, 2.01	8454.00, 2.63	9667.80, 0.40	8939.00, **0.09**	8933.80, 0.17	**5964.80**, 0.25
$7M$	6919.80, 1.04	7778.60, 1.72	6837.20, 9.57	6847.20, 2.39	8454.80, 3.06	9668.40, 0.47	8940.20, **0.10**	8934.40, 0.20	**5965.20**, 0.30
$8M$	6921.40, 1.20	7779.60, 1.95	6836.60, 11.00	6851.20, 2.72	8455.80, 3.49	9670.20, 0.53	8938.80, **0.12**	8934.40, 0.22	**5965.00**, 0.35
$9M$	6920.20, 1.35	7780.20, 2.20	6837.40, 12.21	6845.00, 3.09	8455.20, 4.06	9671.20, 0.60	8940.20, **0.13**	8937.00, 0.25	**5965.00**, 0.39
$10M$	6923.00, 1.52	7779.20, 2.43	6836.40, 13.73	6847.40, 3.48	8455.40, 4.56	9671.20, 0.66	8941.00, **0.15**	8938.00, 0.28	**5964.00**, 0.44

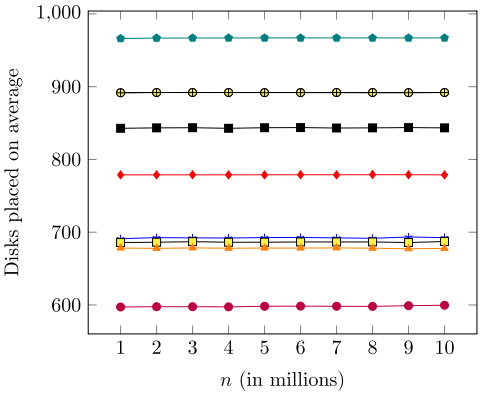
Fig. 17. Convex pointsets drawn from a square of area 10^7. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.



n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	2185.00, 0.14	2458.60, 0.25	2154.40, 1.15	2166.80, 0.30	2670.60, 0.38	3052.60, 0.07	2824.80, **0.01**	2822.40, 0.03	**1886.80**, 0.04
$2M$	2187.60, 0.28	2460.60, 0.50	2158.00, 2.40	2169.60, 0.61	2673.40, 0.75	3056.20, 0.14	2827.20, **0.03**	2824.80, 0.06	**1887.00**, 0.08
$3M$	2189.20, 0.42	2462.20, 0.75	2158.80, 3.64	2169.00, 0.95	2673.80, 1.18	3057.40, 0.20	2826.80, **0.04**	2825.80, 0.08	**1887.40**, 0.13
$4M$	2189.60, 0.56	2462.20, 1.00	2158.60, 4.90	2170.60, 1.29	2673.60, 1.59	3058.20, 0.27	2827.80, **0.06**	2826.60, 0.11	**1887.60**, 0.17
$5M$	2188.00, 0.72	2461.80, 1.25	2159.40, 6.16	2170.20, 1.63	2673.60, 2.05	3058.00, 0.34	2827.80, **0.07**	2827.00, 0.14	**1887.80**, 0.22
$6M$	2188.20, 0.72	2461.20, 1.50	2159.40, 7.43	2169.80, 1.99	2674.80, 2.47	3058.80, 0.40	2827.40, **0.09**	2826.80, 0.17	**1888.00**, 0.26
$7M$	2188.00, 1.04	2462.00, 1.74	2158.40, 8.77	2171.60, 2.38	2674.20, 2.85	3059.00, 0.47	2827.40, **0.10**	2827.40, 0.20	**1888.00**, 0.29
$8M$	2189.40, 1.10	2461.20, 2.00	2158.80, 10.02	2170.20, 2.72	2674.20, 3.26	3059.00, 0.53	2827.80, **0.12**	2827.60, 0.22	**1888.00**, 0.34
$9M$	2188.80, 1.30	2461.80, 2.24	2158.60, 11.27	2169.80, 3.05	2674.00, 3.83	3059.20, 0.60	2827.80, **0.13**	2827.40, 0.25	**1888.00**, 0.39
$10M$	2188.60, 1.46	2461.00, 2.50	2159.00, 12.64	2172.00, 3.46	2674.20, 4.29	3059.00, 0.67	2827.60, **0.15**	2827.60, 0.28	**1887.80**, 0.44

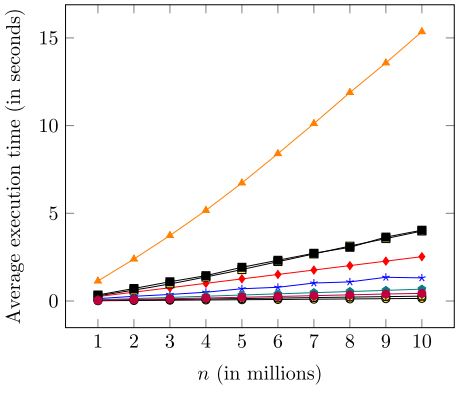
Fig. 18. Convex pointsets drawn from a square of area 10^6. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

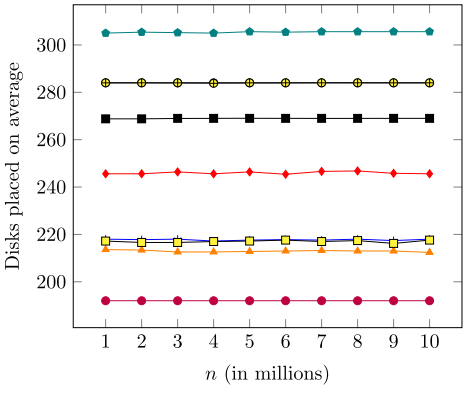




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	691.00, 0.14	778.80, 0.25	678.20, 1.13	685.80, 0.29	842.80, 0.35	966.20, 0.07	891.80, **0.01**	891.60, 0.03	**597.20**, 0.04
$2M$	692.60, 0.27	778.80, 0.50	677.80, 2.32	686.20, 0.61	843.40, 0.73	966.80, 0.14	892.00, **0.03**	892.00, 0.06	**597.60**, 0.08
$3M$	692.20, 0.41	779.00, 0.75	678.40, 3.55	687.00, 0.95	843.60, 1.13	967.00, 0.20	892.00, **0.04**	892.00, 0.08	**597.60**, 0.13
$4M$	692.00, 0.55	778.80, 1.00	678.00, 4.82	686.20, 1.31	842.80, 1.50	967.00, 0.27	892.00, **0.06**	892.00, 0.11	**597.40**, 0.17
$5M$	692.60, 0.69	778.80, 1.24	678.20, 6.11	686.20, 1.70	843.60, 1.98	967.00, 0.34	891.80, **0.07**	891.80, 0.14	**598.20**, 0.22
$6M$	692.80, 0.83	779.00, 1.49	678.20, 7.42	686.60, 2.05	843.80, 2.34	967.00, 0.40	892.00, **0.09**	891.80, 0.17	**598.40**, 0.26
$7M$	692.20, 0.98	778.80, 1.74	678.40, 8.75	686.60, 2.43	843.20, 2.74	967.00, 0.47	892.00, **0.10**	892.00, 0.20	**598.20**, 0.31
$8M$	691.60, 1.11	779.20, 1.99	678.00, 10.11	686.60, 2.82	843.40, 3.15	967.00, 0.54	892.00, **0.12**	892.00, 0.22	**598.00**, 0.36
$9M$	693.40, 1.14	779.00, 2.24	677.20, 11.49	685.80, 3.20	843.80, 3.69	967.00, 0.60	891.80, **0.13**	891.80, 0.25	**599.00**, 0.40
$10M$	692.60, 1.27	778.80, 2.49	677.80, 12.96	687.20, 3.60	843.40, 4.10	967.00, 0.67	892.00, **0.15**	892.00, 0.27	**599.60**, 0.43

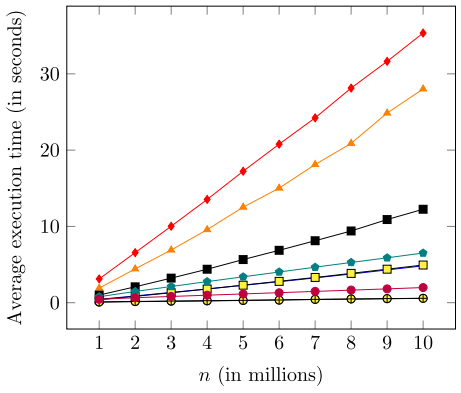
Fig. 19. Convex pointsets drawn from a square of area 10^5. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

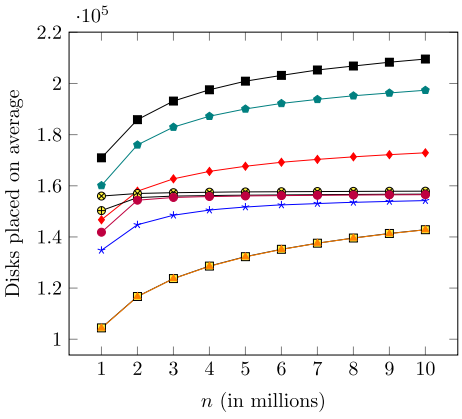




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	218.00, 0.13	245.60, 0.25	213.60, 1.14	217.20, 0.29	268.80, 0.34	305.00, 0.07	284.00, **0.01**	284.00, 0.03	**192.00**, 0.04
$2M$	217.80, 0.27	245.60, 0.50	213.40, 2.39	216.60, 0.62	268.80, 0.71	305.40, 0.14	284.00, **0.03**	284.00, 0.06	**192.00**, 0.08
$3M$	218.00, 0.37	246.40, 0.75	212.60, 3.73	216.60, 0.98	269.00, 1.10	305.20, 0.20	284.00, **0.04**	284.00, 0.08	**192.00**, 0.13
$4M$	217.20, 0.50	245.60, 1.01	212.60, 5.17	217.00, 1.36	269.00, 1.45	305.00, 0.27	284.00, **0.06**	283.80, 0.11	**192.00**, 0.17
$5M$	217.60, 0.70	246.40, 1.26	212.80, 6.73	217.20, 1.79	269.00, 1.92	305.60, 0.34	284.00, **0.07**	284.00, 0.14	**192.00**, 0.22
$6M$	217.80, 0.78	245.40, 1.51	213.00, 8.40	217.60, 2.25	269.00, 2.32	305.40, 0.41	284.00, **0.09**	284.00, 0.17	**192.00**, 0.26
$7M$	217.60, 1.02	246.60, 1.76	213.20, 10.11	217.00, 2.68	269.00, 2.72	305.60, 0.48	284.00, **0.10**	284.00, 0.20	**192.00**, 0.30
$8M$	218.00, 1.09	246.80, 2.01	213.00, 11.89	217.40, 3.13	269.00, 3.06	305.60, 0.54	284.00, **0.12**	284.00, 0.22	**192.00**, 0.35
$9M$	217.40, 1.35	245.80, 2.27	213.00, 13.57	216.20, 3.56	269.00, 3.64	305.60, 0.61	284.00, **0.13**	284.00, 0.25	**192.00**, 0.40
$10M$	218.00, 1.32	245.60, 2.53	212.40, 15.36	217.60, 3.99	269.00, 4.04	305.60, 0.68	284.00, **0.15**	284.00, 0.28	**192.00**, 0.43

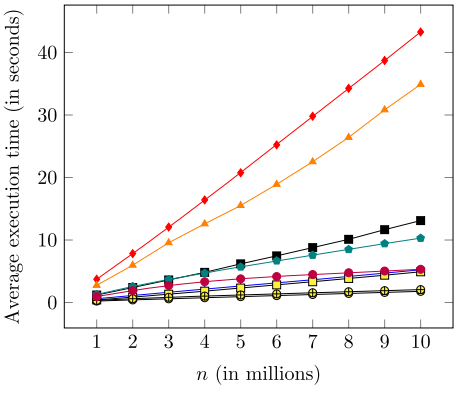
Fig. 20. Convex pointsets drawn from a square of area 10^4. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

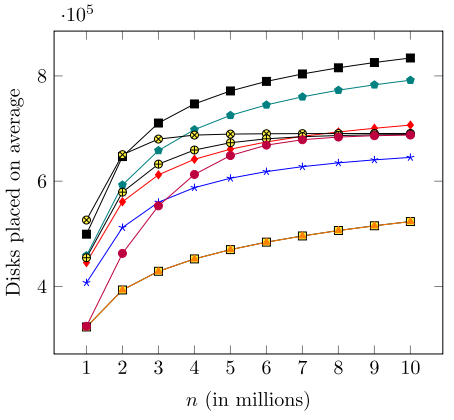




n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	134780.00, 0.44	146698.20, 3.12	**104349.20**, 1.91	104439.00, 0.39	170936.00, 1.00	160100.80, 0.78	155984.80, **0.08**	150294.80, 0.10	141845.80, 0.45
$2M$	144766.20, 0.89	157894.80, 6.56	**116646.40**, 4.43	116711.20, 0.83	185858.60, 2.08	175997.80, 1.47	156986.60, **0.14**	155379.60, 0.16	154364.00, 0.65
$3M$	148493.20, 1.34	162721.40, 10.02	**123690.80**, 6.90	123719.20, 1.31	193126.80, 3.22	182944.60, 2.13	157307.40, **0.19**	155967.00, 0.22	155431.20, 0.82
$4M$	150498.80, 1.80	165629.00, 13.53	**128517.20**, 9.58	128603.00, 1.79	197538.20, 4.39	187194.80, 2.77	157499.40, **0.24**	156186.00, 0.26	155804.80, 0.99
$5M$	151712.80, 2.28	167636.00, 17.23	**132199.40**, 12.52	132286.00, 2.29	200905.00, 5.65	190047.00, 3.39	157595.00, **0.29**	156333.40, 0.31	156013.20, 1.15
$6M$	152503.60, 2.73	169196.00, 20.78	**135108.60**, 15.02	135160.60, 2.79	203167.40, 6.87	192199.00, 4.03	157676.60, **0.33**	156416.60, 0.35	156138.60, 1.30
$7M$	153086.80, 3.26	170324.20, 24.23	**137531.60**, 18.11	137551.00, 3.31	205278.20, 8.12	193788.00, 4.65	157758.80, **0.43**	156537.20, 0.44	156286.60, 1.48
$8M$	153533.80, 3.75	171316.60, 28.14	**139533.20**, 20.89	139577.40, 3.84	206827.60, 9.40	195193.60, 5.28	157816.60, **0.49**	156627.60, 0.50	156381.20, 1.66
$9M$	153893.40, 4.30	172166.60, 31.63	**141277.40**, 24.86	141359.40, 4.39	208314.00, 10.90	196293.40, 5.88	157857.80, **0.52**	156685.00, 0.54	156457.60, 1.81
$10M$	154212.60, 4.83	172896.20, 35.35	**142791.60**, 28.02	142828.00, 4.94	209554.80, 12.25	197356.60, 6.51	157894.00, **0.57**	156736.60, 0.57	156517.80, 1.98

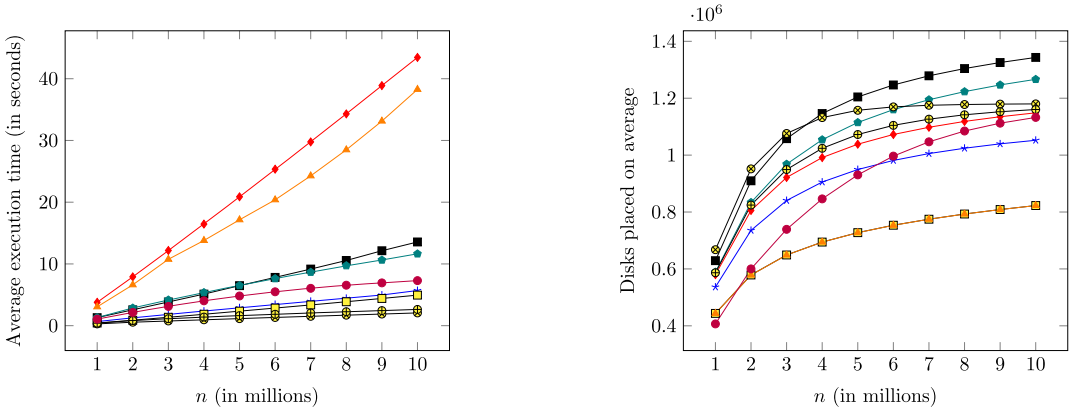
Fig. 21. Points drawn from an annulus. Radius of the outer circle is 10^3 and that of the inner circle is $0.95 \cdot 10^3$. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.





n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	407221.00, 0.61	445479.80, 3.72	**323024.20**, 2.75	323194.40, 0.41	499369.40, 1.20	458617.60, 1.29	526251.20, **0.23**	454723.00, 0.33	324406.40, 0.96
$2M$	512007.60, 1.13	561015.20, 7.82	**393356.60**, 5.97	393431.80, 0.86	646782.60, 2.39	592895.40, 2.57	650388.40, **0.42**	579200.20, 0.61	462901.80, 1.91
$3M$	560012.60, 1.64	612128.20, 12.07	**428795.20**, 9.56	428925.80, 1.33	710463.00, 3.61	658293.60, 3.69	680029.20, **0.60**	632440.40, 0.83	553186.40, 2.72
$4M$	587765.40, 2.15	641600.40, 16.41	**452315.40**, 12.58	452417.80, 1.82	746745.60, 4.83	697954.00, 4.72	687545.20, **0.77**	659224.80, 1.02	612825.60, 3.33
$5M$	605652.60, 2.65	660842.80, 20.75	**469953.60**, 15.52	470092.40, 2.31	771395.20, 6.19	725048.20, 5.70	689482.20, **0.94**	673078.80, 1.21	648709.00, 3.79
$6M$	618291.40, 3.11	674543.80, 25.22	**484090.40**, 18.91	484274.40, 2.82	789528.60, 7.46	744890.80, 6.68	690094.80, **1.10**	680502.60, 1.38	668529.00, 4.16
$7M$	627622.00, 3.66	685274.40, 29.80	**495988.40**, 22.52	496033.80, 3.33	803668.80, 8.79	760212.60, 7.58	690368.20, **1.28**	684540.20, 1.56	678748.80, 4.48
$8M$	634837.80, 4.18	693560.40, 34.27	**506257.20**, 26.42	506343.80, 3.85	815370.00, 10.11	772852.80, 8.50	690497.80, **1.45**	686743.40, 1.72	683831.40, 4.76
$9M$	640536.00, 4.70	700489.20, 38.72	**515259.00**, 30.83	515385.60, 4.37	825395.40, 11.65	783049.00, 9.42	690591.00, **1.62**	688035.20, 1.90	686404.00, 5.04
$10M$	645181.60, 5.24	706588.60, 43.28	**523225.20**, 34.90	523396.40, 4.92	833979.00, 13.11	791785.60, 10.30	690689.60, **1.79**	688691.00, 2.07	687664.40, 5.32

Fig. 22. Points drawn from an annulus. Radius of the outer circle is 10^3 and that of the inner circle is $0.75 \cdot 10^3$. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.



n	G-1991	CCFM-1997	LL-2014	LL-2014-1P	BLMS-2017	DGT-2018	FastCover	FastCover+	FastCover++
$1M$	536307.00, 0.66	579798.80, 3.78	443540.80, 3.05	443613.40, 0.43	629045.40, 1.30	588815.00, 1.31	667558.80, **0.29**	586546.20, 0.40	**406735.80**, 1.06
$2M$	735446.80, 1.27	805474.00, 7.91	**578417.00**, 6.61	578707.20, 0.88	909758.60, 2.57	833603.20, 2.86	951614.60, **0.56**	824422.80, 0.83	600087.40, 2.14
$3M$	840501.40, 1.82	921463.20, 12.18	**648763.40**, 10.75	648995.20, 1.36	1057855.00, 3.84	968272.20, 4.17	1076099.60, **0.76**	949271.00, 1.13	739146.60, 3.12
$4M$	905399.80, 2.35	991357.80, 16.45	**694231.00**, 13.80	694335.40, 1.85	1146178.20, 5.11	1054145.20, 5.37	1131895.40, **0.96**	1023962.00, 1.39	846187.80, 4.02
$5M$	949292.00, 2.90	1038365.60, 20.87	**727326.60**, 17.15	727490.00, 2.34	1204483.00, 6.48	1114684.60, 6.50	1157601.80, **1.14**	1072403.20, 1.62	930522.80, 4.81
$6M$	981310.20, 3.44	1072516.80, 25.34	**753246.40**, 20.38	753555.40, 2.84	1246528.80, 7.82	1159906.60, 7.62	1169522.20, **1.33**	1104615.80, 1.85	996551.60, 5.49
$7M$	1005434.20, 3.94	1097831.80, 29.76	**774569.80**, 24.26	774702.80, 3.36	1278771.00, 9.17	1194620.40, 8.66	1175193.60, **1.51**	1126389.60, 2.06	1046733.40, 6.05
$8M$	1024214.40, 4.43	1118448.80, 34.30	**792774.60**, 28.48	792957.60, 3.87	1304237.20, 10.56	1223142.80, 9.72	1177876.60, **1.70**	1141696.20, 2.26	1084649.20, 6.55
$9M$	1039609.80, 4.97	1134916.60, 38.89	**808606.40**, 33.14	808698.00, 4.41	1325467.80, 12.14	1246706.80, 10.64	1179304.20, **1.88**	1152502.00, 2.44	1112248.60, 6.93
$10M$	1052198.80, 5.69	1148652.20, 43.45	**822610.60**, 38.27	822750.20, 4.95	1343388.00, 13.56	1266389.40, 11.64	1180049.60, **2.07**	1160334.40, 2.63	1132512.60, 7.30

Fig. 23. Points drawn from an annulus. Radius of the outer circle is 10^3 and that of the inner circle is $0.5 \cdot 10^3$. A pair x, y in a cell denotes the average number of disks placed and the average running time in seconds, respectively.

6. A visual comparison of the engineered algorithms

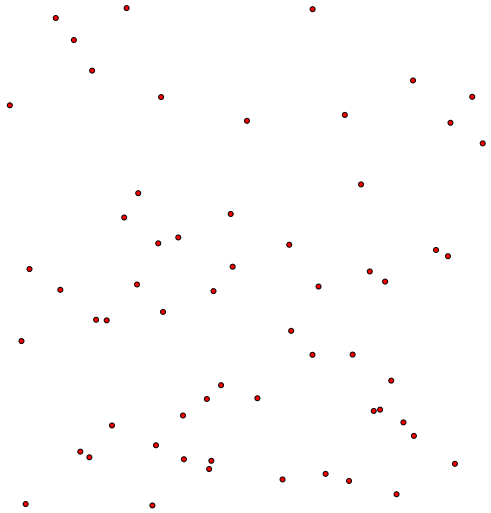


Fig. 24. A 60-element pointset drawn from a 20×20 square using `CGAL::Random_points_in_square_2`.

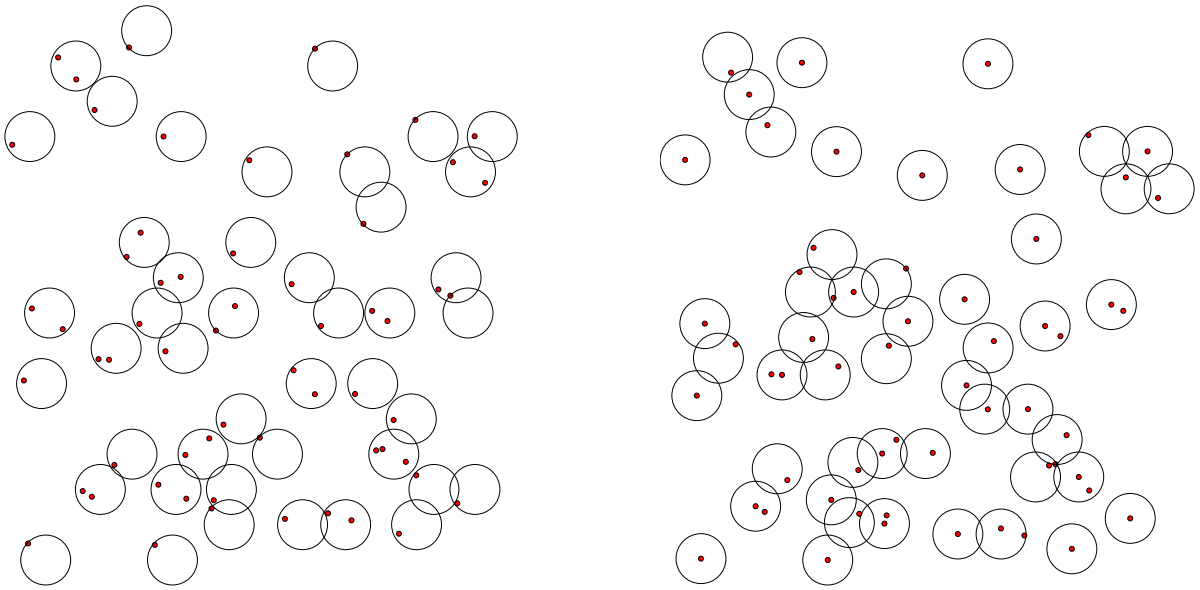


Fig. 25. The outputs produced by G-1991(left) and CCFM-1997(right) on the pointset shown in Fig. 24.

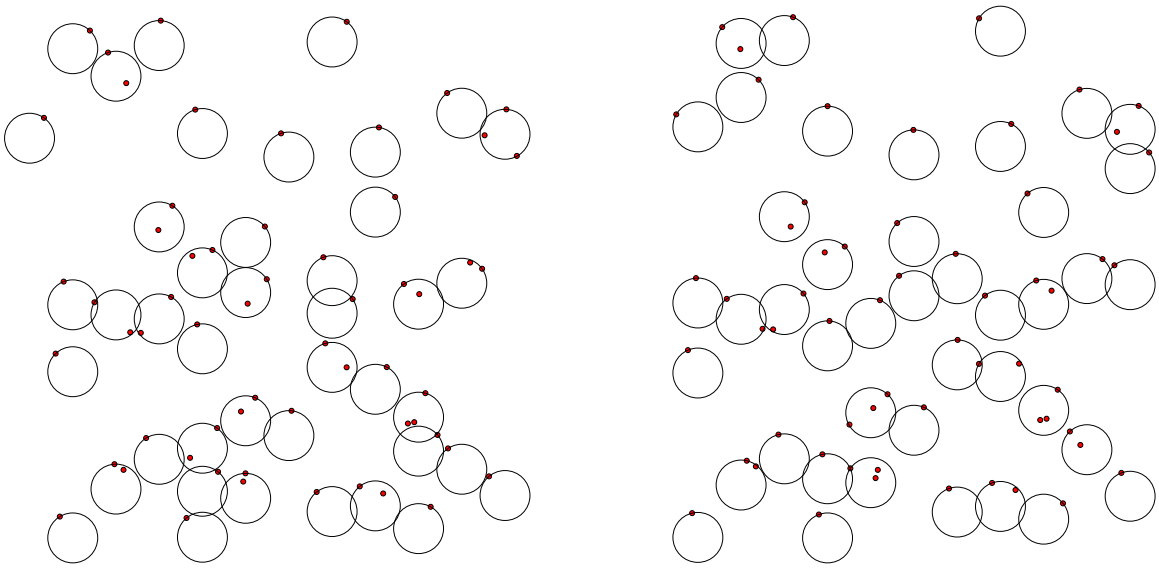


Fig. 26. The outputs produced by LL-2014(left) and LL-2014-1P(right) on the pointset shown in Fig. 24.

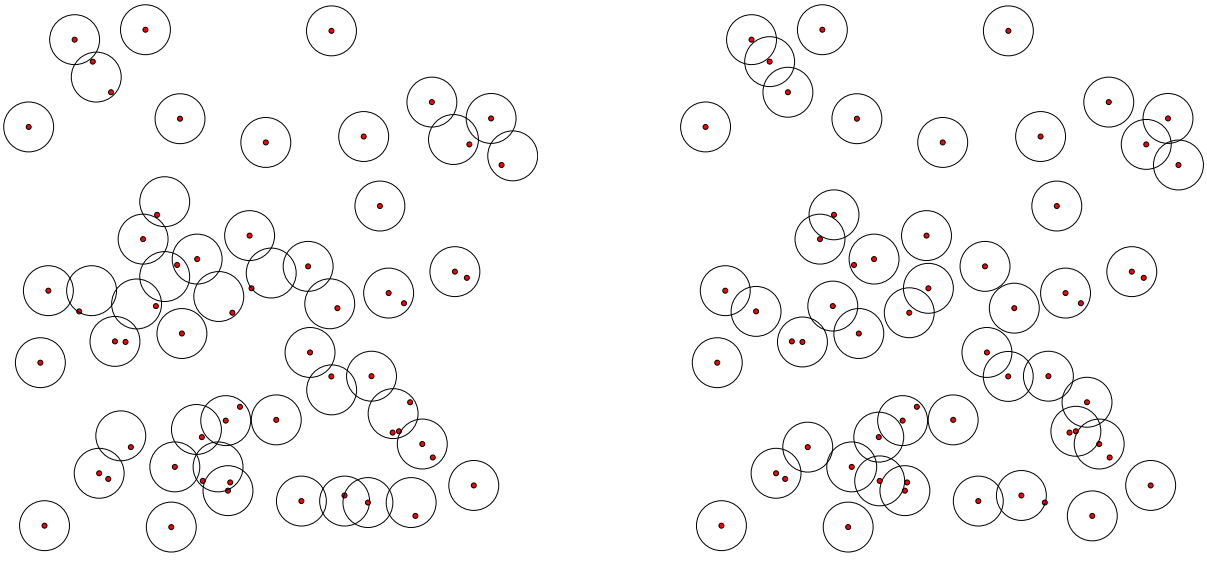


Fig. 27. The outputs produced by BLMS-2017(left) and DGT-2018(right) on the pointset shown in Fig. 24.

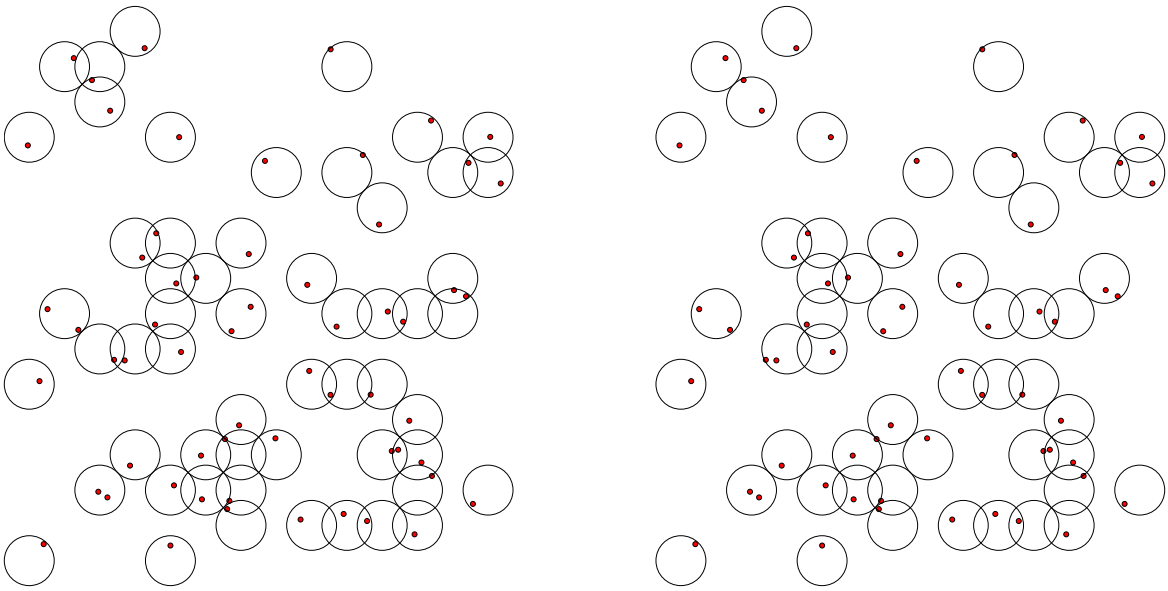


Fig. 28. The outputs produced by FastCover(left) and FastCover+(right) on the pointset shown in Fig. 24.

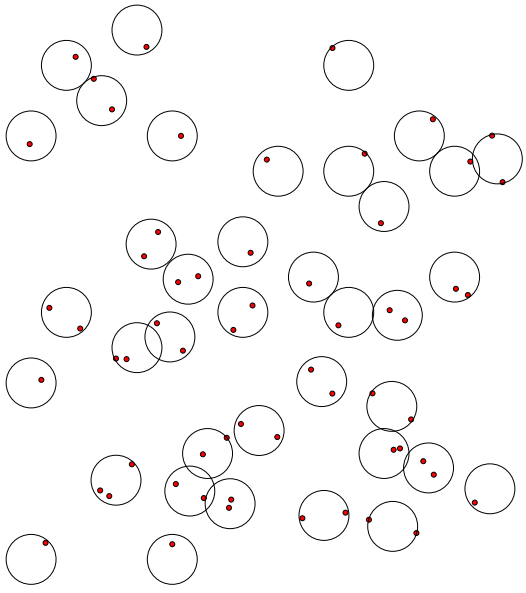


Fig. 29. The output produced by FastCover++ on the pointset shown in Fig. 24.

Acknowledgments. We sincerely thank all the anonymous reviewers of this manuscript for their careful reading and their numerous insightful comments and suggestions for improvements. We especially thank the anonymous reviewer who has generously shared improved implementations with us for the algorithms G-1991, BLMS-2017, DGT-2018, DGT-2018, and CCFM-1997.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The github contains the code for this research.

References

[1] www.math.uwaterloo.ca/tsp/.
[2] P.K. Agarwal, J. Pan, Near-linear algorithms for geometric hitting sets and set covers, in: Proceedings of the Thirtieth Annual Symposium on Computational Geometry, ACM, 2014, p. 271.
[3] G. Aloupis, R.A. Hearn, H. Iwasawa, R. Uehara, Covering points with disjoint unit disks, in: CCCG, 2012, pp. 41–46.
[4] F. Anderson, A. Ghosh, M. Graham, L. Mougeot, D. Wisnosky, Bounded-degree plane geometric spanners in practice, arXiv preprint, arXiv:2205.03204, 2022.
[5] R. Bar-Yehuda, D. Rawitz, A note on multicovering with disks, Comput. Geom. 46 (3) (2013) 394–399.
[6] A. Biniaz, P. Liu, A. Maheshwari, M. Smid, Approximation algorithms for the unit disk cover problem in 2D and 3D, Comput. Geom. 60 (2017) 8–18.
[7] H. Brönnimann, M.T. Goodrich, Almost optimal set covers in finite VC-dimension, Discrete Comput. Geom. 14 (4) (1995) 463–479.
[8] N. Bus, N.H. Mustafa, S. Ray, Practical and efficient algorithms for the geometric hitting set problem, Discrete Appl. Math. 240 (2018) 25–32.
[9] M. Charikar, C. Chekuri, T. Feder, R. Motwani, Incremental clustering and dynamic information retrieval, SIAM J. Comput. 33 (6) (2004) 1417–1440.
[10] B.M. Chazelle, D.T. Lee, On a circle placement problem, Computing 36 (1–2) (1986) 1–16.
[11] G.K. Das, R. Fraser, A. Lóopez-Ortiz, B.G. Nickerson, On the discrete unit disk cover problem, Int. J. Comput. Geom. Appl. 22 (05) (2012) 407–419.
[12] M. De Berg, S. Cabello, S. Har-Peled, Covering many or few points with unit disks, Theory Comput. Syst. 45 (3) (2009) 446–469.
[13] A. Dumitrescu, Computational geometry column 68, ACM SIGACT News 49 (4) (2018) 46–54.
[14] A. Dumitrescu, A. Ghosh, C.D. Tóth, Online unit covering in euclidean space, Theor. Comput. Sci. 809 (2020) 218–230.
[15] R.J. Fowler, Optimal packing and covering in the plane are NP-complete, Inf. Process. Lett. 12 (3) (1981) 133–137.
[16] M. Franceschetti, M. Cook, J. Bruck, A geometric theorem for approximate disk covering algorithms, 2001.
[17] B. Fu, Z. Chen, M. Abdelguerfi, An almost linear time 2.8334-approximation algorithm for the disc covering problem, in: International Conference on Algorithmic Applications in Management, Springer, 2007, pp. 317–326.
[18] A. Ghosh, B. Hicks, R. Shevchenko, Unit disk cover for massive point sets, in: International Symposium on Experimental Algorithms, Springer, 2019, pp. 142–157.
[19] T.F. Gonzalez, Covering a set of points in multidimensional space, Inf. Process. Lett. 40 (4) (1991) 181–188.
[20] Z. Guo, Y. Li, Geometric cover with outliers removal, in: 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
[21] D.S. Hochbaum, W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, J. ACM 32 (1) (1985) 130–136.
[22] H. Kaplan, M.J. Katz, G. Morgenstern, M. Sharir, Optimal cover of points by disks in a simple polygon, SIAM J. Comput. 40 (6) (2011) 1647–1661.
[23] C. Liao, S. Hu, Polynomial time approximation schemes for minimum disk cover problems, J. Comb. Optim. 20 (4) (2010) 399–412.
[24] C. Liaw, P. Liu, R. Reiss, Approximation schemes for covering and packing in the streaming model, in: Canadian Conference on Computational Geometry, 2018.
[25] P. Liu, D. Lu, A fast 25/6-approximation for the minimum unit disk cover problem, arXiv preprint, arXiv:1406.3838, 2014.
[26] R.R. Madireddy, A. Mudgal, A constant–factor approximation algorithm for red–blue set cover with unit disks, Algorithmica (2022) 1–33.
[27] The CGAL Project: CGAL User and Reference Manual, 5.3 edn., CGAL Editorial Board, 2021, https://doc.cgal.org/5.3/Manual/packages.html.