Journal of Discrete Algorithms 30 (2015) 1–12

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Range search on tuples of points

Akash Agrawal, Saladi Rahul, Yuan Li, Ravi Janardan *

Dept. of Computer Science and Eng., Univ. of Minnesota-Twin Cities, 4-192 Keller Hall, 200 Union St. S.E., Minneapolis, MN 55455, USA

ARTICLE INFO

Article history:
Received 2 April 2014
Received in revised form 20 October 2014
Accepted 21 October 2014
Available online 11 November 2014

Keywords:
Algorithms
Data structures
Computational geometry
Spatial query processing
Range search
Geometric transformation

ABSTRACT

Range search is a fundamental query-retrieval problem, where the goal is to preprocess a given set of points so that the points lying inside a query object (e.g., a rectangle, or a ball, or a halfspace) can be reported efficiently. This paper considers a new version of the range search problem: Several disjoint sets of points are given along with an ordering of the sets. The goal is to preprocess the sets so that for any query point q and a distance δ, all the ordered sequences (i.e., tuples) of points can be reported (one per set and consistent with the given ordering of the sets) such that, for each tuple, the total distance traveled starting from q and visiting the points of the tuple in the specified order is no more than δ. The problem has applications in trip planning, where the objective is to visit venues of multiple types starting from a given location such that the total distance traveled satisfies a distance constraint. Efficient solutions are given for the fixed distance and variable distance versions of this problem, where δ is known beforehand or is specified as part of the query, respectively.

1. Introduction

1.1. Motivation

Range search is a fundamental query-retrieval problem. In this problem, we are given a set, S, of n points in $\mathbb{R}^d$ and for any query object q (e.g., a rectangle, or a ball, or a halfspace) we wish to report the points of S lying inside q. Since many queries may be asked, it is worthwhile investing effort to preprocess S into a suitable data structure (i.e., index) so that the query time is small. Thus, the resource measures of interest are the query time and the space used by the data structure. Due to its many applications in diverse domains such as spatial databases, robotics, VLSI design, CAD/CAM, computer graphics etc., the range search problem has been studied extensively in the computational geometry and database literature and space- and query time-efficient solutions have been devised for many instances of the problem (see, for instance, [2,3,8,20,22]).

In this paper, we consider a new type of range search problem that is motivated by applications in trip planning. Consider the following scenario: Suppose that we are given a database that contains the locations of venues of different types (e.g., restaurants and theaters) in a large city. A tourist is interested in visiting a restaurant and a theater, in that order, but has a constraint on the maximum travel distance (due to reasons such as cost of travel, time, mileage restriction etc.). She wishes to identify a subset of (restaurant, theater) tuples, among the set of all such tuples, so that, for every tuple in the subset, the distance from her current location to the restaurant plus the distance from the restaurant to the theater is no

* Corresponding author.
E-mail addresses: akash@umn.edu (A. Agrawal), sala0198@umn.edu (S. Rahul), lixx2100@umn.edu (Y. Li), janardan@umn.edu (R. Janardan).

http://dx.doi.org/10.1016/j.jda.2014.10.006
1570-8667/

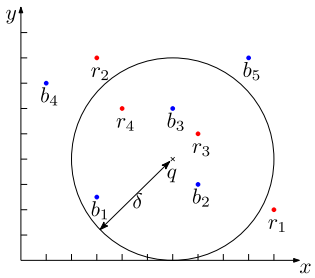


Fig. 1. Locations of two different types of venues (shown as red and blue points). Point q is the query point. The disk with radius δ represents the maximum travel distance constraint. (All figures in this paper are best viewed in color.)

larger than the specified distance. Therefore, she issues such a query from her location using, say, an application running on her smartphone and gets back the set of (restaurant, theater) tuples satisfying the distance constraint. The returned set is usually much smaller than the set of all (restaurant, theater) tuples and enables the user to make a more informed choice of the tuple to visit.

A naïve solution for this problem is to first find all the restaurants and theaters that are reachable from the query location within the specified travel distance and then check this set for the tuples to output. Unfortunately, this can be very expensive. For example, consider Fig. 1 which depicts a small data-set of restaurants (modeled as blue points b_i in the plane) and theaters (red points r_j). The user's location is denoted by the query point q. Let us assume that the maximum travel distance, denoted by δ, is 4 units. This distance constraint is represented as a disk of radius δ centered at q. In this example, it is clear that even though points b_1, b_2, b_3, r_3, and r_4 are all reachable within travel distance δ, only the (blue, red) tuples (b_2, r_3), (b_3, r_3), and (b_3, r_4) with total travel distance 3.4, 3.4, and 4, respectively, are part of the answer set. The other (blue, red) tuples formed by the points reachable from q within travel distance δ have total travel distance more than δ. (The other possible (blue, red) tuples are (b_2, r_4), (b_1, r_4), and (b_1, r_3) with total travel distance 5.7, 7 and 8.1, respectively.) Now consider a scenario where all the blue points lie on the perimeter of a disk centered at q and of radius δ and all the red points lie inside the disk. It is clear that all the points are reachable from q and yet the total travel distance for every (blue, red) tuple is larger than δ, so that the answer set is empty! This is the worst case scenario for the naive solution.

Our search problem generalizes naturally to more than two types of venues (e.g., restaurants, theaters, gas stations, grocery stores, etc.) where the objective is to start from the query point, q, and visit one venue of each type, in a pre-specified order, so that the total distance traveled is no larger than the specified maximum travel distance δ.

In fact, depending on the application there are two versions of the problem that are of interest. In the first version, δ is known beforehand (during preprocessing) while in the second version δ is known only at query time. (In both versions, q is known only at query time.) We will refer to these versions as the *fixed distance* and the *variable distance* problems, respectively.

For example, some car rental companies place a restriction on the maximum distance a user can travel. In this case the maximum travel distance, δ, is fixed for all the users and is pre-specified. This is an instance of the fixed distance problem. On the other hand, for some applications, the distance constraint can be imposed by reasons like travel time, travel cost etc. In this scenario, the maximum travel distance, δ, depends on the user and is specified only during the query. This is an instance of the variable distance problem. One might suspect that the fixed distance problem might admit a more efficient solution than the variable distance problem and, as we will see, this is indeed the case.

1.2. Problem statement and contributions

In what follows, it will be convenient to associate with each type of venue a distinct color and formulate our problem over colored point-sets in the plane.

Let S be a set of n points in $\mathbb{R}^2$, where each point is assigned a color c_i from a palette of m colors ($m \geq 2$). Let C_i be the set of points of color c_i and let $n_i = |C_i|$. Note that $C_i \cap C_j = \emptyset$ if $i \neq j$ and $\bigcup_{i=1}^{m} C_i = S$. Let $CS = (c_1, \ldots, c_m)$ be a given ordering of the colors. We wish to preprocess S into a suitable data structure so that for any query point q in $\mathbb{R}^2$ and a distance $\delta > 0$, we can report all tuples $(p_1, \ldots, p_m)$, where $p_i \in C_i$, such that $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$. (Here $d(\cdot, \cdot)$ denotes Euclidean distance.) Hereafter, we will call this the *Colored Tuple Range Query* (CTRQ) problem.

Throughout, we will discuss the CTRQ problem for $m = 2$ colors, which will henceforth be called the 2-CTRQ problem. As we will discuss in Section 4, our solutions for 2-CTRQ generalize to $m > 2$ colors. For simplicity, we define the 2-CTRQ problem as follows.

Let B be a set of blue points, $B = \{b_1, b_2, \ldots, b_{n_1}\}$, and R be a set of red points, $R = \{r_1, r_2, \ldots, r_{n_2}\}$, in $\mathbb{R}^2$, where $n_1 + n_2 = n$. We wish to preprocess the sets, B and R, so that for any query point $q : (x_q, y_q) \in \mathbb{R}^2$ and distance $\delta > 0$, we can report all tuples (b_i, r_j), such that $d(q, b_i) + d(b_i, r_j) \leq \delta$.

In this paper, we present efficient solutions to the fixed and variable distance versions of the CTRQ problem. Our results include:

- An algorithm for the fixed distance 2-CTRQ problem that exhibits a trade-off between space and query time. Specifically, our algorithm uses $O(nt)$ space and has a query time of $O((1+\frac{k}{t})\log n)$, where t is a user-specified integer parameter that controls the trade-off, $1 \leq t \leq \log n$, and k is the output size, i.e., the number of tuples reported. For instance, at the two extremes of the trade-off, i.e., $t = 1$ (resp. $t = \log n$), our algorithm uses $O(n)$ (resp. $O(n \log n)$) space and has a query time of $O((1+k)\log n)$ (resp. $O(\log n + k)$). This is discussed in Section 2.
- Several algorithms for the variable distance 2-CTRQ problem that exhibit different space and query time bounds. These include, for instance, an $O(n)$-space algorithm with a query time of $O(\sqrt{n}\log^{O(1)} n + k\log n)$ and an $O(n\log^2 n)$-space algorithm with a query time of $O((k+1)\log n)$. In fact, if we allow probabilistic preprocessing then it is possible to reduce the query time to (deterministic) $O(\log n + k)$ while still using $O(n\log^2 n)$ space. These results are discussed in Section 3. (Moreover, as in the fixed distance problem, it is possible to obtain a general trade-off in terms of the parameter t, but we omit further discussion of this as it mirrors the approach for the fixed distance problem.)
- Algorithms for the CTRQ problem for $m > 2$ colors. Specifically, for the fixed distance CTRQ problem, our algorithm uses $O(n)$ (resp. $O(n\log^2 n)$) space and the query time is $O(\log n + km\sqrt{n}\log^{O(1)} n)$ (resp. $O((1+km)\log n)$).
  For the variable distance CTRQ problem, our algorithm uses $O(n)$ (resp. $O(n\log^2 n)$) space and the query time is $O((1+km)\sqrt{n}\log^{O(1)} n)$ (resp. $O((1+km)\log n)$). Also, if we allow probabilistic preprocessing then it is possible to reduce the query time for the fixed and the variable distance CTRQ problem to (deterministic) $O(\log n + k)$ while still using $O(n\log^2 n)$ space. These results are discussed in Section 4. (Note that the results for 2 colors are not a special case of the result for $m > 2$ colors. As will be seen in Section 4, the reason is that the solutions for $m > 2$ colors involve solving the variable distance 2-CTRQ problem, instead of circular range search, in the second step.)

1.3. Approach and key ideas

Our algorithms employ a 2-step approach. In the first step, we identify an appropriate subset of blue points from which to search for (blue, red) pairs to output (with respect to the query point, q, and query distance δ). In the second step we search from each such blue point and identify those red points that are reachable from q within distance δ and output the corresponding (blue, red) pairs.

The identification of blue points in the first step needs to be done with care. Selecting too few of these will result in some valid (blue, red) pairs not being reported. On the other hand, selecting too many will lead to a high query time since many of them may not be part of any reported pair (as the example in Section 1.1 shows).

Thus, a key idea underlying our (exact) algorithms is to select efficiently *precisely* those blue points that are guaranteed to be part of at least one reported (blue, red) pair, with respect to the query point, q, and query distance δ. This, combined with the second step above, ensures that only those pairs that should be reported are actually reported. Moreover, time is not wasted in processing blue points that will not contribute to the output. We will refer to the blue points so selected as *fruitful* points.

A second key idea concerns tuning the query algorithm to the (unknown) output size. In general, a straightforward application of the 2-step method above does not necessarily result in the best possible query time, particularly if the output size is "small" since then the overall query time is dominated by other terms in the query time. To help balance these costs, we show how to identify a suitable threshold on the output size as a function of the input size, n, and how to query judiciously on the two sides of the threshold to achieve the desired balance; however, this improvement usually comes at the expense of more space. (We note that this is just a rough description of the approach and the general idea manifests itself in different ways in the fixed and the variable distance problems.)

1.4. Related work

To the best of our knowledge, there is no prior work on the CTRQ problem. However, as mentioned earlier, range search is a well-studied problem and several theoretical and practical solutions have been proposed for different instances of this problem (see, for instance, [2,3,8,20,22]). The CTRQ problem is closely linked to the circular range query problem, whose goal is to preprocess a given set of n points in $\mathbb{R}^d$ so that the points inside a query ball can be reported efficiently. Due to applications in proximity queries, the circular range query problem has been studied extensively [20]. Furthermore, via a standard lifting transformation [9], the circular range query problem in $\mathbb{R}^d$ can be transformed into a halfspace range reporting problem in $\mathbb{R}^{d+1}$. In [1], Afshani and Chan have shown that the halfspace range reporting problem in $\mathbb{R}^d$, $d \geq 4$, can be solved in $O(n^{1-1/\lfloor d/2 \rfloor}\log^{O(1)} n + k)$ query time using an $O(n)$-space data structure, where k is the output size, which is the most efficient solution known so far. Moreover, as shown in [1], the halfspace range search problem in $\mathbb{R}^3$ (hence circular range search in $\mathbb{R}^2$) can be solved in $O(n)$ space and $O(\log n + k)$ query time.

In the context of spatial databases, the CTRQ problem is closely related to the optimal sequenced route (OSR) query [23] (also called multi-type nearest neighbor query [18] or transitive nearest neighbor query [25,26]). The goal of the OSR query is to preprocess the given m sets of points into a data structure such that given a query point and an order in which to visit the sets, we can report an m-tuple of points, one per set, that minimizes the total travel distance when visited in the specified order. The main difference between the CTRQ problem and the OSR problem is that, in the latter, the goal is to output only one tuple that minimizes the total travel distance, whereas in the former all m-tuples within a specified distance from the query point are to be reported. For the OSR problem, Sharifzadeh et al. [23] have proposed three

algorithms, LORD, R-LORD, and PNE that utilize certain threshold values to filter out the non-candidate points. The proposed PNE technique can be adapted to report top-k optimal routes also. A version of the OSR problem where the order of visit to the sets is not specified is proven to be NP-hard and is known as the trip planning query (TPQ). In [15], Li et al. have developed approximation algorithms for the instance of TPQ where a source and a destination are specified. Kanza et al. [12] have shown polynomial-time heuristics for the constrained version of the TPQ problem. In [6], Chen et al. have proposed a generalization of the OSR query, called multi-rule partial sequenced route (MRPSR) query, where the order of traversal is defined by a partial order. Due to its practical applications, the OSR problem has been studied extensively and practical solutions have been proposed for many different versions of the problem such as OSR on road networks [17], k-OSR [24], interactive route search [13], traffic-aware OSR [14], and many others [4,10,11,16,19].

2. Fixed distance 2-CTRQ

Recall the problem that we wish solve: Let B be a set of blue points, $B = \{b_1, b_2, \ldots, b_{n_1}\}$, and R be a set of red points, $R = \{r_1, r_2, \ldots, r_{n_2}\}$, in $\mathbb{R}^2$, where $n_1 + n_2 = n$. We wish to preprocess the sets, B and R so that for any query point $q : (x_q, y_q) \in \mathbb{R}^2$ and distance δ, we can report all tuples (b_i, r_j), such that $d(q, b_i) + d(b_i, r_j) \leq \delta$. Note that in the fixed distance version, δ is known beforehand (during preprocessing).

We first describe an algorithm that uses $O(n)$ space and has a query time of $O((k+1)\log n)$, where k is the output size. We then show how to generalize this to a space-query time tradeoff.

As mentioned in Section 1.3, the notion of fruitful blue points is central to our approach. Recall that a blue point is a fruitful point if it is part of at least one output tuple for a given q and δ. The following lemma provides a useful characterization of such points.

Lemma 1. *For any $b_i \in B$, let $r_j \in R$ be its closest red point. Let q be a query point and δ the (fixed) query distance. Then b_i is fruitful if and only if $d(q, b_i) + d(b_i, r_j) \leq \delta$.*

Proof. ($\Leftarrow$) If $d(q, b_i) + d(b_i, r_j) \leq \delta$ then the tuple (b_i, r_j) is output. Hence, by definition, b_i is fruitful.

($\Rightarrow$) Suppose that b_i is fruitful. Thus, there is an $r_k \in R$ such that (b_i, r_k) is output. Thus, $d(q, b_i) + d(b_i, r_k) \leq \delta$. Since $d(b_i, r_j) \leq d(b_i, r_k)$, we have $d(q, b_i) + d(b_i, r_j) \leq \delta$. □

In other words, b_i is fruitful if (b_i, r_j) is part of the answer set, and not fruitful otherwise. Note that whether or not b_i is fruitful (for a fixed δ) depends on q, i.e., b_i may be fruitful for some q and may not fruitful for some other q.

Lemma 1 suggests the following preprocessing step to help determine the fruitful point at query time. For each $b_i \in B$, we compute its nearest red point $r_j \in R$ and associate with b_i the real number $d(b_i, r_j)$ as a *weight* $w(b_i)$. For a query point q, b_i is fruitful if and only if $d(q, b_i) + w(b_i) \leq \delta$, i.e., $d(q, b_i) \leq \delta - w(b_i)$, i.e., if and only if q lies inside the disk D_i of radius $\delta - w(b_i)$ centered at b_i. Thus, we build a data structure on the disks D_i that can identify the disks that are "stabbed" by q, and hence can identify the fruitful blue points. This facilitates the first step of our query algorithm.

The second step is to identify for each fruitful blue point, b_i, all the red points r_k (not merely the nearest red point) that are reachable from q within a distance of δ. Each such r_k satisfies the condition $d(q, b_i) + d(b_i, r_k) \leq \delta$, i.e., $d(b_i, r_k) \leq \delta - d(q, b_i)$, i.e., r_k is in a disk of radius $\delta - d(q, b_i)$ centered at b_i. To facilitate the identification of such red points, r_k, we build a data structure for circular range queries on the set, R, of red points.

The above preprocessing steps are shown as Algorithm 1.

As noted in Section 1.4, circular range search in $\mathbb{R}^2$ can be transformed to halfspace range search in $\mathbb{R}^3$ via the lifting map [9]. Likewise, disk stabbing can also be transformed to halfspace range search in $\mathbb{R}^3$ using the lifting map followed by geometric duality [9]. Therefore, the structure DS_f and DS_t in Algorithm 1 can be implemented as the halfspace range search structures developed in [1]. They occupy $O(n)$ space and answer queries in $O(\log n + k)$ time, where k is the output size.

Algorithm 1: Preprocessing.

Input: B, R: set of input points; δ: a real
Output: DS_f: Data structure to find fruitful points; DS_t: Data structure to form tuples.

```
1: D ← ∅ // set of disks
2: Create Voronoi diagram, V_r, for points in R.
3: Build a point location structure on V_r.
4: for all b_i ∈ B do
5:     nn_i ← nearest neighbor red point of b_i via point location in V_r.
6:     if d(b_i, nn_i) ≤ δ then // b_i cannot be fruitful if d(b_i, nn_i) > δ
7:         w(b_i) ← d(b_i, nn_i)
8:         D_i ← disk with radius δ − w(b_i) and centered at b_i
9:         D ← D ∪ {D_i}
10:    end if
11: end for
12: Create disk stabbing data structure, DS_f, on disks in D
13: Create circular range search data structure, DS_t, on points in R
```

Algorithm 2: Query.

Input: DS_f: Data structure to find fruitful points; DS_t: Data structure to form tuples; δ: a real; q: query point
Output: O: set of colored tuples satisfying the distance constraint δ with respect to q

```
O ← ∅ // Find fruitful points for q
D' ← Subset of disks D stabbed by q, as found by a disk stabbing query on DS_f with q
F ← Subset of blue points that are centers of disks in D' // F is the set of fruitful blue points
for all b_i ∈ F do
    ρ_i ← δ − d(q, b_i) // Query using fruitful points to form tuples
    R'_i ← Subset of red points found by a circular range search on DS_t with b_i as query point and radius ρ_i
    for all r_j ∈ R'_i do
        O ← O ∪ {(b_i, r_j)}
    end for
end for
return O
```

Additionally, the nearest neighbor-finding structure (Voronoi diagram and point location structure) can be implemented in $O(n)$ space and has a query time of $O(\log n)$ [8].

The query algorithm is shown as Algorithm 2. The first step identifies the set F of blue points by querying DS_f and the second step identifies tuples to output by querying DS_t.

We argue that a tuple (b_i, r_j) is reported if and only if $d(q, b_i) + d(b_i, r_j) \leq \delta$. Suppose that (b_i, r_j) is reported. Thus, r_j is found when DS_t is queried with b_i and radius $\rho_i = \delta - d(q, b_i)$, i.e., $d(b_i, r_j) \leq \delta - d(q, b_i)$, i.e., $d(q, b_i) + d(b_i, r_j) \leq \delta$. On the other hand, suppose that $d(q, b_i) + d(b_i, r_j) \leq \delta$. Then, $d(q, b_i) + d(b_i, nn_i) \leq \delta$, where nn_i is the red point nearest to b_i. Thus, by Lemma 1, b_i is a fruitful point. Hence DS_t is queried with b_i and radius $\rho_i = \delta - d(q, b_i) \geq \delta - (\delta - d(b_i, r_j)) = d(b_i, r_j)$, which implies that r_j is found by the query. Hence, (b_i, r_j) is reported. This establishes the correctness of the query algorithm.

The first step takes $O(\log n + k_F) = O(\log n + k)$ time, where $k_F = |F|$ is the number of fruitful points and k is the size of the output for the 2-CTRQ query. Crucially, note that $k_F \leq k$ since, by definition, every fruitful point belongs to at least one output tuple. The second step takes $O(\log n + k_i)$ time for each $b_i \in F$, where k_i is the size of output of the circular range search from b_i, i.e., the number of red points reported, hence the number of tuples that b_i is a part of in the answer set. Therefore, the total time for the second step is $O(\sum_{b_i \in F}(\log n + k_i)) = O((k+1)\log n)$ since $\sum_{b_i \in F} k_i = k$. Thus, the second step dominates the overall query time, which is $O((k+1)\log n)$. Each of the structures $\mathcal{V}_r$, DS_f, and DS_t uses $O(n)$ space, so the total space used is $O(n)$.

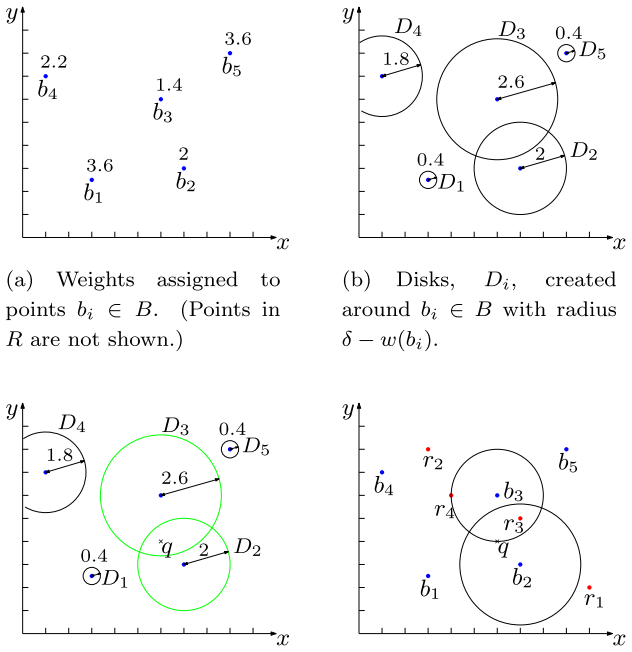
2.1. An example

We illustrate our algorithm using the point-set of Fig. 1 with $\delta = 4$. First, for each point $b_i \in B$, we compute its nearest neighbor in R and assign the distance as weight $w(b_i)$ to b_i as shown in Fig. 2a. Next, we filter out all the b_i's for which $w(b_i)$ is larger than δ and create a disk, D_i, around each remaining weighted point $b_i \in B$ with radius $\delta - w(b_i)$, as shown in Fig. 2b. (Since, in this example, we do not have any b_i with weight larger than δ, we do not discard any weighted points.) For a given query point, q, as shown in Fig. 2c, we find the disks D_2 and D_3 (shown in green) stabbed by q. The points b_2 and b_3 corresponding to the centers of the stabbed disks D_2 and D_3, respectively, are the fruitful points. Next, as shown in Fig. 2d, we perform two circular range search queries, one centered at b_2 with radius $\delta - d(q, b_2) = 4 - 1.4 = 2.6$, and one centered at b_3 with radius $\delta - d(q, b_3) = 4 - 2 = 2$. We then form tuples (b_2, r_3) with total distance 3.41 from q, since r_3 is the only point reported in the circular range search with b_2 as center, and (b_3, r_3) and (b_3, r_4) with total distance 3.41 and 4, respectively, from q, since r_3 and r_4 are the points reported in the circular range search with b_3 as center.

2.2. Space–time trade-off

Ideally, one would like a query time of $O(\log n + k)$ for the 2-CTRQ problem instead of $O((k+1)\log n) = O(\log n + k \log n)$. The roadblock to this is the second step, i.e., the time to perform a circular range query from each fruitful blue point. For a fruitful point b_i, the circular range query takes $O(\log n + k_i)$ time, where k_i is the number of red points reported. An important observation is that if $k_i \geq \log n$ then $O(\log n + k_i) = O(k_i)$; otherwise $O(\log n + k_i) = O(\log n)$. Based on this observation, we can divide the fruitful points into two sets; *heavy* fruitful points, b_i, for which $k_i \geq \log n$ and *light* fruitful points, b_i, for which $k_i < \log n$. The heavy fruitful points are, in fact, "good" points because the output size (k_i) dominates the query overhead ($O(\log n)$), which allows us to absorb the latter inside the former and obtain a query bound of $O(k)$ for such points. On the other hand, for the light fruitful points, the query overhead dominates the output size and, therefore, results in the $k \log n$ term in the query time.

Therefore, our approach is to use a simpler data structure for the light fruitful points that avoids the $O(\log n)$ query overhead. For the heavy fruitful points, we will continue to use the circular range search structure DS_t. Note that we do not know ahead of time which blue points are fruitful and which among these are light or heavy, as this depends on q. Therefore, we will build the simpler data structure for each blue point $b_i \in B$. This structure is merely a linked list, L_i, which contains the $(\log n)$-nearest red points to b_i, stored in non-decreasing order of their distances from b_i (ties broken arbitrarily).



(a) Weights assigned to points $b_i \in B$. (Points in R are not shown.)

(b) Disks, D_i, created around $b_i \in B$ with radius $\delta - w(b_i)$.

(c) Disks found to be stabbed by q (shown in green).

(d) Tuples found via circular range search.

Fig. 2. A running example to illustrate our solution for fixed distance 2-CTRQ with $\delta = 4$.

To answer a query q, we first identify the set, F, of fruitful blue points using DS_f. Next, we classify each $b_i \in F$ as light or heavy as follows. Let $\hat{r}_i$ denote the last point in L_i, i.e., the one that is $(\log n)$-nearest from b_i. It is easy to see that if $d(q, b_i) + d(b_i, \hat{r}_i) \leq \delta$ then b_i is heavy; otherwise b_i is light. For each b_i that is light, we scan L_i in non-decreasing order and output (b_i, r_j) for each r_j that is seen such that $d(q, b_i) + d(b_i, r_j) \leq \delta$. (We stop the scan as soon as this condition fails.) The query time is $O(k_i + 1)$, which, as desired, avoids the $O(\log n)$ query overhead that was incurred previously. Thus, the total query time for all light points queried is $O(k + k_F) = O(k)$, since $k_F \leq k$. For each heavy point b_i, we query DS_t, which takes $O(\log n + k_i) = O(k_i)$ time (since $k_i \geq \log n$). Thus the query time for all heavy points queried is $O(k)$. Thus the total time for the second step of the 2-CTRQ query is $O(k)$. The overall query time, inclusive of the first step, is hence $O(\log n + k)$, which improves upon the bound of $O((k+1)\log n)$ obtained previously. This improvement comes at the expense of a slightly higher space bound of $O(n \log n)$ since we store a list L_i of length $\log n$ with each $b_i \in B$.

The preceding approach can be generalized. For a user-specified integer parameter t, $1 \leq t \leq \log n$, we store a list of the t-nearest red points with each blue point. For a given q, there can be at most $\frac{k}{t}$ heavy fruitful points, where k is the output size, since the output size of a heavy blue point is, by definition, at least t. Therefore, we perform at most $\frac{k}{t}$ circular range queries, each of which takes $O(\log n + k_i)$ time and yields a total of $O(\frac{k}{t} \log n + k) = O(\frac{k}{t} \log n)$ over all heavy points. The time for the light points continues to be $O(k)$, as before. Therefore, the overall time for the 2-CTRQ query, inclusive of the first step, is $O(\log n + \frac{k}{t} \log n + k) = O((1 + \frac{k}{t}) \log n)$ and the space requirement is $O(nt)$. For example, if $t = \sqrt{\log n}$ then we get a scheme with $O(n\sqrt{\log n})$ space and $O(\log n + k\sqrt{\log n})$ query time.

Theorem 1. *A fixed-distance* 2-CTRQ *query on a set of n red and blue points can be answered in* $O((1 + \frac{k}{t}) \log n)$ *time using* $O(nt)$ *space, where t is a user-specified parameter,* $1 \leq t \leq \log n$ *and k is the output size. In particular,* $t = 1$ *(resp.* $t = \log n$*) yields a solution with* $O((k+1)\log n)$ *(resp.* $O(\log n + k)$*) query time and* $O(n)$ *(resp.* $O(n \log n)$*) space.*

3. Variable distance 2-CTRQ

In this section, we discuss our solution for the variable distance 2-CTRQ problem. As before, we present a two-step solution for this problem, where, in the first step, we find the set of fruitful points and, in the second step, we explore B and R using the fruitful points to form and report the tuples. However, the challenge now is that δ is not known at preprocessing time. Therefore it is not possible to build the disk stabbing data structure DS_f to find the fruitful blue points. In the rest of this section we discuss alternative methods to find these fruitful points. (The second step of our query algorithm remains unchanged as it does not rely on knowing δ at preprocessing time.)

It is easy to see that Lemma 1 continues to hold even if δ is variable. Thus, as before, we precompute for each $b_i \in B$ its closest red point $r_j \in R$ and store with b_i the weight $w(b_i) = d(b_i, r_j)$. Thus, the problem of finding fruitful points of B for q and δ boils down to finding a set of weighted points $b_i \in B$ such that $d(q, b_i) + w(b_i) \leq \delta$. In Section 3.1, we describe an algorithm for finding fruitful points that is based on a certain geometric transformation. Then in Section 3.2, we describe a method based on additively-weighted higher-order Voronoi diagrams to find the fruitful points. In Section 3.2.1 we show how to improve the storage requirement of the method of Section 3.2 using a graph traversal technique to record certain information compactly. In Section 3.2.2 we discuss briefly how to also improve the query time by using a probabilistic method to suitably partition the points in preprocessing. (The query time remains deterministic.) Finally, in Section 3.3, we describe the overall algorithm for the variable distance 2-CTRQ problem.

3.1. Geometric transformation-based algorithm for fruitful points

The main idea behind this approach is to transform the weighted blue points in $\mathbb{R}^2$ to (unweighted) points in $\mathbb{R}^4$ in such a way that the solution to a certain half-space range search problem on these points in $\mathbb{R}^4$ yields the desired fruitful points in $\mathbb{R}^2$. We propose the following transformation:

1. Point $b_i = (x_i, y_i)$ in $\mathbb{R}^2$, with weight $w(b_i)$ is mapped to point $b'_i = (x_i, y_i, w(b_i), z_i)$ in $\mathbb{R}^4$, where $z_i = x_i^2 + y_i^2 - w(b_i)^2$.
2. Query point $q = (x_q, y_q)$ in $\mathbb{R}^2$ with query distance δ is mapped to hyperplane q' in $\mathbb{R}^4$ defined by the equation $a_1 x + a_2 y + a_3 w + a_4 z = c$, where $a_1 = -2x_q, a_2 = -2y_q, a_3 = 2\delta, a_4 = 1$, and $c = \delta^2 - x_q^2 - y_q^2$.

The following lemma establishes the desired property of this transformation.

Lemma 2. *Consider the transformation given above. Then, for any $b_i \in B$, $d(q, b_i) + w(b_i) \leq \delta$ if and only if b'_i is on or below q'.*

Proof. We have

$$
\begin{aligned}
& d(q, b_i) + w(b_i) \leq \delta \\
\Leftrightarrow \quad & \sqrt{(x_q - x_i)^2 + (y_q - y_i)^2} \leq \delta - w(b_i) \\
\Leftrightarrow \quad & x_q^2 + y_q^2 + x_i^2 + y_i^2 - 2x_q x_i - 2y_q y_i \leq \delta^2 + w(b_i)^2 - 2\delta w(b_i) \\
\Leftrightarrow \quad & (-2x_q)x_i + (-2y_q)y_i + (2\delta)w(b_i) + \left(x_i^2 + y_i^2 - w(b_i)^2\right) \leq \delta^2 - x_q^2 - y_q^2 \\
\Leftrightarrow \quad & a_1 x_i + a_2 y_i + a_3 w(b_i) + a_4 z_i \leq c \\
\Leftrightarrow \quad & b'_i \text{ is on or below } q' \qquad \square
\end{aligned}
$$

Based on Lemma 2, we transform the weighted blue points in $\mathbb{R}^2$ to points in $\mathbb{R}^4$ and create a data structure for half-space range search on these points in $\mathbb{R}^4$. Given q and δ, we compute q' and query the data structure with q' to find the points that are on or below q', hence the fruitful points. The query algorithm is shown as Algorithm 3.

As mentioned in Section 1.4, for $d \geq 4$, there is a data structure for halfspace range search in $\mathbb{R}^d$ that has a query time of $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n + k)$ and uses $O(n)$ space [1]. We use this structure for DS_{4hs}, with $d = 4$. Therefore, the space used by this approach is $O(n)$ and the query time is $O(\sqrt{n} \log^{O(1)} n + k_F)$.

Lemma 3. *Let B be a set of blue points in the plane, where $|B| \leq n$. The points of B that are fruitful with respect to a query point q and query distance δ can be computed in $O(\sqrt{n} \log^{O(1)} n + k_F)$ time and $O(n)$ space, where k_F is the number of fruitful points.*

Algorithm 3: Geometric-transformation-based method.

Input: DS_{4hs}: Data structure for half space range search in $\mathbb{R}^4$; q: query point; δ: query distance
Output: F: set of fruitful blue points
1: $F \leftarrow \emptyset$
2: $q' \leftarrow$ hyperplane in $\mathbb{R}^4$ obtained by applying the above-mentioned transformation on q and δ
   // `Perform half-space range search`
3: $F' \leftarrow$ set of points returned when DS_{4hs} in queried with q'
4: **for all** $b'_i \in F'$ **do**
5: $\quad b_i \leftarrow$ point in $\mathbb{R}^2$ corresponding to b'_i
6: $\quad F \leftarrow F \cup \{b_i\}$
7: **end for**
8: **return** F

3.2. Finding fruitful points via additively-weighted higher-order Voronoi diagrams

Recall that each of the k_F fruitful blue points, b_i, satisfies a distance constraint relative to q; specifically, $d(q, b_i) + w(b_i) \leq \delta$. Therefore, if the points of B are ordered by non-decreasing weighted distance from q, then the fruitful points are exactly the first k_F points in this order. In other words, the fruitful points constitute the set of k_F-closest neighbors of q, under the weighted distance function.

Thus, we find fruitful points by finding the k_F-closest neighbors of q, under weighted distance. To accomplish this, we could build on the points of B an order-k_F additively-weighted Voronoi diagram in the plane [21]. This diagram partitions the plane into cells and associates with each cell a set of k_F blue points (called *generators*) that are the k_F-closest neighbors, under weighted distance, of any point $p \in \mathbb{R}^2$ lying in the cell. (The ordering of the generators can be different for different points p in the cell.) Thus, the generators associated with the cell containing q (determined via point location in the diagram) constitute the desired fruitful points.

Unfortunately, this approach is not feasible since q and δ are not known beforehand, hence k_F is not known at preprocessing time. Therefore, we build the following data structure. We create a complete binary tree, $\mathcal{T}$, on the points of B where at most $h = c \log n$ points, in any order, are stored at the leaves, for some constant $c \geq 1$. (In Section 3.2.2, we show that a more careful distribution of points can further improve the query time.) At each non-leaf node $v \in \mathcal{T}$ (including the root), we store an order-h additively-weighted Voronoi diagram $\mathcal{V}_h(v)$ built on the set, $B(v)$, of points stored at the leaves of the subtree rooted at v. We also build a data structure for doing point location in $\mathcal{V}_h(v)$.

Given q and δ, we explore $\mathcal{T}$, starting at the root, to determine the fruitful points as follows. At node v, if v is a leaf node then we compute the weighted distance from q to the points stored at v, and report the points with distance at most δ as fruitful points. Otherwise, we query $\mathcal{V}_h(v)$. This involves locating q in $\mathcal{V}_h(v)$, retrieving the h generators of the cell containing q, computing the weighted distance from q to each generator, and checking this distance against δ. If there is a generator with weighted distance larger than δ, then $k_v < h$, where k_v is the number of fruitful points in $B(v)$. We report the generators that have weighted distance at most δ from q as fruitful points and abandon searching below v. Otherwise, $k_v \geq h$. In this case, we do not report anything, but instead continue to explore the two children of v.

We analyze the query time as follows. For a given q and δ, each node v of $\mathcal{T}$ that is explored in the search is of one of two types: (i) v is a non-leaf node for which $k_v \geq h$. We do not report any fruitful points at v but instead explore v's children. We call v a *non-terminal* node. And (ii) v is a non-leaf for which $k_v < h$ or a leaf. We report fruitful points (if any) at v and abandon the search below v. We call v a *terminal* node.

The time spent at a non-terminal node, for point location and for checking the generators, is $O(\log n + h) = O(\log n)$. Similarly, for the time spent at a terminal node that is not a leaf of $\mathcal{T}$. At a terminal node that is a leaf, we spend $O(h) = O(\log n)$ time. Therefore, to complete the query time analysis, we need to only upper-bound the number of terminal and non-terminal nodes.

The number of terminal nodes is at most twice the number of non-terminal nodes, since a terminal node that is not the root has a non-terminal node as parent and since $\mathcal{T}$ is binary. Therefore the total number of terminal and non-terminal nodes is at most three times the number of non-terminal nodes plus one. (The one extra node in the count handles the case where the root is the only terminal node in the search.)

To upper-bound the number of non-terminal nodes, consider any such node v. We charge one unit for v and distribute this charge uniformly over the $k_v \geq h$ fruitful points in v's subtree, which results in a charge of at most $1/h$ to each of these fruitful points. In the worst case, any fruitful point in $\mathcal{T}$ is charged at most $1/h$ in this way for each non-terminal node to whose subtree it belongs. Since the height of $\mathcal{T}$ is $O(\log n)$, this results in a total charge of $O((\log n)/h) = O(1)$ per fruitful point. It follows that the total number of terminal and non-terminal nodes is $O(1 + k_F)$. Thus, the query time is $O((1 + k_F) \log n)$.

The size of $\mathcal{V}_h(v)$ at a node v, inclusive of the h generators stored with each cell, is $O(h^2(|B(v)| - h)) = O(h^2|B(v)|)$ [21]. For nodes v at the same depth in $\mathcal{T}$, the sets $B(v)$ are pairwise disjoint. Therefore, the total size of $\mathcal{V}_h(v)$ at these nodes v is $O(h^2 n)$. Since the maximum depth in $\mathcal{T}$ is $O(\log n)$, the total space used is $O(h^2 n \log n) = O(n \log^3 n)$.

3.2.1. Reducing the space used

The space usage can be improved as follows. Note that the size of the symmetric difference of the sets of the generators associated with any two consecutive cells of a $\mathcal{V}_h(\cdot)$, is two, i.e., consecutive cells have $h - 1$ common generators. Based on this observation, we do not store the list of h generators with each cell; instead, we create a query data structure to find the list of generators for a cell. We first create a dual graph $\mathcal{G}$ of $\mathcal{V}_h(\cdot)$, where each vertex of $\mathcal{G}$ corresponds to a cell in $\mathcal{V}_h(\cdot)$ and two vertices in $\mathcal{G}$ are connected by an edge if and only if the corresponding cells in $\mathcal{V}_h(\cdot)$ share an edge. Note that the number of vertices in $\mathcal{G}$ is the same as the number of cells in $\mathcal{V}_h(\cdot)$, which we denote by $|\mathcal{V}_h(\cdot)|$. Next, we create a spanning tree of $\mathcal{G}$ and duplicate each edge in it and perform an Euler tour on it. (An Euler tour exists because the resulting graph is undirected and all the vertices have even degree.)

We record the "time" when a vertex is visited during the tour using a sequence of integers $1, 2, \ldots, 2|\mathcal{V}_h(\cdot)| - 1$. (The largest label is easily seen to be $2|\mathcal{V}_h(\cdot)| - 1$.) During the traversal, we maintain a list of h "active" generators, i.e., generators associated with the Voronoi cell corresponding to the current vertex in tour. Each edge traversal results in deletion of a generator and insertion of another generator in this list. With each generator, we associate a time interval when it was active during the traversal. Note that each edge traversal ends an interval for a generator (deletion of a generator) and

starts a new interval for another generator (insertion of a generator). Therefore, the total number of intervals is $O(|\mathcal{V}_h(\cdot)|)$. We store these intervals in an interval tree $\mathcal{T}_I$. With each Voronoi cell, we also store a time-stamp of the visit of the corresponding vertex of the dual graph during the traversal. (If multiple time-stamps are associated with a vertex, we pick any one.)

To find the generators of the cell containing q, we query $\mathcal{T}_I$ with the time-stamp associated with the cell to find the intervals stabbed by it and report the generators corresponding to the stabbed intervals. Since there are $O(|\mathcal{V}_h(\cdot)|)$ intervals, $\mathcal{T}_I$ uses $O(|\mathcal{V}_h(\cdot)|)$ space and reports the intervals that are stabbed in $O(\log|\mathcal{V}_h(\cdot)| + h) = O(\log n)$ time.

With this technique, the space at any node v of $\mathcal{T}$ (inclusive of the interval tree) is $O(h(|B(v)| - h)) = O(h|B(v)|)$, rather than $O(h^2|B(v)|)$ [21], since generators are not stored explicitly with each cell of $\mathcal{V}_h(v)$. This results in an overall space bound of $O(n\log^2 n)$.

Lemma 4. *Let B be a set of blue points in the plane, where $|B| \leq n$. The points of B that are fruitful with respect to a query point q and query distance δ can be computed in $O((1+k_F)\log n)$ time and $O(n\log^2 n)$ space, where k_F is the number of fruitful points.*

3.2.2. Improving the query time

We describe a different way of creating the data structure described in Section 3.2 so that the desired fruitful points can be found in $O(\log n + k_F)$ time. The data structure is built in preprocessing using a probabilistic method; however, the query time is deterministic. The approach follows closely the techniques used by Chazelle et al. [5], so our discussion here is brief. We refer the reader to [5] for details.

One drawback of assigning $h = c\log n$ points in any order to the leaf nodes of $\mathcal{T}$ is that, for some q and δ, it is possible that the number of fruitful points is h (for example) and all of them are stored in a single leaf node. In this case the fruitful points are each assigned a charge of $1/h$ at $O(\log n)$ nodes, which is too much as it leads to the $O((1+k_F)\log n)$ query time shown in Section 3.2.1. To avoid this situation, we use a probabilistic approach to assign points to the leaf nodes of $\mathcal{T}$ is such a way that for any pair of intermediate nodes $v, w \in \mathcal{T}$, if w is a child of v, then the following holds true for any q:

$$\left|N_{h(w)}\big(B(w), q\big) - N_{h(v)}\big(B(v), q\big)\right| \geq d\log\left|B(w)\right|,$$

where $N_{h(\cdot)}(B(\cdot), q)$ represents the set of $h(\cdot) = c\log|B(\cdot)|$ points in $B(\cdot)$ that are closest to q and $d > 0$ is a constant. This condition ensures that sufficiently many new fruitful points are "exposed" at w to absorb the cost of exploring at w. It has been shown by Chazelle et al. [5], in the context of circular range search, that there exist constants c, d, and n_0 such that for $n \geq n_0$, the desired assignment is possible with probability at least $1/2$. Their result does not depend on the underlying distance metric and, therefore, is valid for our weighted distance too.

Based on this discussion, we create the data structure mentioned in Section 3.2 with the following modifications. We create a complete binary tree $\mathcal{T}$ on points of B and assign $\max\{h, n_0\}$ points of B to the leaves of $\mathcal{T}$ as described above. Also, at each intermediate node $v \in \mathcal{T}$, we build an order-$h(v)$ additively-weighted Voronoi diagram, $\mathcal{V}_{h(v)}(v)$, on $B(v)$ and a point location data structure on $\mathcal{V}_{h(v)}(v)$. The query process remains the same.

It is clear that the space usage for this data structure remains $O(n\log^2 n)$. By an analysis similar to the one in [5] it can be shown that the query time for this approach is $O(\log n + k_F)$.

Lemma 5. *Let B be a set of blue points in the plane, where $|B| \leq n$. The points of B that are fruitful with respect to a query point q and query distance δ can be computed in $O(k_F + \log n)$ time using a data structure of size $O(n\log^2 n)$ computed probabilistically, where k_F is the number of fruitful points.*

*3.3. Putting it all together: The overall algorithm for variable distance 2-*CTRQ

Recall that the first step is to find the fruitful blue points with respect to q and δ and the second step is to explore from each fruitful blue point and form and report (blue, red) tuples. Section 3.1, Section 3.2.1, and Section 3.2.2 have addressed the first problem and the bounds are summarized in Lemma 3, Lemma 4, and Lemma 5, respectively.

The second step is identical to the second step for the fixed distance 2-CTRQ problem in Section 2. Theorem 1 summarizes the bounds for the fixed distance 2-CTRQ problem and these are also the bounds for the second step for that problem. For simplicity, we will focus on the pair of bounds at the extremes, i.e., $O(n)$ (resp. $O(n\log n)$) space and $O((k+1)\log n)$ (resp. $O(\log n + k)$) query time, corresponding to the parameter value $t = 1$ (resp. $t = \log n$), although it is possible to consider bounds corresponding to other values of t as well.

It is now straightforward to combine a pair of bounds, one for each step, to obtain the overall bounds for the variable distance 2-CTRQ problem. (Note that the number, k_F, of fruitful points is at most the output size k.) Theorem 2 below lists the four possibilities. (Of the six possibilities, two have the same space and query time bounds and one is strictly worse than the others; these have been eliminated.)

Theorem 2. *A variable distance 2-*CTRQ *query on a set of n red and blue points can be answered in either (i) $O(\sqrt{n}\log^{O(1)} n + k\log n)$ time using $O(n)$ space, or (ii) $O(\sqrt{n}\log^{O(1)} n + k)$ time using $O(n\log n)$ space, or (iii) $O((1+k)\log n)$ time using $O(n\log^2 n)$*

space, or (iv) $O(\log n + k)$ time using $O(n \log^2 n)$ space. (The bounds in (iv) involve a data structure that is built probabilistically in preprocessing; the query time is deterministic.)

4. Handling more than two colors

In this section, we discuss an approach to solve the CTRQ problem for $m > 2$ colors using the solution for 2-CTRQ. Recall the CTRQ problem for $m > 2$ colors: Let S be a set of n points in $\mathbb{R}^2$, where each point is assigned a color c_i from a palette of m colors ($m > 2$). Let C_i be the set of points of color c_i and let $n_i = |C_i|$. Note that $C_i \cap C_j = \emptyset$ if $i \neq j$ and $\bigcup_{i=1}^{m} C_i = S$. Let $CS = (c_1, \ldots, c_m)$ be a given ordering of the colors. We wish to preprocess S into a suitable data structure so that for any query point q in $\mathbb{R}^2$ and a distance $\delta > 0$, we can report all tuples $(p_1, \ldots, p_m)$, where $p_i \in C_i$, such that $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$. Note that here $\delta > 0$ can be fixed or variable.

As before, we present a two-step solution for this problem: In the first step, we find the set of fruitful points of color c_1 and, in the second step, we explore from each such fruitful point the points of color $c_2, \ldots, c_m$ to form and report the tuples. However, the challenge now is that we have more than two colors and therefore Lemma 1 cannot be applied directly to assign weights to the points of color c_1. Also, due to the same reason, in the second step we cannot use circular range search to explore the points of color $c_2, \ldots, c_m$ to form and report the tuples. In the rest of this section we discuss how to overcome these issues.

As mentioned in Section 1.3, the idea of fruitful points is central to our approach. As in the case of two colors, we define fruitful points for $m > 2$ colors as follows: For a given ordering, $CS = (c_1, \ldots, c_m)$, a point of color c_1 is *fruitful* if it is part of at least one output tuple for the query point, q, and query distance δ. The following lemma provides a generalization of Lemma 1 for $m > 2$ colors.

Lemma 6. *For any $p_1 \in C_1$, let $(p_1, \ldots, p_m)$ be an m-tuple of $C_1 \times \cdots \times C_m$ that minimizes $\sum_{i=2}^{m} d(p_{i-1}, p_i)$. Let q be a query point and δ the query distance. Then p_1 is fruitful if and only if $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$.*

Proof. ($\Leftarrow$) If $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$ then the tuple $(p_1, \ldots, p_m)$ is output. Hence, by definition, p_1 is fruitful.

($\Rightarrow$) Suppose p_1 is fruitful. Thus, there is a tuple $(p_1, p'_2, \ldots, p'_m)$ that is output. Thus, $d(q, p_1) + d(p_1, p'_2) + \sum_{i=3}^{m} d(p'_{i-1}, p'_i) \leq \delta$. Since $\sum_{i=2}^{m} d(p_{i-1}, p_i) \leq d(p_1, p'_2) + \sum_{i=3}^{m} d(p'_{i-1}, p'_i)$, we have $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$. □

Lemma 6 suggests the following preprocessing step to determine the fruitful points at query time. For each $p_1 \in C_1$, we compute an m-tuple, $(p_1, \ldots, p_m)$, of $C_1 \times \cdots \times C_m$ that minimizes $\sum_{i=2}^{m} d(p_{i-1}, p_i)$ and associate with p_1 the real number $\sum_{i=2}^{m} d(p_{i-1}, p_i)$ as a weight $w(p_1)$. The following lemma provides a useful characterization of the desired m-tuple, $(p_1, \ldots, p_m)$.

Lemma 7. *For some $p_1 \in C_1$, let $P = (p_1, \ldots, p_m)$ be an m-tuple of $C_1 \times \cdots \times C_m$ that minimizes $\sum_{i=2}^{m} d(p_{i-1}, p_i)$. Then for the point $p_i \in C_i$, $1 \leq i < m$, $P_i = (p_i, \ldots, p_m)$ is an $(m - i + 1)$-tuple of $C_i \times \cdots \times C_m$ that minimizes $\sum_{j=i+1}^{m} d(p_{j-1}, p_j)$.*

Proof. The case $i = 1$ is true by assumption since $P_1 = P$. Let $i > 1$ and assume that P_i is not optimal for p_i and let $P'_i = (p_i, p'_{i+1}, \ldots, p'_m) \in C_i \times \cdots \times C_m$ be optimal. Then $\sum_{j=2}^{i} d(p_{j-1}, p_j) + d(p_i, p'_{i+1}) + \sum_{j=i+2}^{m} d(p'_{j-1}, p'_j) < \sum_{j=2}^{i} d(p_{j-1}, p_j) + \sum_{j=i+1}^{m} d(p_{j-1}, p_j) = \sum_{j=2}^{m} d(p_{j-1}, p_j)$, which contradicts the optimality of P. □

Consider the optimal tuple $P = (p_1, \ldots, p_m)$ defined in Lemma 7. Then Lemma 7 implies that for the point $p_{m-1} \in C_{m-1}$, the point $p_m \in C_m$ minimizes $d(p_{m-1}, p_m)$, i.e., p_m is a nearest neighbor of p_{m-1}. Similarly, for the point $p_{m-2} \in C_{m-2}$, the point $p_{m-1} \in C_{m-1}$ minimizes $d(p_{m-2}, p_{m-1}) + d(p_{m-1}, p_m) = d(p_{m-2}, p_{m-1}) + w(p_{m-1})$, where $w(p_{m-1}) = d(p_{m-1}, p_m)$ is a *weight* assigned to p_{m-1}. In general, for the point $p_i \in C_i$, $1 \leq i < m$, the point $p_{i+1} \in C_{i+1}$ minimizes $d(p_i, p_{i+1}) + w(p_{i+1})$, where $w(p_{i+1}) = \sum_{j=i+2}^{m} d(p_{j-1}, p_j)$ is the weight of p_{i+1}, i.e., $w(p_{i+1})$ is the length of the shortest path from p_{i+1} that visits one point from each of $C_{i+2}, \ldots, C_m$, in that order. (For notational convenience, we take $w(p_m) = 0$.) Thus, p_{i+1} minimizes the sum of $w(p_{i+1})$ and the Euclidean distance between p_i and p_{i+1}.

Our goal is to eventually assign weights (as defined above) to the points of C_1. Based on the above discussion, we preprocess the sets $C_{m-1}, C_{m-2}, \ldots, C_1$, in that order, and assign weights to the points of each set. For any point $p_i \in C_i$, $1 \leq i < m$, $w(p_i)$ is computed by finding the point $p_{i+1} \in C_{i+1}$ that minimizes the sum of $w(p_{i+1})$ and the Euclidean distance between p_i and p_{i+1}. This involves building an additively-weighted Voronoi diagram on the points of C_{i+1}, using the weights found in the previous step (or weight zero if $i + 1 = m$) and querying this with each point of C_i.

With this weight assignment, the problem of finding fruitful points of C_1 for a q and δ boils down to finding (similar to the case of $m = 2$ colors) a set of weighted points $p_1 \in C_1$ such that $d(q, p_1) + w(p_1) \leq \delta$ and, therefore, we can find the fruitful points by using the solution in Section 2 or Section 3 (for the fixed distance or the variable distance CTRQ problem, respectively).

Recall that the second step is to explore the points of color $c_2, \ldots, c_m$ to form and report the tuples, i.e., for each fruitful point $p_1 \in C_1$, the goal is to identify all the $(m-1)$-tuples $(p_2, \ldots, p_m) \in C_2 \times \ldots \times C_m$ such that $d(q, p_1) + \sum_{i=2}^{m} d(p_{i-1}, p_i) \leq \delta$,

i.e., $d(p_1, p_2) + \sum_{i=3}^{m} d(p_{i-1}, p_i) \leq \delta - d(q, p_1)$, i.e., $d(p_1, p_2) + w(p_2) \leq \delta - d(q, p_1)$, i.e., $p_2 \in C_2$ is a fruitful point for the "query" point p_1 and query distance $\delta - d(q, p_1)$ with color sequence $CS' = (c_2, \ldots, c_m)$. (Note that p_2 is an input point that functions as a "query" point here.) Therefore, for the second step, we recursively solve the variable distance CTRQ problem using the fruitful points as "query" points with query distance and ordering of colors adjusted appropriately. (The problem to be solved is of the variable distance type since successive query distances (e.g., $\delta - d(q, p_1)$) are not known beforehand even if δ itself is.) We terminate the recursion when only two colors are left to process. At this point, we use the solution for variable distance 2-CTRQ. We form tuples as we backtrack and finally report the tuples so formed as the answer set.

4.1. Analysis

Let $Q_{fix}(m, n)$ (resp., $Q_{var}(m, n)$) be the query time for the fixed (resp., variable) distance CTRQ problem on n points and m colors. Let $Q_{F,fix}(n)$ (resp., $Q_{F,var}(n)$) be the time to find fruitful points in a set of n points for the fixed (resp., variable) distance CTRQ problem. For $m \geq 3$, the above discussion leads to the following recurrence relations.

$$Q_{fix}(m, n) = Q_{F,fix}(n) + k_{F_2} \times Q_{var}(m-1, n), \tag{1}$$

$$Q_{var}(m, n) = Q_{F,var}(n) + k_{F_2} \times Q_{var}(m-1, n), \tag{2}$$

where k_{F_2} is the number of fruitful points of the second color in CS for q and δ. We use $m = 3$ as the base case for the above recurrences. For $m = 3$, the query time for the fixed distance CTRQ problem is

$$Q_{fix}(3, n) = Q_{F,fix}(n) + k_{F_2} \times Q_{var}(2, n)$$

We use the results mentioned in Theorem 2 for $Q_{var}(2, n)$ to derive the corresponding space and time bounds for $Q_{fix}(3, n)$. For example, using $Q_{var}(2, n) = O(\log n + k)$ (with space usage $O(n \log^2 n)$), we get

$$Q_{fix}(3, n) = O(\log n + k_{F_2}) + k_{F_2} O(\log n) + \sum_{i=1}^{k_{F_2}} k_i = O\big((k+1)\log n\big).$$

Here, k_i is the output size for ith fruitful point and $\sum_{i=1}^{k_{F_2}} k_i = k$. (Note that the product of the number of fruitful points of each color is at most the output size k and, hence, the number of fruitful points of each color is at most k.) The space usage for $Q_{F,fix}(n)$ is $O(n)$, therefore, the overall space usage is bounded by the space usage of the variable distance 2-CTRQ problem, and hence, is $O(n \log^2 n)$. Note that $Q_{fix}(3, n)$ can be further improved to $O(\log n + k)$ by following the technique mentioned in Section 2.2 while still using $O(n \log^2 n)$ space. Other space and query time bounds for the fixed distance CTRQ problem, for $m = 3$, are (*i*) $O(\log n + k\sqrt{n} \log^{O(1)} n)$ time using $O(n)$ space, and (*ii*) $O((1+k)\log n)$ time using $O(n \log^2 n)$ space. Similarly, the various space and query time for the variable distance CTRQ problem, for $m = 3$, can be verified to be (*i*) $O((1+k)\sqrt{n} \log^{O(1)} n)$ time using $O(n)$ space, (*ii*) $O((1+k)\log n)$ time using $O(n \log^2 n)$ space, (*iii*) $O(\log n + k)$ time using $O(n \log^2 n)$ space (using a probabilistically-computed data structure).

Using $m = 3$ as the base case for the recurrence relations in Eqs. (1) and (2), the space and time bounds for the fixed (resp. variable) distance CTRQ problem can be verified to be as listed in Theorem 3 (resp. Theorem 4). (Of the four possibilities, one is strictly worse than others in both space and time, and is hence eliminated.) Note that the space bounds are independent of m. This is because the structure built on points of color c_i uses space proportional to $|C_i| = n_i$ (e.g. $O(n_i)$ or $O(n_i \log^2 n_i)$), so the total space is proportional to $\sum_{i=1}^{m} n_i = n$.

Theorem 3. *A fixed distance* CTRQ *query on a set of n points in* $\mathbb{R}^2$*, where each point is assigned a color* c_i *from a palette of m colors* ($m > 2$) *can be answered in either* (*i*) $O(\log n + km\sqrt{n} \log^{O(1)} n)$ *time using* $O(n)$ *space, or* (*ii*) $O((1+km)\log n)$ *time using* $O(n \log^2 n)$ *space, or* (*iii*) $O(\log n + k)$ *time using* $O(n \log^2 n)$ *space. (The bounds in* (*iii*) *are based on a probabilistically-computed data structure; the query time is deterministic.)*

Theorem 4. *A variable distance* CTRQ *query on a set of n points in* $\mathbb{R}^2$*, where each point is assigned a color* c_i *from a palette of m colors* ($m > 2$) *can be answered in either* (*i*) $O((1+km)\sqrt{n} \log^{O(1)} n)$ *time using* $O(n)$ *space, or* (*ii*) $O((1+km)\log n)$ *time using* $O(n \log^2 n)$ *space, or* (*iii*) $O(\log n + k)$ *time using* $O(n \log^2 n)$ *space. (The bounds in* (*iii*) *are based on a probabilistically-computed data structure; the query time is deterministic.)*

5. Conclusion

We have presented a new query-retrieval problem on sets of colored points. Given a query point q and a distance δ, the goal is to report all tuples of points, one of each color in a prescribed ordering of the colors, such that the total distance from q through the sequence is no more than δ. We have given efficient solutions for the fixed distance and variable distance version of this problem, where δ is known beforehand or known only at query time, respectively.

We close by mentioning some avenues for further research. An obvious open question is to improve upon the algorithms presented here. A second direction is to restrict the search to a query range (e.g., an axes-parallel rectangle) specified by the user. This is quite natural since a user is often interested in visiting points (e.g. venues) in a geographical region of interest (e.g. a few city blocks). For this range version of CTRQ problem, we are able to show that it is unlikely that a solution exists that simultaneously achieves low space (close to linear) and low query time (polylogarithmic). This is via a reduction from the well-known Set Intersection problem [7]. However, it would still be useful to explore algorithms that are highly efficient in terms of space or query time or demonstrate a good trade-off between the two. Finally, it would be interesting to extend the CTRQ problem to graphs (i.e., road networks), where distances are measured along the edges of the graph (instead of in the Euclidean plane).

Acknowledgement

The authors thank the reviewers for many useful comments that helped improve the paper. In particular, the methods presented in Section 3.2.1 and Section 3.2.2 build on ideas suggested by one of the reviewers.

References

[1] P. Afshani, T.M. Chan, Optimal halfspace range reporting in three dimensions, in: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2009, pp. 180–186.
[2] P.K. Agarwal, Range searching, in: CRC Handbook of Discrete and Computational Geometry, CRC Press, Inc., 2004, pp. 809–837.
[3] P.K. Agarwal, J. Erickson, Geometric Range Searching and Its Relatives, Contemp. Math., vol. 223, American Mathematical Society Press, 1999, pp. 1–56.
[4] X. Cao, L. Chen, G. Cong, J. Guan, N.-T. Phan, X. Xiao, KORS: Keyword-aware optimal route search system, in: Proceedings of the 29th International Conference on Data Engineering (ICDE), IEEE, 2013, pp. 1340–1343.
[5] B. Chazelle, R. Cole, P. Preparata, C. Yap, New upper bounds for neighbor searching, Inf. Control 68 (1–3) (1986) 105–124.
[6] H. Chen, W.-S. Ku, M.-T. Sun, R. Zimmermann, The partial sequenced route query with traveling rules in road networks, Geoinformatica 15 (3) (2011) 541–569.
[7] P. Davoodi, M. Smid, F. van Walderveen, Two-dimensional range diameter queries, in: LATIN 2012: Theoretical Informatics, Springer, 2012, pp. 219–230.
[8] M. de Berg, O. Cheong, M. Van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer, 2008.
[9] H. Edelsbrunner, Algorithms in Combinatorial Geometry, vol. 10, Springer, 1987.
[10] T. Hashem, T. Hashem, M.E. Ali, L. Kulik, Group trip planning queries in spatial databases, in: Advances in Spatial and Temporal Databases, Springer, 2013, pp. 259–276.
[11] Y. Kanza, E. Safra, Y. Sagiv, Route search over probabilistic geospatial data, in: Advances in Spatial and Temporal Databases, Springer, 2009, pp. 153–170.
[12] Y. Kanza, E. Safra, Y. Sagiv, Y. Doytsher, Heuristic algorithms for route-search queries over geographical data, in: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2008, pp. 1–10.
[13] R. Levin, Y. Kanza, Interactive traffic-aware route search on smartphones, in: Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, ACM, 2012, pp. 1–8.
[14] R. Levin, Y. Kanza, TARS: traffic-aware route search, Geoinformatica (2013) 1–40.
[15] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, S.-H. Teng, On trip planning queries in spatial databases, in: Advances in Spatial and Temporal Databases, Springer, 2005, pp. 273–290.
[16] J. Li, Y. Yang, N. Mamoulis, Optimal route queries with arbitrary order constraints, IEEE Trans. Knowl. Data Eng. 25 (5) (2013) 1097–1110.
[17] X. Ma, S. Shekhar, H. Xiong, Multi-type nearest neighbor queries in road networks with time window constraints, in: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2009, pp. 484–487.
[18] X. Ma, S. Shekhar, H. Xiong, P. Zhang, Exploiting a page-level upper bound for multi-type nearest neighbor queries, in: Proceedings of the 14th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2006, pp. 179–186.
[19] X. Ma, C. Zhang, S. Shekhar, Y. Huang, H. Xiong, On multi-type reverse nearest neighbor search, Data Knowl. Eng. 70 (11) (2011) 955–983.
[20] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer, 1988.
[21] H. Rosenberger, Order-k Voronoi diagrams of sites with additive weights in the plane, Algorithmica 6 (1–6) (1991) 490–521.
[22] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann, 2006.
[23] M. Sharifzadeh, M. Kolahdouzan, C. Shahabi, The optimal sequenced route query, VLDB J. 17 (4) (2008) 765–787.
[24] D. Zhang, C.-Y. Chan, K.-L. Tan, Nearest group queries, in: Proceedings of the 25th International Conference on Scientific and Statistical Database Management, ACM, 2013, pp. 1–12.
[25] X. Zhang, W.-C. Lee, P. Mitra, B. Zheng, Processing transitive nearest-neighbor queries in multi-channel access environments, in: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, ACM, 2008, pp. 452–463.
[26] B. Zheng, K.C. Lee, W.-C. Lee, Transitive nearest neighbor search in mobile environments, in: Sensor Networks, Proceedings of the International Conference on Ubiquitous, and Trustworthy Computing, IEEE, 2006, pp. 14–21.