Application of minimal disc covering to the analysis of wildlife signs

M. S. Ridout

School of Mathematics, Statistics and Actuarial Science,
University of Kent, Canterbury, Kent CT2 7NF, U.K.

email: m.s.ridout@kent.ac.uk

Wordcount: 7146

Summary

1. We consider the analysis of data that comprise two-dimensional locations of animal signs, for example footprints, scrapes or scats, where individual animals cannot be distinguished. We ask what is the minimum number of animals that could have generated these signs, assuming that individual animals move within circular home ranges of given radius. This provides an estimated lower bound for the true number of animals present.

2. Calculating the minimum number of animals in this way is equivalent to a well-known problem in computational geometry known as the minimal disc covering problem. Applications arise for example in the location of facilities and in the design of wireless networks but, as far as we are aware, this is the first application to ecology.

3. Although the problem is well-known, no algorithm is known that is guaranteed to provide a minimal disc covering of a set of locations. We describe randomized algorithms for this problem. The algorithms aim to generate coverings with a small number of discs. By running multiple times we can generate minimal or close-to-minimal coverings. The performance of the algorithms is demonstrated using various artificial data sets.

4. The methodology is applied to tiger pugmark data from a survey of the Kerinci Seblat National Park in west-central Sumatra. We investigate how the estimated minimum number of tigers changes with the assumed radius of the home range and demonstrate that, even with modest computational effort, it is usually possible to obtain an answer that is close to the true minimum.

5. In practice, the methodology is only likely to be useful for rare species, but here we argue that an estimate of the minimum number of animals that must be present is of some interest. Estimates of the true number of animals present are of course of greater interest, but these inevitably require more sophisticated methods of analysis based on careful assumptions about the sampling method. We believe that estimates of minimum number, interpreted carefully, can provide a useful supplement to estimates of true population size.

Keywords
Clustering; geometric optimization; home range; minimum enclosing circle; randomised algorithm; Sumatran tiger

Introduction

The minimal disc covering problem is to find the smallest number of discs of radius r that are needed to cover a set of n points in the plane. There are no constraints on the positioning of the discs, in particular they are allowed to overlap.

In the application that led to our interest in this problem, the points are the locations of signs, such as footprints, scrapes, nests or scats, left by a species of wildlife. We ask what is the minimum number of animals that could have generated these signs, assuming that individual animals move within a circular home range of radius r.

There are no known algorithms that are guaranteed to find a solution to the minimal disc covering algorithm. The main contribution of this paper is to present new randomized algorithms that find reasonable coverings that are not necessarily minimal. By running the algorithms multiple times and selecting the best solution, we can generally obtain results close to the true minimum in reasonable computation time, at least for problems involving up to a few hundred points. These algorithms are conceptually simple and relatively straightforward to program; in the Supporting Materials we provide code in R.

We apply the methodology to a data set comprising the locations of Sumatran tiger pugmarks, to estimate the minimum number of tigers present. The dependence of the estimate on the assumed radius of the home range is investigated. Whilst there is certainly more interest in estimating the actual number of tigers present, this can only be done using sophisticated analyses that rely on several assumptions (e.g. Guillera-Arroita *et al.*, 2011; Guillera-Arroita *et al.*, 2012). We believe that the methodology presented here offers a simple and intuitive complement to such analyses.

There are many variants of minimal disc covering problem. For example, the Euclidean p-centre problem fixes the number of discs and seeks the minimum disc radius that is needed to obtain a covering. Or one may fix both the number of discs and their radius and seek the maximum number of points that can be covered. Agarwal and Sharir (1998) give a general overview of geometric optimization problems of this type. In general, these problems are difficult computational problems, with no known algorithms that are guaranteed to find the best solution. We briefly review previous algorithms for the minimal disc problem in the Methods section before describing the new algorithms. The performance of these algorithms is then investigated using artificial data before application to the tiger pugmark data.

Methods

In this section, we first describe a simpler geometric problem, that of finding the smallest circle that encloses a set of points. This well-studied problem forms the basic building block of our algorithms.

We then consider the minimal disc covering problem itself, which is known to be 'hard', specifically strongly NP-complete (Fowler *et al.*, 1981). We give a short overview of some approximate algorithms for the minimal disc covering problem that run in polynomial time and that guarantee to find a number of discs that is no more than q times the true minimum, where $q > 1$ is termed the *approximation ratio*. However, for the type of application that we have in mind, having a guaranteed worst-case approximation ratio is only useful if this ratio is close to one. Unfortunately, none of the known algorithms can guarantee this in reasonable computational time. Moreover, several of the algorithms described appear to be quite difficult to implement.

We therefore explore alternative approaches. We discuss a greedy algorithm that is conceptually simple, but demonstrate that this is not always effective. Instead, we introduce some heuristic alternatives that are reasonably straightforward to program. Specifically we discuss two classes of randomized algorithms that we term `grow` and `shrink`.

Note that a disc covering gives a clustering of the n points where the clusters may overlap. We are concerned only with identifying the points that are covered by each disc, not with the precise geometric location of the disc, which is uniquely identified only if points cannot be covered by a disc of smaller radius.

The minimal enclosing circle problem

The minimal enclosing circle problem is to find the circle of smallest radius that encloses a set of $n(\geq 2)$ points, $\mathcal{P}$, in the plane. The minimal enclosing circle is unique and passes through at least two of the points of $\mathcal{P}$. If it passes through exactly two points, the centre of the circle is the midpoint of the line segment joining these two points. Otherwise, the circle is determined by any three points on the circumference.

To find the minimal enclosing circle, we have used the algorithm of Elzinga and Hearn (1972), which builds progressively larger circles until all points are enclosed. Our implementation is a translation into R of the Fortran code provided by Hearn, Vijay and Nickel (1995). Whilst calling the Fortran code directly from R would be expected to give a faster implementation, we wanted to maintain transparency of the code at the R level. To increase the speed of the algorithm we have restricted the search to consider only the convex hull of $\mathcal{P}$, obtained using the R function `chull`, and we have also taken the initial circle to be the circle whose diameter is the line segment connecting the two most distant points of the hull.

The Elzinga-Hearn algorithm is straightforward to implement and the authors report that in test cases its run time increased approximately linearly with n. However, its complexity is at least $O(n^2)$ (Drezner and Shelah, 1987). More efficent algorithms are known, including the $O(n)$ algorithm of Megiddo (1983) and the recursive randomized algorithm of Welzl (1991), for which the expected run time is also $O(n)$, but we have not investigated whether these would lead to significant improvements in computational time for our algorithms.

Algorithms for the minimal disc covering problem

Let $\mathcal{P}$ denote the set of n points for which we require a minimal disc covering. Let $\mathrm{d}(P, Q)$ denote the Euclidean distance between points P and Q in $\mathcal{P}$ and define the points to be *r-neighbours* if $\mathrm{d}(P, Q) \leq 2r$. Note that every point is an r-neighbour of itself.

Points that are not r-neighbours cannot be covered by a disc of radius r and therefore a useful first step is to partition $\mathcal{P}$ into disjoint subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, such that no two points that are in different subsets are r-neighbours. The minimal disc covering problem can then be addressed separately for each subset of points in turn. The partition is easily generated by running a single linkage cluster analysis and cutting the dendrogram at a height of $2r$. We therefore focus our attention on sets $\mathcal{P}$ for which every point has at least one r-neighbour.

Because the minimal disc covering problem is NP-complete, attention has focused on polynomial-time approximation schemes that guarantee to achieve a specified approximation ratio. We now give a brief overview of algorithms of this type.

Polynomial-time approximation algorithms

Hochbaum & Maass (1985) introduced the first algorithm of this type, with $q = (1+1/k)^2$, where k is a positive integer chosen by the user. However, whilst the running time is polynomial, the polynomial is of high degree, $d = 2\lceil k\sqrt{2}\rceil^2 + 1$, where $\lceil x \rceil$ is the smallest integer not less than x, giving for example $d = 9, 19, 51$ for $k = 1, 2, 3$, respectively. This makes the algorithm impractical for even moderate values of n. Nonetheless, the shifting strategy proposed by Hochbaum and Maass (1985) has been exploited in several subsequent algorithms.

The Hochbaum and Maass algorithm is expensive because it repeatedly solves the covering problem exactly on small areas of the plane. Franceschetti *et al.* (2001) give a similar algorithm but restrict the disc centres to lie on a grid. This leads to a larger approximation ratio $q = c(1 + 1/k)^2$ $(c > 1)$, but the running time is $O(Cn)$, where C depends upon c. Unfortunately, whilst the running time increases only linearly with n, the constant C is prohibitively large for even modest values of q. For example, Xiao *et al.* (2004) state that $q = 6$ would require $C \approx 10^{16}$. Franceschetti *et al.* (2001) review other algorithms and provide a useful table (their Table 1) of their approximation ratios and running times. More recently, Fu *et al.* (2007) have developed an $O(n(\log n)^2(\log\log n)^2)$ algorithm, which also uses the shifting strategy, and gives $q = 2.8334$, but no details are given of actual run times.

A greedy algorithm

One natural approach to the minimal disc covering problem is to use a greedy algorithm that successively positions discs to cover the largest possible number of points that have not already been covered, until no points remain. However, it is easy to contruct examples for which this algorithm fails to generate a minimal solution. Figure 1a shows a simple example involving 6 points. The greedy algorithm leads to a 3-disc covering, but clearly the optimal solution requires just 2 discs.

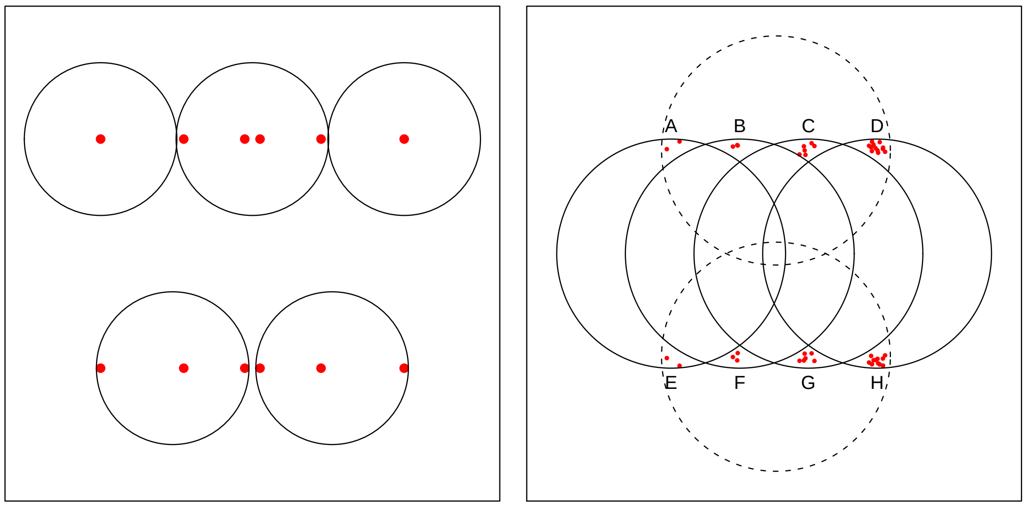

Figure 1. Failure of the greedy algorithm. Panel (a) shows six points that can be covered by two discs, but for which the greedy algorithm gives three discs. Panel (b) shows a construction due to Franceschetti *et al.* (2001) that is described in the text.

More generally, Franceschetti *et al.* (2001) note that the approximation ratio of the greedy algorithm is bounded by $1 + \log n$ and give an explicit construction involving $n = 3 \times 2^k - 2$ points for which the approximation ratio is $k/2$, for any positive integer k. This construction is illustrated for $k = 4$ in Figure 1b. Clusters of $2, 3, 6$ and 12 points are placed at points A, B, C and D and at points E, F, G and H. The points can be covered by two discs, as indicated by the dashed circles, each of which covers 23 points. However, the greedy algorithm begins by covering the 24 points in the clusters at D and H. The clusters at C and G are positioned just too far from D and H to also be covered by the first greedy disc. The greedy algorithm then generates the remaining three solid circles, moving from right to left.

Heuristic algorithms for the minimal disc covering problem

We consider algorithms that, like the greedy algorithm, sequentially identify points that have not yet been covered and find a disc that covers the point identified. However, we select points randomly rather than in a greedy way. In practice, we run the algorithms several times, and select the best solution obtained.

We therefore consider a function `cover(`$\mathcal{P}$`)` which returns a set of discs that cover the n points in the set $\mathcal{P}$. This function has the following generic form:

```
cover ← function($\mathcal{P}$)
    $\mathcal{U} \leftarrow \mathcal{P}$                          # $\mathcal{U}$ is the set of points not yet covered
    $\mathcal{C} \leftarrow \emptyset$                          # $\mathcal{C}$ is the set of covering discs
    while ($\mathcal{U} \neq \emptyset$) begin
        $P \leftarrow$ choose.point($\mathcal{U}$)       # select a point $P \in \mathcal{U}$ that is not yet covered
        $D \leftarrow$ disc.cover($P, r$)        # find a disc of radius $r$ that covers $P$
        $\mathcal{C} \leftarrow \mathcal{C} \cup D$                  # add this disc to the set of covering discs
        $\mathcal{D} \leftarrow$ cover.set($D$)          # identify the points covered by this disc
        $\mathcal{U} \leftarrow \mathcal{U} \backslash \mathcal{D}$                   # remove these points from uncovered set
    end
    return $\mathcal{C}$
end
```

The function `cover.set()`, which identifies the points that are covered by a specified disc, is a straightforward deterministic function. We shall assume that the function `choose.point()`, which selects a point from a set of points, does so by simple random sampling, though in principle a wide range of alternative deterministic or stochastic selection mechanisms could be considered; for example, selection could favour points that lie on the convex hull of $\mathcal{U}$. Note that the greedy algorithm selects P so that the size of the set D is maximized, thus requiring a form of look-ahead.

The remaining function, `disc.cover()` identifies a disc D of radius r that covers the point P. The greedy algorithm selects the disc that covers the greatest number of points that have not yet been covered. We consider two less greedy approaches, giving algorithms that we refer to as `grow` and `shrink`. Both algorithms compute the disc D by generating the set of points, $\mathcal{D}$, that are covered by the disc. Clearly $\mathcal{D}$ must be a subset of the set of r-neighbours of P. We let $\text{rad}(\mathcal{D})$ denote the radius of the minimum enclosing circle of $\mathcal{D}$.

Algorithm `grow` sets $\mathcal{D} = \{P\}$ initially and adds points sequentially. At each stage, the point to be added is chosen from the set of points that could be added whilst still maintaining $\text{rad}(\mathcal{D}) \leq r$. The algorithm terminates when there are no such points.

Algorithm `drop` sets $\mathcal{D}$ to be the full set of r-neighbours of P initially. If $\text{rad}(\mathcal{D}) > r$, a point on the circumference of this circle is selected and removed from $\mathcal{D}$. The point P itself is never dropped in this way. This process is repeated until $\text{rad}(\mathcal{D}) \leq r$.

We now discuss each of these algorithms in more detail.

Algorithm `grow`

Figure 2 shows a single run of the algorithm `grow` to cover the point P, which is shown by an asterisk in the figure. Other points that have been included in $\mathcal{D}$ are shown as solid circles. The remaining points are shown as open circles if they are still eligible to the included in $\mathcal{D}$ and as crosses otherwise. Panel (a) shows the point P and its r-neighbours. Initially, the algorithm picks one of the r-neighbours at random, in this instance the nearest neighbour of P. The minimum enclosing circle of these two points is shown in panel (b) as a solid circle; the dashed circle has the same centre, but radius r. At this point, some of the original r-neighbours of P can no longer be covered by a disc that covers both P and the first point selected and therefore become ineligible for inclusion in $\mathcal{D}$. In subsequent panels, additional points are chosen at random from those that remain eligible and added to $\mathcal{D}$, until no eligible points remain. Note that the point indicated by the arrow in panel (c), which is covered by the current disc, but is not part of the set $\mathcal{D}$, is ultimately not covered, as shown in panel (g).

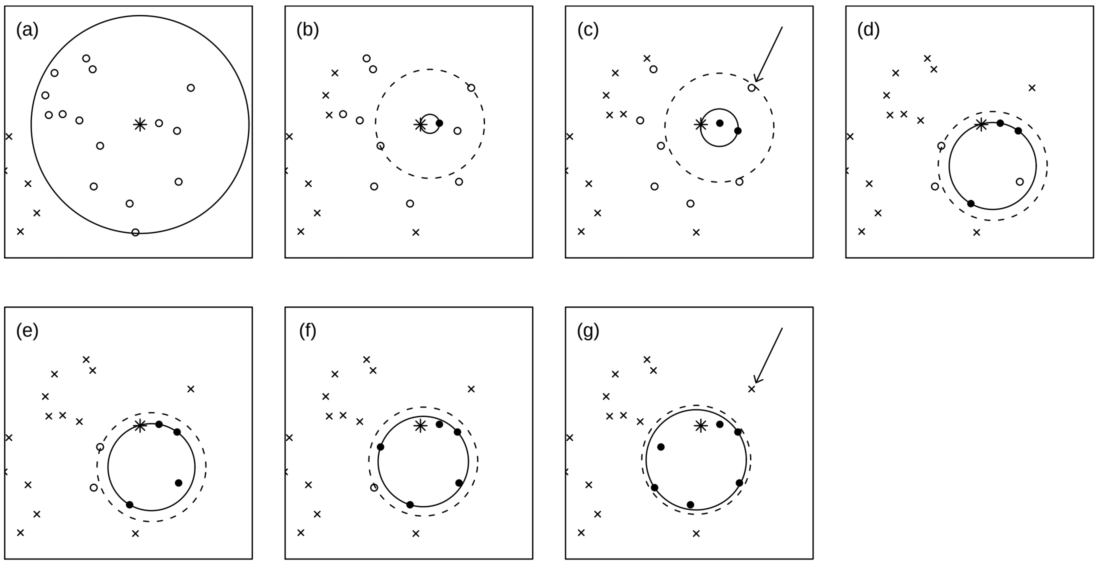

Figure 2. Example of the algorithm `grow`. The point P is denoted by an asterisk. Points that have been selected by the algorithm are shown as filled circles, other points are shown as open circles if they are eligible to be selected at the next stage and as crosses otherwise.

Variants of the algorithm can arise by modifying the way in which a point is selected from amongst the eligible points at each stage. Specifically, if the eligible points are $Q_1, \ldots, Q_k$, we select Q_j with probability proportional to $\mathrm{d}(P, Q_j)^\alpha$, where α is a real parameter. We denote the resulting algorithm by `grow(`α`)`. Setting $\alpha = 0$ gives simple random sampling, whereas positive (negative) values of α preferentially select points at are far from (close to) P. Intuitively, one might expect that negative values of α would tend to increase the number of points covered and therefore reduce the number of covering discs. In the limit as $\alpha \to -\infty$, the algorithm selects the nearest eligible point to P, or, if there are several such points, chooses randomly between them. In Figure 2, the first two points selected (panels b and c) are the nearest eligible points, but the third point selected is not the nearest eligible point (panel d). For convenience we use the term `grow.random` as an alternative to `grow(`0`)` and `grow.nearest` as an alternative to `grow(`$-\infty$`)`.

Algorithm `shrink`

Figure 3 shows a single run of the algorithm `shrink` to cover the point P. Initially, in panel (a), all r-neighbours of P are shown as filled circles. Points that lie on the circumference of the minimum enclosing circle, excluding P itself, are selected at random and dropped. Note that points that are dropped in this way may nonetheless reappear in the minimum enclosing circle of the remaining points at a later stage. This is illustrated by the point marked with an arrow in panels (f) and (g).

Eventually, at panel (k), we arrive at a minimum enclosing circle with radius that does not exceed r; this is indicated by the solid circle in panel (k). The dashed circle in panel (k) has the same centre, but the radius is now r. By using the full allowed radius, we capture the lowermost of the 3 points at the top left of the panel (shown as open circles), which were eliminated previously. However, it is also possible to move the circle in an attempt to capture additional points, a process that we term *realignment.* The result of this is shown as the solid circle in panel (l). All three points at the top left of the panel (now shown as closed circles) are covered by the realigned disc. In general, several distinct realignments may be possible. Realigment is implemented by running the `grow(`α`)` algorithm, starting from the endpoint of the `shrink` algorithm.

Again, variants of the algorithm can arise by modifying the way in which the point to be dropped is selected from amongst the points on the circumference of the minimum enclosing circle. If the eligible points are $Q_1, \ldots, Q_k$, we select Q_j with probability proportional to $\mathrm{d}(P, Q_j)^\beta$, where β is a real parameter. Setting $\beta = 0$ gives simple random sampling, whereas positive (negative) values of β preferentially select points at are far from (close to) P. Intuitively, one might expect that positive values of β would tend to increase the number of points covered and therefore reduce the number of covering discs. In the limit as $\beta \to \infty$, the algorithm always drops the point furthest from P, or, if there are several such points, chooses randomly between them. We denote the resulting algorithm by `shrink(`α, β`)` and alternatively use the terms `shrink.random` and `shrink.furthest` as synonyms for `shrink(`$-\infty, 0$`)` and `shrink(`$-\infty, \infty$`)` respectively. Note that in both of these latter algorithms, realignment selects any additional points in order of their closeness to P, since $\alpha = -\infty$.

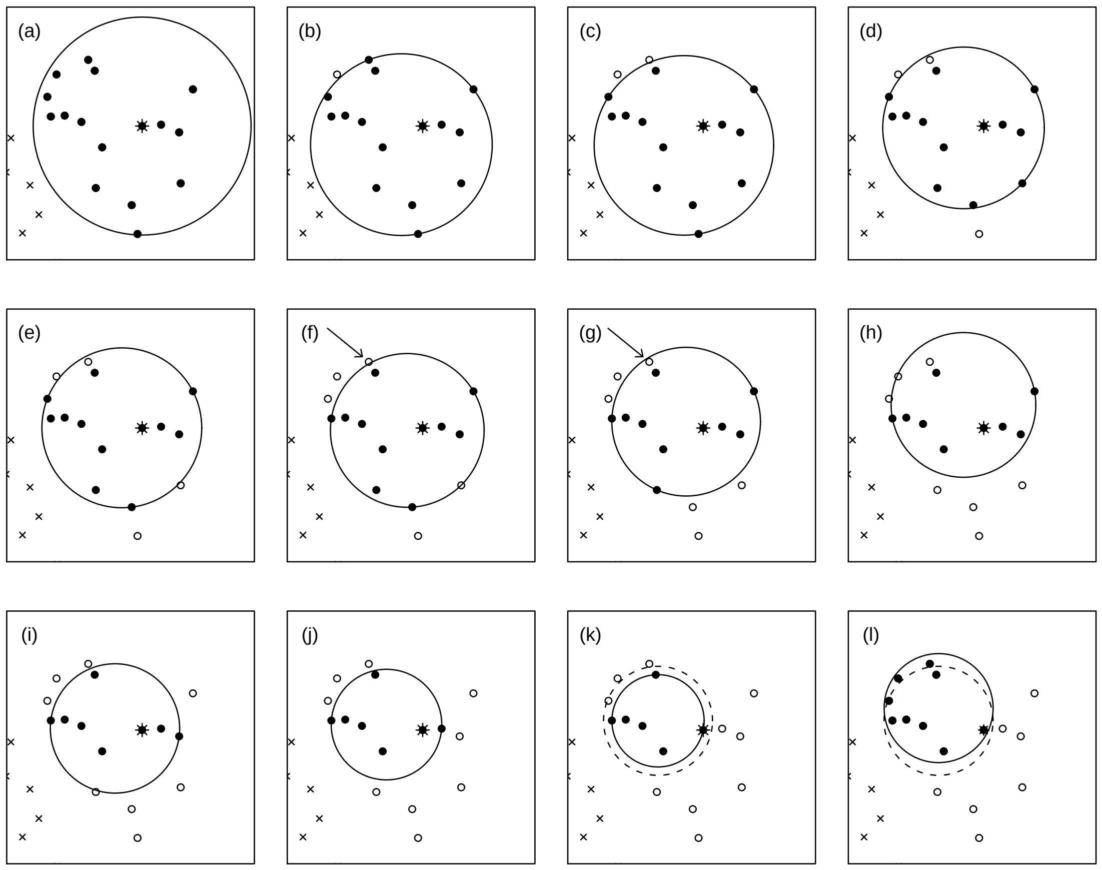

Figure 3. Example of the algorithm `shrink`. The point P is denoted by an asterisk. Points that have been eliminated are shown by open circles. Points that are not r-neighbours of P are shown as crosses.

Overall algorithm summary

The complete algorithm proceeds as follows. First we partition the set $\mathcal{P}$ into disjoint subsets $\mathcal{P}_1, \ldots, \mathcal{P}_m$, as outlined above. For subset $\mathcal{P}_j$ we obtain a set of covering discs $\mathcal{C}_j$ as the smallest set that results from K_j calls to the function `cover(`$\mathcal{P}_j$`)`. The disc covering for the points in $\mathcal{P}$ is then $\bigcup_{j=1}^{m} \mathcal{C}_j$.

We shall assume that $K_j \equiv K$, for all j, though more generally one might wish to choose K_j to be dependent on the size of the set $\mathcal{P}_j$. Note, however, that if a covering consists of either one or two discs, then this must be optimal and further calls to the function `cover()` are unnecessary.

In the following, when we refer to a specific algorithm, such as `grow.nearest`, we mean that this is the algorithm used to implement the function `disc.cover()` within the function `cover()`.

Because of their random nature, there is a non-zero probability, that the algorithms `grow(`α`)` and `shrink(`α, β`)` will generate a minimal disc covering of the points in $\mathcal{P}_j$, for any finite values of α and β. Thus as $K \to \infty$, the probability that a minimal covering will have been identified approaches one.

The limiting algorithms `grow.nearest` and `shrink.furthest` generate a disc to cover the point P in a deterministic fashion, unless there are points equidistant for P, and it may be possible to construct examples in which these algorithms are unable to find a minimal covering, though we have not been able to identify any such examples. Of course, in these algorithms, the starting point P is still selected at random from those available at each stage.

Results

We begin by applying the algorithms to artificially generated data. We compare the algorithms `grow(`α`)` and `shrink(`α, β`)` for different choices of α and β. We show that there is no universally optimal choice for the parameters α and β, using the examples shown in Figure 1, and study the performance of the algorithms for random, regular and clustered point patterns. Finally, we investigate how computing time increases with the number of points, n, for random distributions of points.

We then apply the methodology to the Sumatran tiger pugmark data.

Failures of the greedy algorithm revisited

Consider first the 6-point example shown in Figure 1a. Simple probability calculations show that algorithm `grow.random` finds the optimal covering with probability 5/9 whereas for `shrink.random` the probability is 1/2. On the other hand, the algorithms `grow.nearest` and `shrink.furthest` find the optimal solution only if the point chosen initially is either the leftmost or rightmost point, which occurs with probability 1/3.

Conversely, for the example shown in the right hand panel of Figure 1, the algorithms `grow.nearest` and `shrink.furthest` both find the optimal solution with probability one. The corresponding probabilities for the algorithms `grow.random` and `shrink.random` are more difficult to compute exactly for this example but simulations indicate that they are approximately 0.29 and 0.14 respectively.

These contrasting examples show that there is no universally optimal choice for the parameters α and β in algorithms `grow(`α`)` and `shrink(`α, β`)`. However, the calculations also show that, whilst the greedy algorithm is unsuccessful for both of these examples, any of the algorithms considered in this paper will find the optimal solution with probability close to one, provided that K is not too small; for example, $K = 30$ will ensure a probability of at least 0.99 even when using the poorly-performing `shrink.random` algorithm in the second example.

Performance for simulated point patterns

Next we compared the performance of the algorithms for three types of point pattern simulated on the unit square.

Regular: 100 points from a simple sequential inhibition process with minimum inter-point distance of 0.075. The process is simulated by generating points randomly within the unit square, one at a time, and accepting points that are a distance of at least 0.075 from the nearest existing point. This is repeated until 100 points have been accepted.

Random: 100 points positioned randomly with the unit square.

Clustered: Simulation of a Matérn process with parents generated as a Poisson process of rate 10 per unit area. The number of offspring per parent is a Poisson variable with mean 10 and offspring are generated uniformly on a circle of radius 0.1 centred at the parent point. Parents are not included in the final set of points. Thus, the expected number of points per realization is 100 (= 10 × 10).

For a fuller description of these processes see, for example, Illian *et al.* (2008).

We considered three choices of disc radius (0.05, 0.1, 0.2). For each of these we generated 400 realisations of each type of pattern and ran several versions of the `grow` and `shrink` algorithms with $K = 5$. A small value of K was chosen to accentuate differences between algorithms. An example set of simulations with $r = 0.1$ is shown in Figure 4.

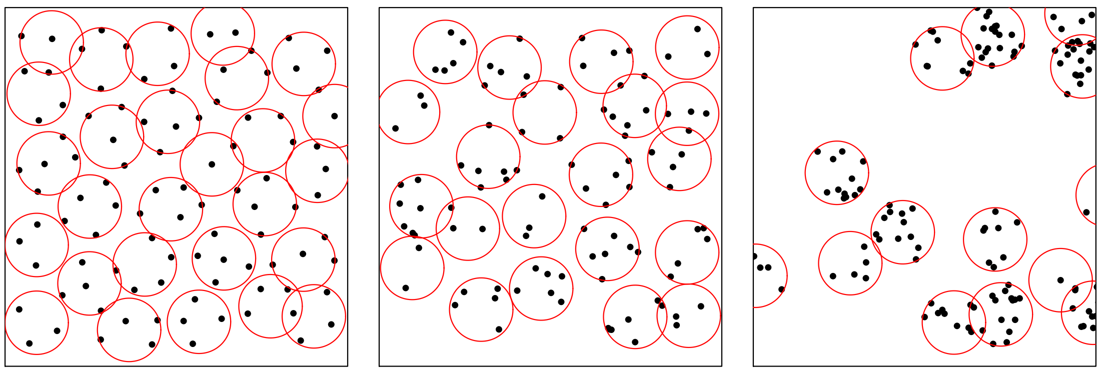

Figure 4. Example of simulated point patterns. From left to right, the patterns are regular, random and clustered.

Table 1 shows results for five algorithms. The first algorithm was `shrink.random` *without* realignment. The other algorithms were `shrink.random`, `shrink.furthest`, `grow.random` and `grow.nearest`. All five algorithms give similar results for the inhibition pattern with $r = 0.05$. Elsewhere, there is a clear benefit of including the realignment step in `shrink.random`. Furthermore, algorithms `grow.nearest` and `shrink.furthest` perform similarly to each other and better than algorithms `grow.random` and `shrink.random`. Although the results are not shown in Table 1, algorithms `grow(1)` and `grow(2)` were also run and, as might be anticipated, gave results intermediate between those of `grow.random` and `grow.nearest`. Similarly, algorithms `shrink(`$-\infty, 1$`)` and `shrink(`$-\infty, 2$`)` gave results intermediate between those of `shrink.random` and `shrink.furthest`.

To investigate the performance of algorithms `grow.nearest` and `shrink.furthest` in more detail, we generated 10 patterns of each type and ran the algorithms with $K = 250$. The two algorithms gave very similar results. Of the total of 90 patterns, 69 gave minimal coverings with the same number of discs, for a further 10 `shrink.furthest` found a covering with one fewer disc and for the remaining 11 `grow.nearest` found a covering with one fewer disc. Table 2 shows the mean minimum number of discs and the median number of iterations required to first encounter this minimum.

Overall, the results of this section suggest that the most useful algorithms are `shrink.furthest` and `grow.nearest`. Although these algorithms generate covering discs in different ways, they consistently gave similar results for a range of simulated point patterns.

Computational time

We simulated additional random point patterns to compare run times for `shrink.furthest` and `grow.nearest` and to investigate the effect of the number of points, n. We generated 25 sets of $n = 25 \times 2^j$ points on the unit square for $j = 0, \ldots, 5$ and applied the algorithms with $K = 25$ and $r = 0.05, 0.1$ or 0.2, as above. Run times were similar for the two algorithms for $n = 25$ but run times for `grow.nearest` increased more rapidly with n than those for `shrink.furthest` and for $n = 800$ `grow.nearest` took up to one third longer than `shrink.furthest`. For $n \geq 100$, run times were approximately proportional to n^θ, where θ was dependent on r as well as on the choice of algorithm. For `shrink.furthest`, estimates of θ from simple linear regression of the logarithm of run time on $\log(n)$ were 1.62 (0.010), 1.40 (0.003) and 1.31 (0.003), for $r = 0.05, 0.1$ and 0.2, respectively; figures in parentheses are standard errors. For `grow.nearest`, the corresponding estimates were 1.71 (0.012), 1.47 (0.005) and 1.36 (0.006).

In these simulations, the density of points per unit area is increasing. We also ran an identical set of simulations, except that the points were generated on a square with area proportional to n, so that the density remained constant. Although `grow.nearest` was again a little slower than `shrink.furthest` for larger values of n, differences in run times were less than 20%. For $n \geq 100$, run times were again approximately proportional to n^θ. For `shrink.furthest`, estimates of θ were 1.63 (0.048), 1.15 (0.010) and 1.22 (0.006), for $r = 0.05, 0.1$ and 0.2, respectively. For `grow.nearest`, the corresponding estimates were very similar, namely 1.64 (0.055), 1.16 (0.011) and 1.22 (0.008).

Application: Sumatran tiger pugmarks

The Sumatran tiger, *Panthera tigris sumatrae*, is critically endangered due to factors such as habitat fragmentation, demand for tiger body parts and retaliatory killing of tigers following conflicts with humans (Linkie *et al.*, 2003; Linkie *et al.*, 2006; Wibisono and Pusparini, 2010). Figure 5a shows the locations of tiger pugmarks detected in a survey of the Kerinci Seblat National Park in west-central Sumatra, an area that includes prime habitat for tigers. This is part of a larger survey (Linkie *et al.*, 2010).

The sampling area was divided into $17 \times 17\,\text{km}^2$ grid cells, on the basis that male tiger home ranges are not expected to exceed $250\,\text{km}^2$. Within each grid cell, a survey team walked a transect of approximately 40km, and recorded GPS locations of sites where pugmarks were observed; these are shown as dots in Figure 5a. Transect length was proportional to forest habitat coverage within the grid cell and ranged from 4km (10% coverage, the minimum) to 40km (100% coverage). There are 89 grid cells and a total of 257 pugmarks were identified; 66 cells had at least one footprint.

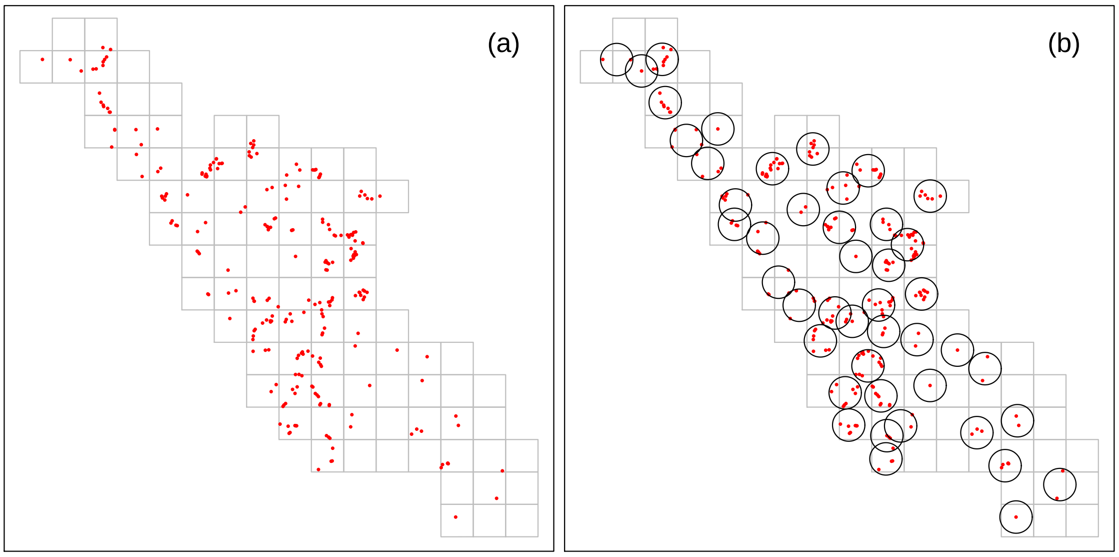

Figure 5. Locations of tiger pugmarks along tracks with $17 \times 17\,\text{km}^2$ grid squares. Panel (b) shows a covering by 45 discs of radius 8.5 km.

A key interest is to estimate the population density of tigers from such data. However, this is fraught with difficulties due, for example, to the incomplete sampling of the region, the irregular nature of the transects, the likelihood that the survey teams will fail to detect some signs and the difficulty of determining how many tigers might have generated a particular set of signs, since tigers cannot be identified individually from their pugmarks. Moreover, the tracks are not 'random' but follow routes through the rugged terrain along which tiger pugmarks might be expected to be found. Thus, estimation of population density will inevitably require complex statistical modelling that incorporates ecologically realistic assumptions Guillera-Arroita *et al.* (2011, 2012).

Here, however, we ask a different question, what is the minimum number of tigers that could have produced the observed signs, assuming that individual tigers occupy a circular home range of maximum radius r? This is exactly the minimum disc covering problem. Table 3 compares the performance of various algorithms for discs of diameter ranging from 8.5 km, which just fits inside a grid square, to 12.5 km, which is large enough to cover a grid square. For each algorithm we set $K = 100$ and ran the algorithm 20 times. We identified a 'true' minimum number of discs using more extensive runs of two of the algorithms, `grow.nearest` and `shrink.furthest`, with $K = 25000$, as discussed below.

The table shows that the algorithms `grow.nearest` and `shrink.furthest` again performed similarly and outperformed the algorithms `grow.random` and `shrink.random`. Typically, the latter algorithms found solutions with one or two fewer discs, except for $r = 8.5$ km. For this radius, a covering can be obtained simply by placing a disc in each of the 66 occupied grid squares. However, the minimal covering requires just 45 discs a reduction of almost one third. Figure 5b shows one such covering, obtained by `shrink.furthest`. For larger values of r, the algorithms `grow.nearest` and `shrink.furthest` usually failed to identify the true minimum with $K = 100$, but the number of discs never exceeded the 'true' minimum by more than two, giving a worst-case approximation ratio of $32/30 = 1.067$.

Figure 6a shows a 30-disc covering obtained by algorithm `grow.nearest`. Here, and in other plots, to resolve the ambiguity in where to position the discs, each disc is centred at the centre of the minimal enclosing circle of the set of points that is covered by the disc. These minimal enclosing circles are shown in Figure 6b. The median radius is 11.3 km and 8 of the 30 circles have radius less than 10 km. This sort of supplementary information may be useful in interpreting the covering, for example if it is thought that most animals would have a home range smaller than 12.5 km in radius.

Table 4 gives more detail about the 'true' minimal coverings obtained by the algorithms `shrink.furthest` and `grow.nearest` with $K = 25000$. The table shows the number of subsets, m, and the number of discs needed to cover the largest subset. In this example, the overall performance of the algorithm was always determined by the performance for this largest subset and the number of times that the minimum was encountered for this subset is shown, along with the number of runs needed to first obtain the 'true' minimum for the full set of points $\mathcal{P}$.

The algorithms `shrink.furthest` and `grow.nearest` found the same number of covering discs for each value of r, except though for $r = 11.5$, where `shrink.furthest` failed to find a 34-disc covering overall. We therefore ran a further simulation with K=250000. Table 5 shows the frequencies with which different solutions were found in the combined set of 275000 runs for $r = 11.5$ are shown in Table 5. Both algorithms struggle to find the 'optimal' covering for this example. Figure 7 compares the performance of the two algorithms in terms for values of K up to 1000, treating the empirical frequency distributions in Table 5 as if they were the true distributions. Figure 7a shows the

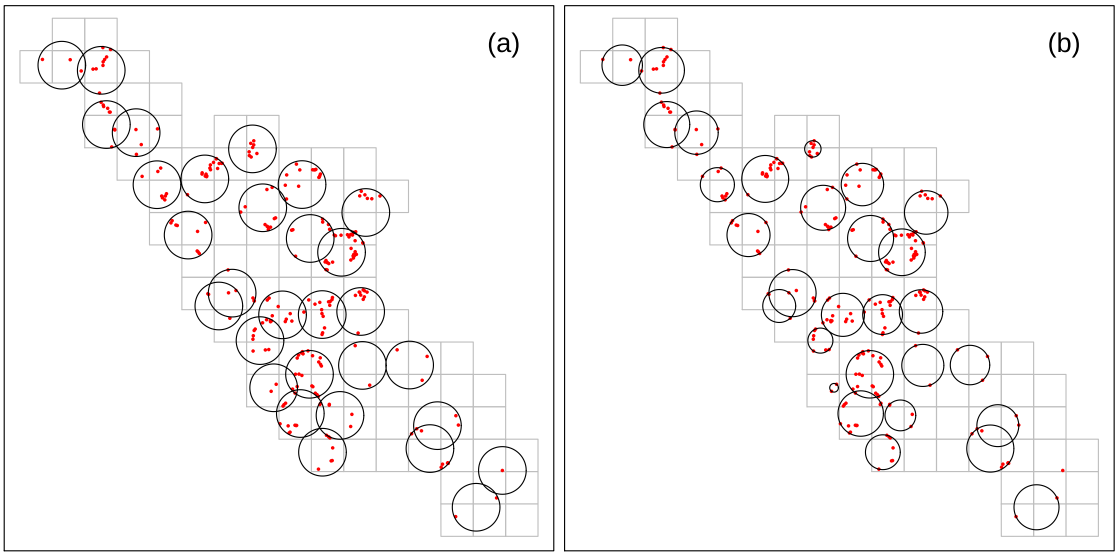

Figure 6. Locations of tiger pugmarks along tracks with $17 \times 17\,\text{km}^2$ grid squares. Panel (a) shows a covering by 30 discs of radius 12.5 km, whilst panel (b) shows the a covering by 30 discs with the same centres but with radius no larger than needed to cover the same points as in (a). One point near the bottom right of the figure is isolated and 'covered' by a disc of radius zero.

expected total number of discs in the minimal covering. For $K = 1000$, the probability of finding the true minimum is less than 0.02, and the expected minimum number of discs is approximately one more than the 'true' minimum. Figure 7b shows the probability that one method will give a better solution than the other, or that the two methods will give the same result. Algorithm `shrink.furthest` is slightly more likely than `grow.nearest` to give a smaller solution, but if $K > 2$ the most likely outcome is a tie.

Discussion

We have presented simple heuristic algorithms for the minimal disc covering problem. The algorithms proceed sequentially, selecting a point that is not yet covered by simple random sampling and finding a disc that covers that point. The two basic procedures for finding a disc, `grow` and `shrink`, could also include further random selection. However, except in the pathological example of Figure 1a, deterministic versions of these algorithms, namely `grow.nearest` and `shrink.furthest`, were more effective. These algorithms had similar performance in a variety of simulated and real examples but `grow.nearest` was generally slower, particularly as the number of points, n, increased.

One advantage of a search approach that generates multiple candidate solutions is that it lends itself to incorporation of secondary optimisation criteria. For example, we might seek minimal coverings that minimize or maximize the number of points that are covered by more than one disc. Related to this, one may wish to compare different solutions that

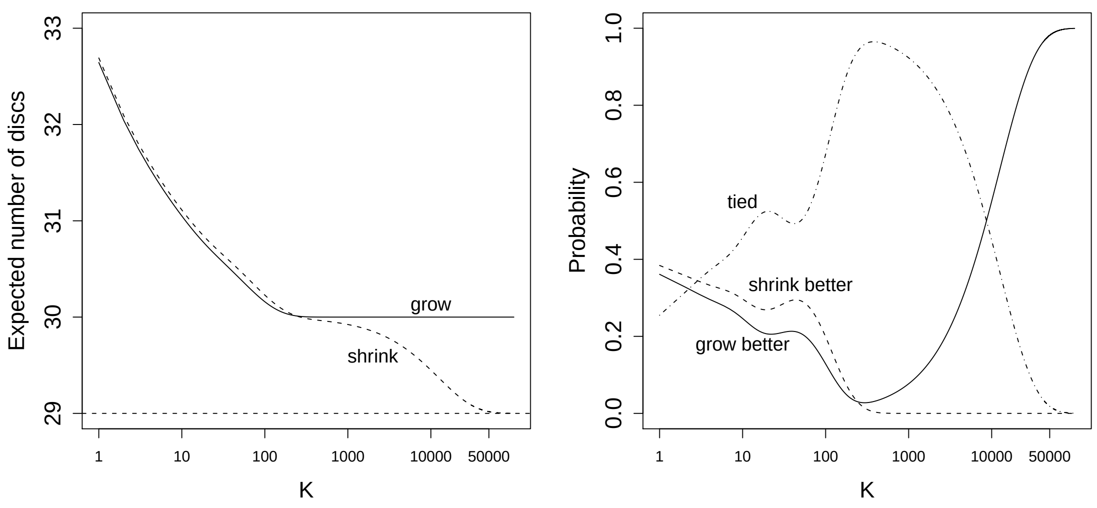

Figure 7. Comparative performance of `shrink.furthest` and `grow.nearest` for the tiger footprint data with $r = 11.5$, calculated using the empirical distributions in Table 5.

are generated by the algorithm. In terms of the subsets of points that are covered, this is a problem of comparing potentially overlapping clusterings. Alternatively, one might wish to make more geometrical comparisons. These issues will be explored elsewhere.

Each iteration of the algorithm generates a disc covering and the overall minimum from K runs is chosen. Computational time is approximately proportional to K and for simulated random patterns of up to 800 points, the rate of increase with the number of points, n was less than $n^{1.75}$ for the disc radii that we considered.

It is difficult to give general advice about how to choose K. One approach is to link the problem to the species estimation problem, where here the 'species' are the different possible numbers of discs that can be generated by the algorithm. Finch, Mendell and Thode (1989) suggested this approach for evaluating the use of multiple random starting points for continuous optimization problems, using the the Good-Turing statistic (Good, 1953) to estimate the probability that a further run from a new random starting point would discover a solution that had not been encountered previously. A difficulty in applying this approach to the current problem is that the minimum covering is often found much less frequently than a covering with one more disc, as in Table 5. The Good-Turing estimate is then frequently zero even when the true probability is not zero. It may be useful to investigate interval estimates of the probability (Almodevar *et al.*, 2000) or more generally to consider adaptive stopping rules.

We applied the algorithms to tiger pugmarks, to estimate the minimum number of adult animals that could have left the pugmarks, assuming a circular home range of given radius, r. The results of the analysis can be communicated easily to non-technical audiences, for example policy makers and conservation managers. Graphical displays such as Figure 6 are potentially useful, but need to be interpreted in conjunction with details of the sampling effort; for example, ‘gaps’ in the figure may represent limited sampling effort rather than absence of tigers. The central assumption, that the home range is circular is of course simplistic, but allows progress to be made. The effect of varying r is easily investigated. The data set did not distinguish male and female pugmarks, but if this were possible, the data could be analysed separately for males and females, allowing different home range sizes.

Whilst clearly less useful than an estimate of true population size, an estimate of minimum number of tigers present may still have value to conservation biologists as an *index* of population size, i.e. a measure that is correlated with the true population size. Such indices can be useful in conservation biology to monitor changes in population size over time and to assess whether interventions have been effective, provided that indices are based on surveys that involved similar effort and spatial coverage. Jhala *et al.* (2011) discuss indices of tiger abundance.

Acknowledgments

I am grateful to Guru Guillera-Arroita and Matt Linkie for providing access to the data and detailed comments on the manuscript. The idea of applying minimal disc covering to the tiger pugmark data first arose during discussions with Guru Guillera-Arroita and Byron Morgan.

References

Almudevar, A., Bhattacharya, R.N. and Sastri, C.C.A. (2000) Estimating the probability mass of unobserved support in random sampling. *Journal of Statistical Planning and Inference*, **91**, 91–105.

Agarwal, P.K. and Sharir, M. (1998) Efficient algorithms for geometric optimization. *ACM Computing Surveys*, **30**, 412–458.

Burt, W.J. (1943) Territoriality and home range concepts as applied to mammals. *Journal of Mammalogy*, **24**, 346–352.

Drezner, Z. and Shelah, S. (1987) On the complexity of the Elzinga-Hearn algorithm for the 1-center problem *Mathematics of Operations Research*, **12**, 255–261.

Elzinga, J. and Hearn, D.W. (1972) Geometrical solutions for some minimax location problems. *Transportation Science*, **6**, 379–394.

Finch, S.J., Mendell, N.R. and Thode, C. Jr. (1989) Probabilistic measures of adequacy of a numerical search for a global maximum. *Journal of the American Statistical Association*, **84**, 1020–1023.

Fowler, R.J., Paterson, M.S. and Tanimoto, S.L. (1981). Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, **12**, 133–137.

Franceschetti, M., Cook, M. and Bruck, J. (2001) A geometric theorem for approximate disk covering algorithms. *http://paradise.caltech.edu/papers/etr035.pdf.*

Fu, B., Chen, Z. and Abdelguerfi, M. (2007) An almost linear time 2.8334-approximation algorithm for the disc covering problem. In *Algorithmic Aspects in Information and Management: Lecture Notes in Computer Science, Vol. 4508*, pp. 317–326.

Good, I.J. (1953) The population frequencies of species and the estimation of population parameters. *Biometrika*, **40**, 237–264.

Guillera-Arroita, G., Morgan, B.J.T., Ridout, M.S. and Linkie, M. (2011) Species occupancy modeling for detection data collected along a transect. *Journal of Agricultural, Biological and Environmental Statistics*, textbf16, 301-317.

Guillera-Arroita, G., Morgan, B.J.T., Ridout, M.S. and Linkie, M. (2012) Models for species-detection data collected along transects in the presence of abundance-induced heterogeneity and clustering in the detection process. *Methods in Ecology and Evolution*, textbf3, 358-376.

Hearn, D.W., Vijay, J. and Nickel, S. (1995) Codes of geometrical algorithms for the (weighted) minimum circle problem. *European Journal of Operational Research*, **80**, 236–237.

Hochbaum, D. S. and Maass, W. (1985) Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, **1**, 130–136.

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley: Chichester, UK.

Jhala, Y., Qureshi, Q. and Gopal, R. (2011) Can the abundance of tigers be assessed from their signs? *Journal of Applied Ecology*, **48**, 14–24.

Linkie, M., Deborah J. Martyr, D.J., Holden, J., Yanuar, A., Hartana, A.T. Sugardjito, J. and Leader-Williams, N. (2003) Habitat destruction and poaching threaten the Sumatran tiger in Kerinci Seblat National Park, Sumatra. *Oryx*, **37**, 41–48.

Linkie, M., Chapron, G., Martyr, D.J., Holden, J. and Leader-Williams, N. (2006) Assessing the viability of tiger subpopulations in a fragmented landscape. *Journal of Applied Ecology*, **43**, 576–586.

Linkie, M., Guillera-Arroita, G., Smith, J. and Rayan, M. (2010) Monitoring tigers with confidence. *Integrative Zoology*, **5**, 342–250.

Meggido, N. (1983) Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, **12**, 759–776.

Welzl, E. (1991) Smallest enclosing disks (balls and ellipsoids). In "New Results and New Trends in Computer Science" (H. Maurer, ed.), *Lecture Notes in Computer Science*, **555**, 359–370.

Wibisono, H.T. and Pusparini, W. (2010) Sumatran tiger (*Panthera tigris sumatrae*): A review of conservation status. *Integrative Zoology*, **5**, 313–323.

Xiao, B., Cao, J., Zhuge, Q., He, Y. and Sha, E. H.-M. (2004) Approximation algorithms design for disk partial covering problem. International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04), p. 104.

Table 1: Performance of five algorithms, with $K = 5$, for simulated point patterns. The table shows the minimum number of discs found by each algorithm, averaged over 400 simulated patterns. The pooled standard error of the means is also shown.

Radius (r)	Pattern	Algorithm `shrink.random`*	`shrink.random`	`shrink.furthest`	`grow.random`	`grow.nearest`	Pooled SE
0.05	inhibition	58.94	58.79	58.96	58.75	59.01	0.090
0.05	random	46.33	46.02	45.93	45.98	45.94	0.115
0.05	clustered	36.60	35.78	34.86	35.50	34.91	0.412
0.1	inhibition	30.71	29.08	28.36	28.96	28.43	0.051
0.1	random	25.39	24.05	23.10	24.06	23.04	0.062
0.1	clustered	15.26	14.68	13.71	14.38	13.71	0.184
0.2	inhibition	12.73	12.11	10.86	11.60	10.82	0.036
0.2	random	11.37	10.73	9.65	10.31	9.63	0.037
0.2	clustered	9.25	8.84	7.69	8.55	7.68	0.088

* algorithm `shrink.random` *without* realignment

Table 2: Performance of algorithms `shrink.furthest` and `grow.nearest` with $K = 250$. Each row of the table summarizes results from 10 simulated point patterns, giving the the mean minimum number of discs found and the median iteration number on which the minimum was first encountered.

Radius		Mean minimum number of discs		Median number of iterations	
(r)	Pattern	`shrink`	`grow`	`shrink`	`grow`
0.05	inhibition	57.7	57.6	6.5	8.0
0.05	random	44.6	44.6	5.5	4.0
0.05	clustered	35.4	35.3	10.0	17.5
0.1	inhibition	26.6	26.5	63.0	52.5
0.1	random	21.7	21.7	58.5	55.5
0.1	clustered	12.8	12.8	5.0	4.5
0.2	inhibition	9.7	9.6	36.0	34.0
0.2	random	8.6	8.6	22.0	31.0
0.2	clustered	7.1	7.1	2.0	2.0

Table 3: Performance of four algorithms for the tiger data, for discs of different radius. Each algorithm was run 20 times with $K = 100$. The 'true' minimum number of discs is based on a larger run with $K = 25000$. The upper part of the table shows the mean of the minimum number of discs found in each run and the lower part shows the number of runs in which the algorithm found the 'true' minimum.

Radius (r)	'True' minimum	Algorithm `shrink.random`	`grow.random`	`shrink.furthest`	`grow.nearest`
Mean minimum number of discs					
8.5	45	45.10	45.05	45.00	45.00
9.5	40	42.45	42.30	40.50	40.65
10.5	36	38.65	39.00	36.95	37.00
11.5	34	36.70	36.45	35.35	35.05
12.5	30	32.45	32.50	31.20	31.40
Number of runs that gave 'true' minimum					
8.5	45	18	19	20	20
9.5	40	0	0	10	7
10.5	36	0	0	1	0
11.5	34	0	0	0	0
12.5	30	0	0	0	1

Table 4: Performance of algorithms `shrink.furthest` and `grow.nearest` with $K = 25000$. The table shows the number of subgroups of points (m), the number of discs needed to cover largest subgroup (C_{max}), the frequency with which the optimal solution was found for the largest subgroup and the iteration number on which the optimum was first found.

Radius	Groups	'True'		Freq of opt		First found	
(r)	(m)	minimum	C_{max}	`shrink`	`grow`	`shrink`	`grow`
8.5	11	45	23	6927	7014	4	4
9.5	7	40	33	141	141	129	413
10.5	5	36	30	7	5	624	2718
11.5	4	34	29	0	2	> 25000	1872
12.5	3	30	26	9	16	1377	478

Table 5: Frequency distributions of the number of discs needed to cover the largest subset of points using algorithms `shrink.furthest` and `grow.nearest` with $K = 275000$ for $r = 11.5$.

Algorithm	Number of discs needed									
	29	30	31	32	33	34	35	36	37	38
`shrink.furthest`	5	4453	34002	85901	91707	46308	11307	1252	65	0
`grow.nearest`	3	3991	32454	85418	92802	47437	11585	1255	54	1