On k-Enclosing Objects in a Coloured Point Set

Luis Barba* Stephane Durocher†‡ Robert Fraser† Ferran Hurtado§¶ Saeed Mehrabi†
Debajyoti Mondal† Jason Morrison† Matthew Skala† Mohammad Abdul Wahid†

Abstract

We introduce the exact coloured k-enclosing object problem: given a set P of n points in $\mathbb{R}^2$, each of which has an associated colour in $\{1, \ldots, t\}$, and a vector $\mathbf{c} = (c_1, \ldots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each $1 \leq i \leq t$, find a region that contains exactly c_i points of P of colour i for each i. We examine the problems of finding exact coloured k-enclosing axis-aligned rectangles, squares, discs, and two-sided dominating regions in a t-coloured point set.

1 Introduction

Given a set P of n points in $\mathbb{R}^2$ and a positive integer k, the problem of finding a region (e.g., a disc, square, or rectangle) that encloses exactly k points of P while optimizing specific parameters (e.g., minimizing area or perimeter) has been examined extensively [3, 17, 19, 20, 25]. In many applications, the input data are classified into categories, or *colours*, leading us to consider the following natural generalization. Given a set P of n points in $\mathbb{R}^2$, each of which has an associated colour in $\{1, \ldots, t\}$, and a vector $\mathbf{c} = (c_1, c_2, \ldots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each $1 \leq i \leq t$, find a region enclosing at least c_i points in P of colour i for each i. Such problems commonly appear in pattern recognition [25] (e.g., when features are represented as a point set, and the objective is to identify a precise cluster with the prescribed number of features), as database queries (e.g., find a holiday destination with five tourist attractions, two hotels, and six restaurants), and in facility location [1] (e.g., selecting a location for a bus stop in a densely populated area). Unlike the smallest k-enclosing rectangle or disc problems, the solution to the exact coloured k-enclosing object problem may not always exist. Therefore, in the exact coloured k-enclosing object problem, the primary objective is to find *any* coloured k-enclosing object that contains exactly the required number of points of each colour (if such a region exists), rather than finding the smallest such object. The problem is defined formally as follows.

Exact Coloured k-Enclosing Object Problem

INPUT: A set P of n points in $\mathbb{R}^2$, each of which is assigned a colour in $\{1, \ldots, t\}$, and a t-tuple $\mathbf{c} = (c_1, \ldots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each i.

QUESTION: Find a region (such as an axis-aligned rectangle, square, or disc) in $\mathbb{R}^2$ that encloses exactly c_i points of P of colour i for each i.

Although smallest k-enclosing object problems are well explored, very little is known about the exact coloured k-enclosing object problem. In this paper we introduce the exact coloured k-enclosing object problem for axis-aligned rectangles, squares, discs, and two-sided dominance regions in polychromatic point sets. Section 2 begins with an examination of related work. In Sections 3–5, we show that exact coloured k-enclosing axis-aligned rectangles, discs, and two-sided dominance regions can be found in $O(n^2k)$, $O(KVD(n,k))$, and $O(n \log n)$ time, respectively, where $KVD(n,k)$ denotes the time required to construct the kth order Voronoi diagram. In Section 6, we discuss generalizations to higher dimensions.

Throughout the paper, n denotes the number of points in P, t denotes the number of distinct colours of points in P, and k denotes the number of points to be contained in the bounding object, i.e., $k = \sum_{i=1}^{t} c_i$. Also, we assume that points are in general position.

2 Related Work

The exact coloured k-enclosing object problem generalizes several known problems, and was motivated by a desire to generalize the jumbled pattern matching problem to higher dimensions. Jumbled pattern matching [9, 10] asks whether a given sequence contains any permutation of some given query string. Given an arbitrary binary sequence of length n (i.e., $t = 2$), Burcsi et al. [9] show how to construct an $O(n)$-space data structure in $O(n^2)$ preprocessing time that supports queries in $O(k)$ time

*Carleton University, Canada, and Université Libre de Bruxelles, Belgium. luis_barbaflores@carleton.ca

†University of Manitoba, Canada. {durocher,fraser,mehrabi,jyoti}@cs.umanitoba.ca, Jason.Morrison@umanitoba.ca, {mskala,wahid}@cs.umanitoba.ca

‡Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

§Universitat Politècnica de Catalunya (UPC), Spain. Ferran.Hurtado@upc.edu

¶Work of the author is supported in part by projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, and ESF-EuroGIGA-CRP ComPoSe, MICINN EUI-EURC-2011-4306.

for any arbitrary query string of length k. Given an arbitrary sequence of length n (i.e., any $t \geq 2$), Burcsi et al. gave an $O(n)$-space data structure with $O(n)$ preprocessing time to report all occurrences of a query string in $O(n\sqrt{t/(k \log t)})$ expected time. The one-dimensional exact coloured k-enclosing problem reduces to jumbled pattern matching, since a t-coloured set of n points in $\mathbb{R}$ can be mapped to an array A of length n. One can slide an interval of width k over A to find an exact coloured k-enclosing interval in $O(n)$ time (or $O(n \log n)$ time if the point set is unsorted).

The problem of finding a smallest axis-aligned square, rectangle or disc that encloses k of n uncoloured points in $\mathbb{R}^2$ has been studied for over two decades. In 1991, Aggarwal et al. [3] showed that a smallest k-enclosing rectangle or square can be computed in $O(nk^2 \log n)$ time and $O(nk)$ space. Both the time and space complexities were subsequently improved several times [11, 17, 20, 29], and the current best known algorithms in $\mathbb{R}^2$ require $O(nk^2 + n \log n)$ time and $O(n)$ space for rectangles [20] and $O(n \log n + n \log^2 k)$ time and $O(n)$ space for squares [17]. The smallest k-enclosing disc problem also has a long history. The current best known algorithms require $O(n \log n + nk \log k)$ time and $O(nk + k \log^2 k)$ space [20], and $O(nk \log^2 n)$ time and $O(nk)$ space [19]. The lower bound on time is believed to be $\Omega(nk)$ [26].

A generalization of the smallest k-enclosing object problem with respect to a t-coloured point set is to find the smallest colour-spanning object, i.e., the smallest rectangle or disc that contains at least one point of each colour. Abellanas et al. [1] showed how to compute the smallest colour-spanning rectangle in $O(n(n-t) \log^2 t)$ time, which was later improved to $O(n(n-t) \log t)$ [16]. The best known algorithm for computing the smallest colour-spanning disc takes $O(nt \log n)$ time [22].

In $\mathbb{R}^2$, the exact coloured k-enclosing rectangle problem reduces to a subarray sum problem considered by Takaoka [30] that, given an $m \times m$ array and a value v, asks to find a subarray such that the sum of its values is v. Takoaka showed that such a subarray can be computed in $O(m^3 \log m)$ time. The exact coloured k-enclosing rectangle problem can be solved in $O(n^3 \log n)$ time by reduction to this reverse range query problem. In Section 3, we show how to achieve $O(n^2 k)$ time, improving the running time by a linear factor for small values of k.

Problems that involve finding discs that enclose a prescribed set of points with few outliers can be viewed as variants of the exact coloured k-enclosing object problem. For example, consider a point set with r red and b blue points, and the problem of finding a disc that encloses all of the red points and at most c_b blue points. Cheung and Daescu [13] showed that the existence of such a disc can be decided in $O(n + n^{1/4} c_b^{11/4} \log^{O(1)} n)$ time, and later gave an improved $O(rb \log b + r \log r)$-time algorithm to find the smallest disc that minimizes the number of blue points [7]. Backer and Keil [5] showed that given a red-blue point set, an axis-aligned rectangle with the maximum number of red points but no blue points can be computed in $O(n \log^3 n)$ time. Dobkin et al. [18] considered the problem of computing a rectangle that maximizes the difference between the numbers of enclosed red and blue points, and gave an $O(n^2 \log n)$-time algorithm to find such a rectangle.

Finding an exact coloured k-enclosing object is an inverse formulation of the range query problem in a polychromatic point set. One may consider such inverses for a variety of different shapes of query ranges. For example, in Section 5 we examine an inverse problem of the dominance range query that, given a t-coloured point set P in $\mathbb{R}^2$, asks whether there is a point in P that dominates exactly c_i points of P of colour i for each i. JáJá et al. [23] gave an $O(n \log n / \log \log n)$-space data structure for dominance counting queries, i.e., counting the number of points dominated by the query point, in $O(\log n / \log \log n)$ time. One could use such data structures for every colour class to determine whether there is a point in P that dominates exactly c_i points of P of colour i for each i, but it is not obvious how to combine the dominance counts efficiently since the dominating points returned will not coincide in general. In Section 5, we give an algorithm to solve this problem in $O(n \log n)$ time.

3 Axis-Parallel Rectangles

In this section, we study the exact coloured k-enclosing rectangle problem. Given a set P of n coloured points in $\mathbb{R}^2$, where the colour of a point is an integer in $\{1, \ldots, t\}$, and given a query as a t-tuple $\mathbf{c} = (c_1, \ldots, c_t)$, where $\forall i, c_i \in \mathbb{Z}^+$, the problem is to determine whether there exists an axis-aligned rectangle which covers exactly c_i points of colour i for all i. The algorithm works by considering every possible choice for the top and bottom of the rectangle. That is, the algorithm checks whether a solution exists within any of the horizontal strips of the plane determined by a pair of horizontal lines passing through points of P.

We proceed by fixing the bottom of the strip and then increasing its height monotonically, i.e., new points are added to the strip, one by one, in order of increasing y-coordinates. We store all the points contained in the strip in a linked list $\mathcal{L}$ sorted by x-coordinates in increasing order. As the height of the strip increases, new points are inserted to $\mathcal{L}$. Upon insertion of a point, we check each *window* of width k (i.e., a horizontal interval on the strip containing exactly k points) containing the new point (recall that $k = \sum_{i=1}^{t} c_i$). Since this involves sliding a fixed window from left to right, this check may

be performed in $O(k)$ time. For example, keep an array A of width t, where $A[i]$ is the number of points of colour i in the window, and a counter a that tracks the current number of colours satisfied in the query. When the window moves one step, a single point is added to the window and one is removed. For each of these, update $A[i]$ accordingly and compare $A[i]$ with c_i to see whether a should be updated. If $a = t$, then a solution exists containing the points in the window.

To insert a point in $\mathcal{L}$, we use a preprocessing step that allows us to perform this insertion in $O(1)$ time. Prior to starting the algorithm, sort the points of P and store them in order of increasing x-coordinates in a linked list L_x.

With the bottom of the strip fixed on a line through some point p_j of P, we assume that list L_x is updated to store only the points of P that lie above p_j. The preprocessing is described as follows. We copy list L_x into a list L in $O(n)$ time. Then, remove points from list L, one by one, in order of decreasing y-coordinates. Before removing a point p, we store two pointers $p.left$ and $p.right$ to the predecessor and successor of p in L.

After preprocessing, to insert a point p_i into $\mathcal{L}$, we splice it between $p_i.left$ and $p_i.right$. Because points were removed from top to bottom, both $p_i.left$ and $p_i.right$ have y-coordinates less than p_i. Thus, as the insertions into $\mathcal{L}$ occur from bottom to top, both $p_i.left$ and $p_i.right$ belong to $\mathcal{L}$ when p_i is inserted. Since L_x stored the points in sorted order by x-coordinates, p_i lies to the right of $p_i.left$ and to the left of $p_i.right$. Moreover, no point with y-coordinate less than p_i and larger than p_j lies between $p_i.left$ and $p_i.right$. The corresponding pseudocode is given in Algorithm 1.

Theorem 1 *The exact coloured k-enclosing rectangle problem can be solved in $O(n^2k)$ time.*

Proof. The outer for-loop iterates n times, and $O(n)$-time preprocessing is performed on each iteration. An *insert* operation is performed in the inner loop in Step 11. As each insert requires only to splice a new point in the list, it is performed in $O(1)$ time. By Step 31, list L_x always contains the points at or above p_j sorted by x-coordinates and, consequently, list $\mathcal{L}$ is also kept sorted in order of increasing x-coordinates.

Each iteration of the inner loop to determine whether a solution exists around the inserted point takes $O(k)$ time, resulting in a total running time of $O(n^2k)$.

For correctness, consider a solution whose lowest (resp., highest, leftmost, rightmost) point is p_{bot} (resp., p_{top}, p_ℓ, p_r). We consider all pairs of top and bottom points, so one iteration of the inner loop exists where $i = \text{top}$ and $j = \text{bot}$. Since p_{top} is the point inserted into the linked list at this step, the algorithm checks all rectangles whose highest and lowest points are p_{top} and p_{bot} and also contain exactly k points, one of which is p_{top}. Every solution, if any exists, is such a rectangle. □

Algorithm 1 RECT($P, \mathbf{c}$)

1: Sort $P \cup \{(0, -\infty), (0, \infty)\}$ by x-coordinates and store its points in order of increasing x-coordinates in a linked list L_x.
2: Sort P in order of increasing y-coordinates. Let p_i be the i-th point in this ordering.
3: $k \leftarrow \sum_{i=1}^{t} c_i$
4: **for** $j := 0$ **to** $n - 1$ **do**
5: Copy list L_x into a list L.
6: **for** $i := n - 1$ **to** $j + 1$ **do**
7: $p_i.left \leftarrow pred_L(p_i)$, $p_i.right \leftarrow succ_L(p_i)$.
8: Remove p_i from list L.
9: $\mathcal{L} \leftarrow [(0, -\infty), p_j, (0, \infty)]$
10: **for** $i := j + 1$ **to** $n - 1$ **do**
11: Splice p_i into $\mathcal{L}$ between $p_i.left$ and $p_i.right$.
12: $A[1 \dots t] \leftarrow 0$, $a \leftarrow 0$
13: $prev \leftarrow p_i$, $next \leftarrow p_i$
14: % Put the nodes for the k predecessors and successors of p_i into an array C.
15: **for** $l := 0$ **to** k **do**
16: $C[k - l] \leftarrow prev$, $prev \leftarrow pred_{\mathcal{L}}(prev)$
17: $C[k + l] \leftarrow next$, $next \leftarrow succ_{\mathcal{L}}(next)$
18: % Slide a window of width k along the array.
19: **for** $l := 0$ **to** $2k - 1$ **do**
20: $cur \leftarrow col(C[l])$
21: Increment $A[cur]$
22: If $A[cur] = c_{cur}$ then increment a.
23: If $A[cur] = c_{cur} + 1$ then decrement a.
24: **if** $l \geq k$ **then**
25: $cur \leftarrow col(C[l - k])$
26: Decrement $A[cur]$
27: If $A[cur] = c_{cur}$ then increment a.
28: If $A[cur] = c_{cur} - 1$ then decrement a.
29: **if** $a = t$ **then**
30: **return** a rectangle bounded by p_j, p_i, $C[l]$, and $C[l - k + 1]$
31: Remove p_j from list L_x.
32: % There is no rectangle satisfying the query in P.
33: **return** $\emptyset$

See Section 6.1 for a discussion of modifications to Algorithm 1 to reduce running time.

4 Discs and Axis-Parallel Squares

The kth order Voronoi diagram of a set P of n points in the plane is a partition of the plane into maximal convex cells such that any two points in a common cell have the same k nearest neighbours in P. The number of kth order Voronoi cells is $\Theta(k(n - k))$ [24]. Thus, if C is a kth order Voronoi cell whose set of nearest neighbours is $P_C = \{p_1, \dots, p_k\} \subseteq P$, for any point $p \in C$, there exists a disc D_p centered at p such that $D_p \cap P = P_C$. If the points of P are coloured, it suffices

to verify whether there exists a kth order Voronoi cell C such that the frequencies of the colours of the points in P_C correspond to the input colour t-tuple $\mathbf{c}$.

Traversing the cells of the kth order Voronoi diagram by a breadth-first or depth-first search on the dual graph requires $O(k(n-k))$ steps. The sets of k nearest neighbours in any two adjacent cells C_a and C_b differ in exactly two points. Specifically, there exist points $\{p_a, p_b\} \subseteq P$ such that the edge e common to C_a and C_b is on the bisector of p_a and p_b and for any point in C_a close to e, p_a and p_b are respectively its kth and $(k+1)$st closest points in P, whereas the relationship is reversed for C_b. When transitioning from C_a to C_b, the set of k nearest neighbours is updated in $O(1)$ time by $P_{C_b} = (P_{C_a} \cup \{p_b\}) \setminus \{p_a\}$. We find the k nearest neighbours for the first cell in the traversal in $O(n)$ time using selection and partitioning, compute the frequencies of the corresponding colours, and initialize a count s of the number of frequencies that match the input t-tuple $\mathbf{c}$. Upon moving from the cell C_a to its neighbouring cell C_b during the traversal, it suffices to increment the frequency count for the colour of p_b, check whether this new value matches the corresponding value in $\mathbf{c}$, decrement the frequency count for the colour of p_a, check whether this new value matches the corresponding value in $\mathbf{c}$, and update s accordingly. If $s = t$, then a disc centered at any point in C_b with radius $r = \max_{q \in P_{C_b}} \text{dist}(p, q)$ is a solution to the exact coloured k-enclosing disc problem. Since the kth order Voronoi diagram has size $\Theta(k(n-k))$ in general, constructing it requires $\Omega(k(n-k))$ time in the worst case. This gives the following theorem.

Theorem 2 *The exact coloured k-enclosing disc problem can be solved in $O(KVD(n,k))$ time, where $KVD(n,k)$ denotes the time required to construct the kth order Voronoi diagram.*

Efficient deterministic algorithms for constructing the kth order Voronoi diagram include those of Chazelle and Edelsbrunner in $O(n^2 + k(n-k)\log^2 n)$ time using $O(n^2)$ space and $O(n^2 \log n + k(n-k)\log^2 n)$ time using $O(k(n-k))$ space [12], Lee in $O(nk^2 \log n)$ time using $O(n^2(n-k))$ space [24], and Aurenhammer in $O(nk^2 \log n)$ time using $O(k(n-k))$ space [4]. Efficient randomized algorithms include those of Clarkson in $O(n^{1+\epsilon}k)$ expected time for any fixed $\epsilon > 0$ [14], Ramos in $O(n \log n + nk2^{c \log^* n})$ expected time, where c is constant [28], and Agarwal et al. in $O(k(n-k)\log n + n\log^3 n)$ expected time [2].

Under ℓ_∞ distance, a disc of radius r centered at a point p is realized as an axis-parallel square of side length $2r$ centered at point p. Consequently, just as we did for discs under ℓ_2 distance, the kth order Voronoi diagram under ℓ_∞ distance can be used to find a square that contains exactly c_i points of P of colour i for each i, if any such square exists. Equivalently, ℓ_1 distance can be used with a $\pi/4$ rotation of the axes. Several of the algorithms for constructing the kth order Voronoi diagram under ℓ_2 distance can be applied under ℓ_1 or ℓ_∞ distance. For example, Lee [24] states that his $O(nk^2 \log n)$-time algorithm applies to the ℓ_∞ and ℓ_p distance metrics for any $p \in [1, \infty)$. This gives the following corollary.

Corollary 3 *The exact coloured k-enclosing axis-parallel square problem can be solved in $O(\overline{KVD}(n,k))$ time, where $\overline{KVD}(n,k)$ denotes the time required to construct the kth order Voronoi diagram under the ℓ_∞ distance metric.*

5 Two-Sided Dominating Regions

Let P be a set of n points in the plane, each with a colour from $\{1, \ldots, t\}$. For each i, let P_i denote the subset of P of colour i and let $n_i = |P_i|$. The point $p = (p_x, p_y)$ *dominates* the point $q = (q_x, q_y)$ if $p_x > q_x$ and $p_y > q_y$. We show how to determine in $O(n \log n)$ time if there exists some point r in the plane that dominates $\mathbf{c} = (c_1, \ldots, c_t)$ points of P, i.e., r dominates c_i points of P_i for each i, and to return such a point r if one exists.

For each i, the region of points that dominate at least c_i points of P_i is bounded by a monotonic non-increasing orthogonal polygonal chain. The region of points that dominate exactly c_i points of P_i is bounded by two such chains. This boundary is defined by Bose and Morrison [8] as the c_i- and c_{i+1}-levels in P_i, consisting of a staircase that can be partitioned into an x-monotone set of $O(n_i)$ rectangles, as shown in Figure 1. Any solution point r must be contained in one of these rectangles. For each colour i, the corresponding rectangles are bounded by $O(n_i)$ segments [8] and can be constructed in $O(n_i)$ time (or $O(n_i \log n_i)$ time if the points must be sorted).

Once the t sets of candidate rectangles are constructed and stored in t lists, each in order of increasing x-coordinates, the rectangles for any pair of colours (i, j) can be intersected in $O(n_i + n_j)$ time. The time bound follows from the constant complexity of each set of rectangles for any given y-coordinate using Bentley and Ottman's line sweep [6]. Since any pair of rectangles intersect in either zero or one rectangle, recursive pairwise intersection of these sets requires only $O(n)$ space with $O(n)$ time per round and $O(\log t)$ rounds. Thus our algorithm requires $O(n \log t)$ time, $O(n \log n)$ preprocessing time for sorting, and $O(n)$ space.

Theorem 4 *The exact coloured k-enclosing two-sided dominating region problem can be solved in $O(n \log n)$ time.*

6 Discussion

In this section, we address generalizations and refinements of the exact coloured k-enclosing object problem.

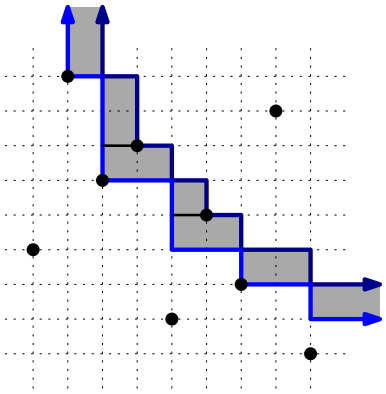

Figure 1: Points in the shaded region dominate exactly two black points. The partition into $O(n_i)$ rectangles is illustrated, at most one of which intersects any horizontal line. Analogous candidate regions are constructed for each colour set P_i. Their common intersection is non-empty if and only if there exists a solution to the k-enclosing two-sided dominating region problem.

6.1 Improved Time for Axis-Parallel Rectangles

Considering that the fastest known algorithm for the uncoloured version of the problem takes $O(nk^2 + n \log n)$ time in the worst case [20], the running time of Theorem 1 compares relatively favourably. Nevertheless, in this section we present two refinements which slightly improve our upper bound.

Since a solution must contain at least k points, Algorithm 1 can be modified so that the two outer loops skip k points and iterate $O(n-k)$ times each. This results in a running time of $O(n \log n + (n-k)^2 k)$. In the remainder of this section, we discuss reducing the factor k of the running time. Complete details are omitted due to space constraints.

Let m denote the most frequent colour in the query, i.e., $c_m = \max_{i \in \{1 \ldots t\}} c_i$. A point that does not have colour m is called $\overline{m}$-coloured. Let $k' = k - c_m$ denote the number of $\overline{m}$-coloured points in the query.

To improve the running time, the approach described in Algorithm 1 is used to search for a solution satisfying the query on the $\overline{m}$-coloured points. Upon finding a match, it is determined whether a rectangle (still an interval of the strip) containing exactly these $\overline{m}$-coloured points may also contain c_m m-coloured points.

To address this, the linked list $\mathcal{L}$ of Section 3 is built containing only $\overline{m}$-coloured points. Given a point p_i in the strip, let $succ^j_{\mathcal{L}}(p_i)$ denote the jth successor in $\mathcal{L}$. Let $succ^0_{\mathcal{L}}(p_i) = p_i$, and let $n_m(succ^j_{\mathcal{L}}(p_i))$ denote the number of m-coloured points with x-coordinate between $succ^{j-1}_{\mathcal{L}}(p_i)$ and $succ^j_{\mathcal{L}}(p_i)$ in this strip. A rightmost dummy point p_∞ at $x = \infty$ is used so that $n_m(p_\infty)$ counts the m-coloured points to the right of the last $\overline{m}$-coloured point. Therefore, a solution exists with a leftmost m-coloured point p_i if the query (except c_m) is satisfied by p_i and its $k' - 1$ successors in $\mathcal{L}$, and $\sum_{j=i+1}^{i+k'} n_m(succ^j_{\mathcal{L}}(p_i)) \leq c_m \leq \sum_{j=i}^{i+k'+1} n_m(succ^j_{\mathcal{L}}(p_i))$.

The challenge is to update the values of $n_m(i)$ and $n_m(i+1)$ following the insertion of an $\overline{m}$-coloured point. The preprocessing step may be augmented to track the number of m-coloured points on each side of an inserted $\overline{m}$-coloured point. For example, the disjoint set union data structure of Gabow and Tarjan [21] would allow (with some extra bookkeeping) to track the numbers of m-coloured points between consecutive $\overline{m}$-coloured points using *find* when encountering an m-coloured point, and storing the sizes of the sets prior to using *union* upon the removal of an $\overline{m}$-coloured point. Complete details are omitted due to space constraints. $O(n)$ union and find operations are performed on the data structure, which requires $O(n)$ time total. Therefore, the algorithm runs in $O(n \log n + n^2(k - \max_i c_i))$ time, which may again be improved by substituting $(n-k)^2$ for n^2 as discussed at the beginning of this section. Note that for a binary alphabet, this yields an overall running time of $O(n \log n + (n-k)^2 \min_i c_i)$.

6.2 Smallest Exact Coloured k-Enclosing Object

The algorithms described are straightforward to modify to return the *smallest* exact coloured k-enclosing object with at most an $O(k)$ increase in running time. For example, in the case of discs it suffices to compute the minimum enclosing disc of the k points associated with each candidate kth order Voronoi cell, which can be achieved in $O(k)$ time per cell using the algorithm of Megiddo [27]. In the case of axis-parallel rectangles, Algorithm 1 can be easily modified to compute the area of every window and return the smallest rectangle without any asymptotic increase in running time.

6.3 Higher Dimensions

Several of the algorithms described have natural generalizations to higher dimensions. For example, a kth order Voronoi diagram in $\mathbb{R}^d$ has $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ cells [15], and so generalizing Theorem 2 to $\mathbb{R}^d$ gives a running time of $O(d \cdot KVD_d(n,k))$, where $KVD_d(n,k)$ denotes the time required to construct the d-dimensional kth order Voronoi diagram. Similarly, Theorem 1 generalizes to give a running time of $O(n^{2(d-1)} kd)$.

6.4 Directions for Future Research

Several questions remain open. Can the time complexity be reduced in $\mathbb{R}^2$? For discs and axis-parallel squares, can the problem be solved faster than the time required to construct a kth order Voronoi diagram? Can the time complexity be improved if the input set of points is bichromatic (i.e., when $t = 2$)?

Acknowledgements

The authors thank the participants of the 2013 Bellairs Workshop on Geometry and Graphs for stimulating discussion of ideas related to this paper. The authors also thank Sharma Thankachan for discussion of query data structures for jumbled pattern matching in strings.

References

[1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, B. Palop, and V. Sacristan. Smallest color-spanning objects. In *Proc. ESA*, volume 2161 of *LNCS*, pages 278–289, 2001.

[2] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comp.*, 27(3):654–667, 1998.

[3] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.

[4] F. Aurenhammer. A new duality result concerning Voronoi diagrams. *Disc. & Comp. Geom.*, 5:243–254, 1990.

[5] J. Backer and J. M. Keil. The bichromatic square and rectangle problems. *Technical Report 2009-01, University of Saskatchewan*, 2009.

[6] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-29:643–647, 1979.

[7] S. Bitner, Y. K. Cheung, and O. Daescu. Minimum separating circle for bichromatic points in the plane. In *Proc. ISVD*, pages 50–55, 2010.

[8] P. Bose and J. Morrison. Translating a star over a point set. In *Proc. CCCG*, pages 179–182, 2005.

[9] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

[10] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On approximate jumbled pattern matching in strings. *Theory Comput. Syst.*, 50(1):35–51, 2012.

[11] T. M. Chan. Geometric applications of a randomized optimization technique. *Disc. & Comp. Geom.*, 22(4):547–567, 1999.

[12] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing kth-order Voronoi diagrams. *IEEE Trans. Comp.*, 36(11):1349–1354, 1987.

[13] Y. Cheung and O. Daescu. Minimum separating circle for bichromatic points by linear programming. In *Proc. FWCG*, 2010.

[14] K. L. Clarkson. New applications of random sampling to computational geometry. *Disc. & Comp. Geom*, 2:195–222, 1987.

[15] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Disc. & Comp. Geom.*, 4:387–421, 1989.

[16] S. Das, P. P. Goswami, and S. C. Nandy. Smallest color-spanning object revisited. *Int. J. Comp. Geom. & App.*, 19(5):457–478, 2009.

[17] A. Datta, H.-P. Lenhof, C. Schwarz, and M. H. M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19(3):474–503, 1995.

[18] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy with applications to computer graphics and machine learning. *J. Comp. & Sys. Sciences*, 52(3):453–470, 1996.

[19] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k-enclosing circle and related problems. *Comp. Geom.: Theory & App.*, 4(3):119–136, 1994.

[20] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Disc. & Comp. Geom.*, 11:321–350, 1994.

[21] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comp. & Sys. Sci.*, 30(2):209 – 221, 1985.

[22] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Disc. & Comp. Geom.*, 9:267–291, 1993.

[23] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. ISAAC*, volume 3341 of *LNCS*, pages 558–568, 2004.

[24] D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comp.*, C-31:478–487, 1982.

[25] P. R. S. Mahapatra, A. Karmakar, S. Das, and P. P. Goswami. k-enclosing axis-parallel square. In *Proc. ICCSA*, volume 6784 of *LNCS*, pages 84–93, 2011.

[26] J. Matoušek. On geometric optimization with few violated constraints. *Disc. & Comp. Geom.*, 14:365–384, 1995.

[27] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comp.*, 4:759–776, 1983.

[28] E. A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. SoCG*, pages 390–399, 1999.

[29] M. Segal and K. Kedem. Enclosing k points in the smallest axis parallel rectangle. *Inf. Proc. Let.*, 65(2):95–99, 1998.

[30] T. Takaoka. The reverse problem of range query. *Elec. Notes Theor. Comp. Sc.*, 78:281–292, 2003.