Neighbours on a Grid*

Andrej Brodnik[1] and J. Ian Munro[2]

[1] Luleå University, Luleå, Sweden; on leave from Institute of Mathematics, Physics, and Mechanics, Ljubljana, Slovenia

[2] University of Waterloo, Ontario, Canada

Abstract. We address the problem of a succinct static data structure representing points on an $M \times M$ grid ($M = 2^m$ where m is size of a word) that permits to answer the question of finding the closest point to a query point under the L_∞ or L_1 norm in constant time. Our data structure takes essentially minimum space. These results are extended to d dimensions under L_∞.

1 Introduction

Given a set of points, a query point, and a distance metric, the closest neighbour problem is that of determining the point of the set whose distance from the query point is minimal. If the query point is a member of the given set then it will be the solution, and if two or more points are of equal distance from the query point we choose one of them arbitrarily. This problem arises in many areas such as modeling of robot arm movements and integrated circuits layouts (cf. [26]). The problem has been heavily studied in the $\mathbb{R}^2$ contiguous domain where it is solved using Voronoi diagrams (cf. [29]). Furthermore, the problem can be generalized by considering the points as multidimensional records in which individual fields are drawn from an ordered domain (cf. [22]).

In this paper we describe an essentially *minimum space*, *constant time* solution to the *static* problem in a *discrete* $M \times M$ point universe under the L_∞ or L_1 norm. Our structure uses $M^2 + o(M^2)$ bits of memory, which differes from the information theoretic minimum by only the lower order term when a random half of the points are present. The solution is a combination of two approaches: under the first, the universe is represented by a bit map; and under the second, each point of the universe "knows" who is its closest neighbour – it has a pointer to the closest point. The advantage of the first approach is that it minimizes space required at the expense of query time. The second guarantees constant response time at the expense of space. We obtain advantages of both. Finally, we generalize the solution to d dimensional M^d-point universe (d is some predefined constant) under the L_∞ norm using $M^d + o(M^d)$ bits of space.

Our general approach is to divide the universe into regions we call *tiles* (cf. cells in [3, 4]) each covering dm ($m = \lg M$) discrete points of the universe. If any of the points in a tile are present, then we simply store a bit map representation of a tile; and if a tile is empty we store a *candidate* value. This value is the closest point in the entire set to the

---

* This work was done while the first author was a graduate student at the University of Waterloo and was supported in part by the NSERC of Canada, grant number A-8237, and the ITRC of Ontario. E-mail: `Andrej.Brodnik@IMFM.Uni-Lj.SI` and `IMunro@UWaterloo.CA`.

middle of the tile. Note that because of the choice of size of a tile, either option requires the same amount of space – dm bits. In this paper, we show that using appropriately shaped tiles the closest neighbour to any query point in a universe can be determined by inspecting constant number of tiles, while table lookup technique facilitates finding the closest element in a tile represented by a bit map.

The paper consists of three major parts: first we define the problem with some additional notation, and review the literature. The bulk of the paper deals with solutions to the problem in two dimensions. The final section explains how to extend the solution to one and d dimensions, and poses some open questions.

2 Definitions and Background

In general we deal with the set of points in d-dimensional space, $T = (x_1, \ldots, x_d)$ ($x_i \in \{0, \ldots, M-1\}$), where d is a predefined constant and coordinates are orthogonal. The distance between points $T_1 = (x_{1,1}, \ldots, x_{1,d})$ and $T_2 = (x_{2,1}, \ldots, x_{2,d})$ is measured by $\delta_f(T_1, T_2) = (\sum_{i=1}^{d} \left|(x_{1,i} - x_{2,i})^f\right|)^{\frac{1}{f}}$ for a real parameter $1 \leq f \leq \infty$ (cf. [26, p.222]). The parameter f also defines L_f, the *norm of the space*. The family of the distance functions defined this way satisfy the *triangle inequality*. Formally:

Definition 1. Let $\mathcal{N}$ ($|\mathcal{N}| = N$) be a subset of points from the universe $\mathcal{M} = [0 \ldots M]^d$. The *static closest neighbour* problem is to represent these points in a data structure so that given a query point, $T \in \mathcal{M}$, the closest member of $\mathcal{N}$, under norm L_f can be found efficiently.

Note if the query point is in the set $\mathcal{N}$, then it is its own closest neighbour, and if several points are the same distance from the query point any of them is a satisfactory answer.

In this paper we focus our attention on $f = \infty$ which, as the limit as $f \to \infty$, defines $\delta_\infty(T_1, T_2) = \max_{0<i\leq d} |x_{1,i} - x_{2,i}|$. First, when $d = 1$ distance functions for all f are the same. Secondly, for two dimensional space ($d = 2$) Lee and Wong ([21]) proved, that search for the closest neighbour under L_∞ is computationally equivalent to search under L_1. However, direct use of their transformation increases the space, violating one of our key concerns. Therefore, we develop a solution for L_1 from scratch.

We use the extended random access machine model (ERAM) whose instruction set includes integer multiplication and division, and bitwise Boolean operations (cf. [6] and MBRAM in [14]). We assume that one register is large enough to store one coordinate of a point – it is m bits ($m = \lg M$) wide.

Our solutions consist of two parts: first we explain how to search for the closest neighbour in a small, b-point universe ($b = dm$), and second how to search in a big M^d-point universe ($M^d = 2^b$). Throughout the paper we assume that all divisions which define the size of a problem do not produce a remainder. It can be verified that by dropping this assumption, all algorithms and data structures remain correct, though the third order terms of the size of a data structure may change.

2.1 The Literature Background

Finding the closest element in a set to a query element is an important problem arising in many subfields of computer science, including computational geometry, pattern recognition, VLSI design, data compression and learning theory (cf. [11, 22, 26]). As noted, there are several versions of the problem. Clearly, the number of dimensions, d, the distance norm, typically L_2, L_1 or L_∞, and model of computation impact the appropriate choice of methods.

First consider continuous searching space (domain), $\mathbb{R}$. In a one-dimensional case all norms are equivalent. Under the comparison based model there is a simple logarithmic lower bound for the static problem, which is matched by a binary search algorithm. The same bound carries over to the dynamic problem, and is matched by any balanced tree algorithm (cf. [11]).

In two dimensions and under the Euclidean norm, L_2, the problem is also known as a *post-office* problem ([20]) and is related to the *point-location* problem (cf. [13]). The most common approach to solve it is to use Voronoi diagrams (cf. [29]) which gives, under the comparison based model, logarithmic deterministic running time using $O(N)$ words of memory (for logarithmic running time solutions see also [8, 9, 28]). Chang and Wu in [7] went outside the comparison based model. Using hashing they achieved constant running time at the cost of using, in the worst case, $O(N^2 + M)$ words. Under the same model Bentley et al. described constant expected time solution using $O(N)$ words of memory ([5]). All Voronoi diagram based approaches have similar bounds also under norms L_1 and L_∞, although the diagrams have different shapes ([21]). Yao in [29] reports that most deterministic solutions generalize to higher dimensions at the expense of using $O(N^{2^{d+1}})$ words, though there is no further discussion.

Unlike the static problem, there is no known efficient deterministic solution to the dynamic version in 2 or more dimensions. There are, however, expected poly-logarithmic time algorithms to maintain Voronoi diagrams under the comparison based model ([10]) and constant time probabilistic solutions under the random access machine model with integer division and multiplication (cf. [16, 27]). Both of these use $O(N)$ words.

Next consider a discrete domain upon which, in combination with a bounded universe, we concentrate in this work. The bounded discrete universe permits completely different data structures. For example, Karlsson ([18]), and Karlsson, Munro and Robertson ([19]) under a comparison based model, adapt the van Emde Boas et al. one-dimensional stratified trees ([15]) to two dimensions. Thus they achieve, in an $M \times M$ universe under norms L_1 and L_∞ a worst case run time of $O(\log^{(2)} M)$ using $O(N)$ words for the static problem, and for the dynamic problem a run time of $O(\log^{3/2} M)$ using $O(N \log M)$ words. On the other hand, Murphy and Selkow go outside the comparison based model and present a constant expected time probabilistic solution under L_∞ that uses $O(N)$ words of memory ([25]).

Finally, the only known lower bound under the cell probe model (cf. [24]) on the number of necessary bits for a data structure which would still allow constant query time in a bounded discrete universe is the trivial one, $\left\lceil \lg \binom{M}{N} \right\rceil$.

3 Preliminaries

3.1 Registers

The registers are m bits wide. Since in a d-dimensional universe d registers are necessary to store coordinates of an arbitrary point, we always consider d registers together. For convenience we use $b \equiv dm$ to denote the number of bits in such a grouping. We also assume b is a power of d. This simplifies the presentation, but alters only lower order terms in the space requirement. In two dimensions we view the grouping of b bits of a pair of registers as a *square register* (cf. double precision numbers):

Definition 2. A *square register* x_r consists of p rows and p columns of bits, where $p^2 = b$. The bit $x_r.\mathtt{b}[i,j]$, for $0 \le i, j < p$, is positioned in the i^{th} column of j^{th} row, and the bit $x_r.\mathtt{b}[0,0]$ is the least significant bit of x_r. A square register with all bits set below its major (minor) diagonal is denoted by $P^{\setminus}$ ($P^{/}$).

Square registers are just a helpful logical abstraction – a different interpretation of a number ([6]). Thus a processor can perform all standard operations on them (e.g. shifts, bitwise Boolean operations, etc.). We will find it helpful to determine the extreme set bits in a square register, that is the left, the right, the top, and the bottom most set bit. If there is more than one extreme set bit in a certain direction (e.g. the left most), the ties are broken arbitrarily. In particular:

Theorem 3. *Using a fixed table of size $O(M^{\epsilon})$, for any $0 < \epsilon \le 1$, we can find the extremal set bits of a square register in constant time.*

Proof. We divide the register into k pieces, all of the same shape in their geometric interpretation. Each $\frac{b}{k}$-bit piece is used as an index to a table. The corresponding table value gives the extremal bit, in each direction of the piece. Each table entry takes $4 \cdot (\lg b - \lg k)$ bits and there are $2^{\frac{b}{k}} = 2^{\frac{2m}{k}} = M^{\frac{2}{k}}$ entries. A search takes time $O(k)$. To satisfy the space constraint, and avoid lower order terms we set ϵ to a little less than $\frac{2}{k}$, that is k is a little more than $\frac{2}{\epsilon}$. $\square$

3.2 Geometry

The geometric background of this subsection provides a simple description of the area of the universe where the closest neighbour to the query point can lie. Though the background is essentially norm independent, it is most efficiently applied under the L_∞ and L_1 norms where the above mentioned area is small enough that it can be exhaustively searched. By a *circle* we mean a set of points that are equidistant from some central point under the norm that is being used. So, for example, under L_∞ a circle is a square aligned with the axes and under L_1 it has a diamond shape. For illustration of individual terms introduced in this section see Fig. 1. We divide (tile) the universe $\mathcal{M}$ into tiles with the following properties: they have to tile the plane in a regular pattern such that from the coordinates of a point we can efficiently compute the tile in which it lies; and if we put a circle with diameter m anywhere on a plane, it must lie in $O(1)$ tiles. Obviously there are many different tilings which satisfy the above conditions, but for the purpose of simplicity of explanation we choose to define:

Definition 4. A *tile* is a $p \times p$ square. The tiles sharing a common edge with a given tile are its *direct neighbours*.

For convenience, we number the direct neighbours of a tile $\mathscr{T}_0$ in a clockwise manner starting with $\mathscr{T}_1$ at the top. Hence, the direct neighbours of $\mathscr{T}_0$ are $\mathscr{T}_1$ through $\mathscr{T}_4$. Next:

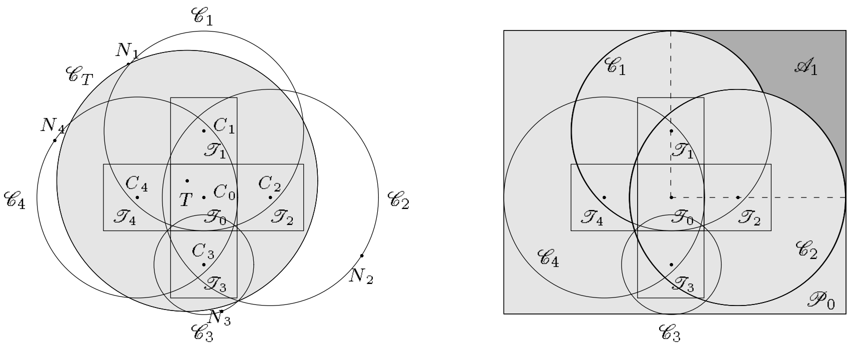

Fig. 1. Circle of candidates $\mathscr{C}_T$ with empty circles $\mathscr{C}_{1-4}$, and empty circles with the enclosing polygon $\mathscr{P}_0$ and the corner area $\mathscr{A}_1$.

Definition 5. Let C_x be the middle of a tile $\mathscr{T}_x$, then $\mathscr{C}_x$, the *empty circle* of $\mathscr{T}_x$, is the largest circle with centre C_x whose *interior* is empty. Thus, if N_x is the closest neighbour of C_x, N_x lies on the circumference of $\mathscr{C}_x$. Note that C_x need not be a point in the discrete domain.

Based on the tile definition and empty circles we define an *enclosing rectangle*:

Definition 6. Let $\mathscr{T}_0$ be a tile and let $\{\mathscr{C}_i\}$ be the empty circles of its respective direct neighbours. Then, $\mathscr{P}_0$, the *enclosing rectangle* of $\mathscr{T}_0$, is the smallest rectangle that has sides parallel to $\mathscr{T}_0$ and includes all empty circles.

A particularly interesting part of the plane is the area which is inside the enclosing rectangle, but outside the empty circles. In order to properly identify this area we define first a *wedge*:

Definition 7. Let $\mathscr{T}_0$ be a tile, and $\mathscr{P}_0$ its enclosing rectangle as above. Let $\mathscr{T}_i$ and $\mathscr{T}_j$ (where $j = (i \bmod 4) + 1$) be direct neighbours of $\mathscr{T}_0$. Further, draw lines from C_0 through C_i, and from C_0 through C_j. Then the rectangle defined by these two lines and sides of the enclosing rectangle is called a *wedge* of the enclosing rectangle.

Obviously, if we draw lines from C_0 through middles of all direct neighbours, we split $\mathscr{P}_0$ into 4 wedges. Inside the wedge we define a *corner area*:

Definition 8. Consider the wedge defined by direct neighbours $\mathscr{T}_i$ and $\mathscr{T}_j$ as above. Then the area that lies inside the wedge and outside empty circles of all direct neighbours is called a *corner area* $\mathscr{A}_i$.

Since the number of corner areas is at most the number of wedges, which is itself at most 4, it follows that:

Lemma 9. *There are at most* 4 *corner areas.*

The last term used in our discussion is the *circle of candidates*:

Definition 10. Let the point T lie on the tile $\mathscr{T}_0$ and let C_i $(0 < i \leq 4)$ be middle points of respective direct neighbours with their closest neighbours N_i. Further, among all points N_i, let N_x be the closest point to T. Then the circle $\mathscr{C}_T$ with centre at T and N_x on its circumference is called the *circle of candidates*.

Finally, based on Definition 10 and Definition 5, we can restrict the location of the closest neighbour of a given point:

Lemma 11. *Let T be a point on $\mathscr{T}_0$. Then the closest neighbour of T lies on the circumference or inside the circle of candidates $\mathscr{C}_T$, and outside the interior of the empty circles $\mathscr{C}_i$, where $0 < i \leq 4$.*

Lemma 11 concludes our brief geometrical excursion and hints at the idea behind our algorithm: compute empty circles of direct neighbours, compute the circle of candidates and search its intersection with the union of complements of empty circles. Later we will show, that under norm L_∞ the intersection lies inside corner areas, and that the corner areas are small enough that we can perform an exhaustive search on them.

4 When Circles are Squares: L_∞

We explore first the L_∞ norm. Under this norm, "circles" have a square shape.

4.1 The Small Universe

The small universe is a square containing order $b = 2m$ points. We represent it by a square register. We map the point $T = (x_1, x_2)$ to a bit $\texttt{b}[x_1, x_2]$ of the register and denote the point's presence or absence by setting the bit to 1 or 0 respectively.

The search algorithm is based on the idea of a search inside 4 distinct *search regions* (see Fig. 2):

Definition 12. Let $T = (x_1, x_2)$ be a query point, at which a *left border line* and a *right border line* with slopes $+45°$ and $-45°$ respectively, cross. These lines divide a plane into four *search regions* $\mathscr{R}_\uparrow$, $\mathscr{R}_\rightarrow$, $\mathscr{R}_\downarrow$, and $\mathscr{R}_\leftarrow$.

In order to search one region at a time, we eliminate points from other regions. This requires that we generate proper masks. These masks are square registers with all bits of $\mathscr{R}_\uparrow$ (respectively $\mathscr{R}_\rightarrow$, $\mathscr{R}_\downarrow$, and $\mathscr{R}_\leftarrow$) and none others set.

Lemma 13. *Let the left and right border lines cross at point $T = (x_1, x_2)$. Then we can generate masks for all four search regions in constant time using $O(m)$ bits of space.*

Proof. Each border line splits the plane into a positive and a negative half-plane, where the negative half-plane lies below the respective border line. In Fig. 2 the half-planes are denoted by $\mathscr{R}_{+\backslash}$, $\mathscr{R}_{-\backslash}$, $\mathscr{R}_{+/}$ and $\mathscr{R}_{-/}$.

It is easy to see that masks for half-planes $\mathscr{R}_{-\backslash}$ and $\mathscr{R}_{-/}$ can be generated in constant time by proper shifting of $P^{\backslash}$ and $P^{/}$ respectively. Finally, masks for the search regions can be computed using formulae: $\mathscr{R}_\downarrow = \mathscr{R}_{-\backslash} \wedge \mathscr{R}_{-/}$, $\mathscr{R}_\uparrow = \overline{\mathscr{R}_{-\backslash} \vee \mathscr{R}_{-/}}$, $\mathscr{R}_\leftarrow = \overline{\mathscr{R}_{-/} \vee \mathscr{R}_\uparrow}$, and $\mathscr{R}_\rightarrow = \overline{\mathscr{R}_{-\backslash} \vee \mathscr{R}_\uparrow}$. $\square$

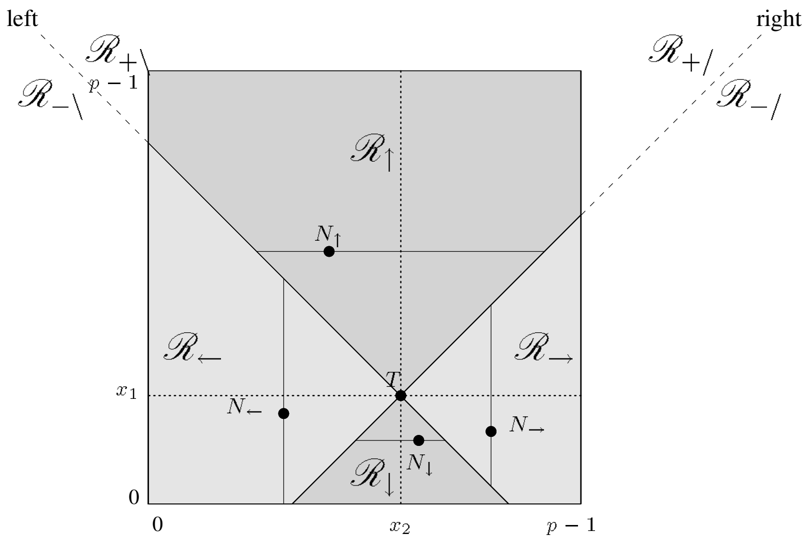

Fig. 2. Four search regions in a plane.

Using Lemma 13 we can easily prove:

Theorem 14. *Let the universe be a set of $b = 2m$ discrete points on a square grid and let $\mathcal{N}$ be some subset of this universe. Then there is an algorithm to find the closest neighbour to a query point in $\mathcal{N}$ under norm L_∞ in constant time using b bits for a data structure and $O(M^\epsilon)$ bits for internal constants.*

Internal constants in the text of the theorem are the constants used by the algorithm and thus they are the same for all possible sets.

Proof. As a data structure representing the set we use the obvious bit map stored in a b-bit square register. The search algorithm divides the plane into four search regions from Definition 12. It then determines the closest point to the query point T in each region. Because of the norm we are using this amounts, for $\mathscr{R}_\downarrow$ and $\mathscr{R}_\uparrow$, to finding the point in the row closest to T and, for $\mathscr{R}_\leftarrow$ and $\mathscr{R}_\rightarrow$, to finding the point in the closest column (see Fig. 2). Since the universe is represented by a square register, we can employ Theorem 3. □

4.2 The Big Universe

Since the sides of a tile are parallel to the axes of the coordinate system, this also implies the orientation of an enclosing rectangle $\mathscr{P}_0$ to be parallel to the axes of the coordinate system. Further, the empty circles and the circle of candidates are also squares with sides parallel to coordinate axes. Thus, circles, tiles, and enclosing rectangles all have parallel sides. Finally, a tile contains $b = p \times p$ points.

The remaining entities of interest are the corner areas. Under the L_∞ norm, the corner areas have the following useful property:

Lemma 15. *Let $\mathscr{T}_0$ be some tile. Then there are at most four corner areas associated with $\mathscr{T}_0$, each of which lies on at most six tiles.*

Proof. By Lemma 9 there are at most four corner areas. Without loss of generality we confine our attention to $\mathscr{A}_1$ (see Fig. 3). Let a_1 and a_2 be the radii of $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively. First, assume that either a_1 or a_2 is at least p and let the sides of $\mathscr{A}_1$ be of lengths u and v (see the left diagram in Fig. 3). Then, since the distance from C_2 to the top side of $\mathscr{P}_0$ is $v + a_2 = p + a_1$ and since the distance from C_1 to the right side of $\mathscr{P}_0$ is $u + a_1 = p + a_2$, $u + v = 2p$ and, consequently, $0 \leq u, v \leq 2p$. Furthermore, the area of $\mathscr{A}_1$ is $u \cdot v \leq p^2 = b$. Thus, $\mathscr{A}_1$ lies on at most 6 tiles.

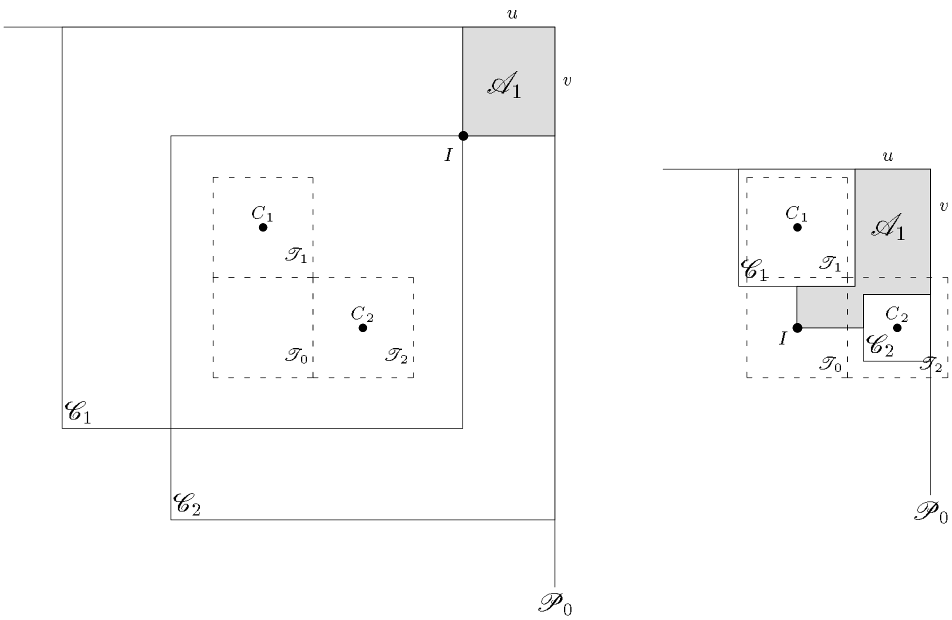

Fig. 3. The corner area $\mathscr{A}_1$ is limited by a distance between centres of empty circles $\mathscr{C}_1$ and $\mathscr{C}_2$.

On the other hand, if $a_1, a_2 \leq p$, then $\mathscr{A}_1$ lies on both tiles adjacent to $\mathscr{T}_1$ and $\mathscr{T}_2$ (the right diagram in Fig. 3). It is not hard to verify that $\mathscr{A}_1$ lies on at most three other tiles and that this occurs when $p \geq a_1, a_2 \geq \frac{p}{2}$. □

The most important consequence of Lemma 15 is that corner areas can be exhaustively searched in constant time using Theorem 14.

The next property relates the circle of candidates and the enclosing rectangle:

Lemma 16. *Under the norm L_∞, the circle of candidates lies inside the enclosing rectangle.*

Proof. Let the middle of tile $\mathscr{T}_0$ be point $C_0 = (0, 0)$ and let $T = (x_T, y_T)$ be a query point where $-\frac{p}{2} < x_T, y_T < \frac{p}{2}$. By definition the radius of the circle of candidates is $r_T = \min_{1 \le i \le 4} \delta_\infty(T, N_i)$, where N_i are closest neighbours of middles of direct neighbours. Thus, we have to show, for all points U on the circumference of the enclosing rectangle $\mathscr{P}_0$, that $\delta_\infty(T, U) \ge r_T$.

Without loss of generality, we may assume that the closest point to T on the circumference is $W = (x_W, y_W)$ where $y_W = y_T$. By the definition of the enclosing polygon (Definition 6), x_W is either $-\delta_\infty(C_4, N_4) - p$ or $\delta_\infty(C_2, N_2) + p$. Again without loss of generality, we may assume $x_W = -\delta_\infty(C_4, N_4) - p$ and thus $\delta_\infty(T, W) = |x_T - p - \delta_\infty(C_4, N_4)| = |x_T - p| + \delta_\infty(C_4, N_4)$. However, since tile $\mathscr{T}_4$ is immediately to the left of $\mathscr{T}_0$, $C_4 = (-p, 0)$ and hence $\delta_\infty(T, C_4) = |x_T - p|$. Therefore, using a triangle inequality, we get $\delta_\infty(T, W) = \delta_\infty(T, C_4) + \delta_\infty(C_4, N_4) \ge \delta_\infty(T, N_4) \ge \min_{1 \le i \le 4} \delta_\infty(T, N_i) = r_T$. □

We can now state:

Theorem 17. *Let the universe be the grid of $M \times M$ points and let $\mathscr{N}$ be some subset. Then there is an algorithm which finds the closest neighbour in $\mathscr{N}$ to a query point under the norm L_∞ in constant time using $M^2 + \frac{M^2}{2 \lg M} + O(M^\epsilon)$ bits of memory.*

Proof sketch. First, we tile universe with an $\frac{M}{p} \times \frac{M}{p}$ array of b-point tiles. With each tile there is associated a bit, which indicates whether the tile is nonempty. If the tile is nonempty it is represented by a bit map and if it is empty we store coordinates of the closest neighbour to the centre of the tile. Since the space needed for either of the stored entities is the same (b bits), the whole data structure occupies $(\frac{M}{p})^2 \cdot b + (\frac{M}{p})^2 = M^2 + \frac{M^2}{2 \lg M}$ bits. An additional $O(M^\epsilon)$ bits are used for the table to find extreme set bits.

Next, circles (empty and circle of candidates) are implicitly defined by the centre and a point on the circumference. Therefore, all circles can be constructed in constant time.

Finally, according to Lemma 11, to find the closest neighbour of T we search that part of interior of circle of candidates $\mathscr{C}_T$ which is outside empty circles $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{C}_3$, and $\mathscr{C}_4$. By Lemma 16, $\mathscr{C}_T$ lies inside the enclosing polygon, and thus it is sufficient to search the corner areas. According to Lemma 15, each corner area overlaps at most six tiles. Finally, by Theorem 14 each tile can be searched in constant time, and hence the closest neighbour can be found in constant time. □

5 When Circles are Diamonds: L_1

Under L_1, the distance between $T_1 = (x_1, y_1)$ and $T_2 = (x_2, y_2)$ is defined as $\delta_1(T_1, T_2) = |x_1 - x_2| + |y_1 - y_2|$. The "circles" under this norm have a diamond shape; they are squares rotated $45°$ from the axes. Though the mapping

$$x' = x - y \qquad y' = x + y \tag{1}$$

of the point (x, y) under L_1 into the point (x', y') under L_∞ preserves the closest neighbourhood property (cf. [21]) and keeps all values integer, its straightforward application increases the domain to $2M \times 2M$ points and so quadruples the space bound of the data structure. Therefore, to achieve the space bound $M^2 + o(M^2)$ bits, we sketch a solution built from scratch:

Theorem 18. *Let the size of a universe be at most $M \times M$ points and let $\mathcal{N}$ be a subset. Then there is an algorithm which finds the closest neighbour in $\mathcal{N}$ to a query point under norm L_1 in constant time using $M^2 + \frac{M^2}{2m} + M \cdot \frac{2m+1}{\sqrt{m}} + O(M^\epsilon)$ bits of memory ($m = \lg M$).*

Proof sketch. The search itself employs ideas explained in Lemma 11 and used under L_∞. This time we tile the universe with diamonds (see above) which requires a few minor changes in some definitions, but keeps Lemmata 15 and 16 correct. As in Theorem 17, our data structure consists of an array of bit maps (or pointers) and an array of bits. Since this time the sides of a tile are not aligned with coordinate axes we have more tiles (the border ones are broken though) and thus a larger third order term in a space bound.

In contrast with L_∞ the bit map is stored in *two* m-bit registers. The points of a tile are mapped into registers using slightly modified mapping from the eq. (1): odd rows are mapped into one register and the even ones into the other register. This ensures the total number of bits in both registers is the same as the number of points in the tile. It also permits us to use Theorem 3 for search inside each register and thus inside a non-empty tile (for details see [6, § 5.4.3.]). □

6 Final Improvements and Generalizations

Obviously, Theorem 17 remains valid for the one-dimensional case with an appropriate change of tiles to line segments. Furthermore, using hyper-cuboidal registers (yet another interpretation of a number) and hyper-cubic tiling of d-dimensional space, we can extend the result also to higher dimensions under L_∞:

Theorem 19. *Let the d-dimensional universe be set of M^d points and let $\mathcal{N}$ be a subset of it. Then there is an algorithm which finds the closest neighbour in $\mathcal{N}$ of a query point under norm L_∞ by searching $O(d^2 \cdot 2^d) = O(1)$ tiles (each in $O(1)$ time) and using $M^d + \frac{M^d}{d \lg M} + O(M^\epsilon)$ bits of memory.*

The only aspect of the extension that requires care is keeping the number of tiles being searched down to $O(d^2 \cdot 2^d)$. A naive implementation searches $O(d^2 \cdot 6^d)$.

Further, by a simple replacement of the pointer to the closest point to the middle of the tile with a pointer to the tile on which this closest point lies we get:

Corollary 20. *Let the universe have dimensions $\{S_1, S_2, \ldots S_d\}$, where $0 < W = \prod_{i=1}^{d} S_i \leq dm \cdot 2^{dm+1}$, and let $\mathcal{N}$ be a subset of it. Then there is an algorithm which finds the closest neighbour in $\mathcal{N}$ of a query point under norm L_∞ in constant time using $W + \frac{W}{d \lg M} + O(M^\epsilon)$ bits of memory.*

In the described solution we use table lookup technique to find extreme set bits in square registers. This technique requires $O(M^{\epsilon})$ bits of space. However, there is another algorithm which also finds extreme set bits under the same model in constant time, but it requires only $O(m)$ bits of memory (cf. [6, Theorem 4.3]). This algorithm employs the technique of performing many Boolean or small domain operations on a computer word in a single processor instruction. We call the technique *word-size parallelism* ([6, Chapter 4]) although the approach has been used by others without giving it a name. For example, computation of $\lfloor \lg x \rfloor$ ([17]), bitonic sorting ([1]), tight packing of fields ([2]) can all be considered specific instances of this technique. Replacing the table lookup technique with the above mentioned algorithm obviously decreases the last term in space requirements of our solutions.

In our solutions we side-stepped the problem of the data structure initialization. It can be seen that it takes $2d$ sweeps of the universe (cf. [12, 23]) and the time necessary to construct the table for search of extreme set bits. Since the time of one sweep is proportional to the number of tiles, the initialization takes time $O(\frac{M^d}{\log M} + M^{\epsilon})$.

Finally, our approach does not seem to work under other norms L_f ($1 < f < \infty$, e.g. Euclidean L_2). Under these norms each corner area can lie on $\Theta(\frac{M}{\sqrt{b}})$ tiles, and thus, we can not exhaustively search all of them (Lemma 15 does not hold). Even adding $O(1)$ more empty circles does not improve the situation significantly. The reason the technique does not seem to work is the difference in curvatures of the circle of candidates and the empty circles. However, there remains the open question of how many tiles the discrete points of a corner areas can intersect. We conjecture that this number is also too large. It is possible to use more than four, but still a constant number, of empty circles and obtain an *approximate* solution. It remains open to be determined how good an approximation this gives.

References

1. S. Albers and T. Hagerup. Improved parallel integer sorting without concurrent writting. In *3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 463–472, Orlando, Florida, 1992.
2. A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? In *27th ACM Symposium on Theory of Computing*, pages 427–436, Las Vegas, Nevada, 1995.
3. J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, 1979.
4. J.L. Bentley and H.A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13:155–168, 1980.
5. J.L. Bentley, B.W. Weide, and A.C. Yao. Optimal expected-time algorithms for closest-point problems. *ACM Transactions on Mathematical Software*, 6(4):563–580, December 1980.
6. A. Brodnik. *Searching in Constant Time and Minimum Space (*MINIMÆRES MAGNI MOMENTI SUNT*).* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1995. (Also published as technical report CS-95-41.).
7. C.-C. Chang and T.-C. Wu. A hashing-oriented nearest neighbor searching scheme. *Pattern Recognition Letters*, 14(8):625–630, August 1993.
8. B. Chazelle. An improved algorithm for the fixed-radius neighbor problem. *Information Processing Letters*, 16(4):193–198, May 13th 1983.
9. B. Chazelle, R. Cole, F.P. Preparata, and C. Yap. New upper bounds for neighbor searching. *Information and Control*, 68(1–3):105–124, 1986.

10. Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, September 1992.
11. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.
12. C.R. Dyer and A. Rosenfeld. Parallel image processing by memory-augmented cellular automata. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(1):29–41, January 1981.
13. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1987.
14. P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 1, pages 1 – 66. Elsevier, Amsterdam, Holland, 1990.
15. P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(1):99–127, 1977.
16. A. Faragó, T. Linder, and G. Lugosi. Nearest neighbor search and classification in $O(1)$ time. *Problems of Control and Information Theory*, 20:383 – 395, 1991.
17. M.L. Fredman and D.E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.
18. R.G. Karlsson. *Algorithms in a Restricted Universe*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1984. (Also published as technical report CS-84-50.).
19. R.G. Karlsson, J.I. Munro, and E.L. Robertson. The nearest neighbor problem on bounded domains. In W. Brauer, editor, *Proceedings 12$^{\text{th}}$ International Colloquium on Automata, Languages and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 318–327. Springer-Verlag, 1985.
20. D.E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, Massachusetts, 1973.
21. D.T. Lee and C.K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-storage applications. *SIAM Journal on Computing*, 9(1):200–211, February 1980.
22. K. Mehlhorn. *Data Structures and Algorithms: Multi-dimensional Searching and Computational Geometry*, volume 3. Springer-Verlag, Berlin, 1984.
23. R. Miller and Q.F. Stout. Geometric algorithms for digitized pictures on a mesh-connected computer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(2):216–228, March 1985.
24. P.B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *26$^{\text{th}}$ ACM Symposium on Theory of Computing*, pages 625–634, Montréal, Québec, Canada, 1994.
25. O.J. Murphy and S.M. Selkow. The efficiency of using k-d trees for finding nearest neighbors in discrete space. *Information Processing Letters*, 23(4):215–218, November 8$^{\text{th}}$ 1986.
26. F.P. Preparata and M.I. Shamos. *Computational Geometry*. Texts and Monographs in Computer Science. Springer-Verlag, Berlin, 2$^{\text{nd}}$ edition, 1985.
27. V. Ramasubramanian and K.K. Paliwal. An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation. *Pattern Recognition Letters*, 13(7):471–480, July 1992.
28. R.F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579 – 589, 1991.
29. F.F. Yao. Computational geometry. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 7, pages 343 – 389. Elsevier, Amsterdam, Holland, 1990.

This article was processed using the LaTeX macro package with LLNCS style and Andy's help