https://doi.org/10.31449/inf.v43i1.2679 Informatica **43** (2019) 45–51 **45**

Incremental 2-D Nearest-Point Search with Evenly Populated Strips

David Podgorelec and Denis Špelič
University of Maribor, Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, SI-2000 Maribor, Slovenia
E-mail: david.podgorelec@um.si, https://gemma.feri.um.si/

Keywords: incremental nearest-point search, dynamic partition, deterministic skip list

Received: October 30, 2018

The incremental nearest-point search successively inserts query points into the space partition data structure, and the nearest point for each of them is simultaneously found among the previously inserted ones. The paper introduces a new approach which solves this task in 2-D space in a nearly optimal manner. The proposed dynamic partition into parallel strips, each containing a limited number of points structured in the deterministic skip list, successfully prevents situations with over-populated strips, while its further advanced version with two perpendicular partitions and four categories of deterministic skip lists efficiently decreases the number of strips to be examined in a great majority of practical cases.

Povzetek: V članku je predstavljen algoritem za reševanje inkrementalnega problema najbližje točke, ki z dinamično delitvijo ravnine v vzporedne trakove preprečuje prenaseljenost le-teh, z dodatno delitvijo, pravokotno na prvo, pa se večinoma izogne tudi preiskovanju prevelikega števila trakov.

1 Introduction

Let us assume a set S of points in a space M and a query point $p \in M$. The *nearest-point search* (NPS) aims to find the point in S being the closest to p. In its most common form, M is a d-dimensional vector space (2-D in this paper), points correspond to their position vectors, and closeness is expressed with Euclidean distance. Applications can be found in a variety of domains, such as computational geometry [2, 20], geographic information systems – GIS [18], bioinformatics [21], image and video compression [3, pattern recognition [16], computer vision [12], robot motion planning [14]], telecommunications [9], and computer graphics [6]. There are a number of versatile generalizations where the distance metric between spatial points is extended to any quantitative measure of similarity between two generic objects, as one may also measure closeness of pairs of polygons, text strings, images, audio sequences etc.

If the distance is computable in $\Theta(1)$ time, the brute-force NPS is trivially handled in $\Theta(n)$ time, but the problem becomes more demanding in a *recurring NPS* where a larger set of query points has to be considered. A straightforward repetition of the brute-force approach results in $\Theta(n^2)$ time when applied to $\Theta(n)$ query points. In spatial domains, space partitioning can be used to bound the number of possible nearest-point candidates in each iteration [19]. The partition is accomplished by constructing a hierarchical or a grid data structure, typically a tree [7], the Voronoi diagram [10], a regular grid [4], or a multi-level organization of these structures [19]. Such a data structure is aimed to accelerate solving the point-location problem i.e. determination of the region where a query point lies. The methodology is based on the assumption that the points in the same region or those in adjacent regions are closer to each other than the points in more distinct regions. A static partition does not depend on point distribution or, eventually, it only utilizes the bounding box and/or the number of points in S. On the contrary, a dynamic partition maintains the numbers of points in all regions within the previously determined limits. Particularly in higher dimensions, where either the query time or storage space must be sacrificed, a user may also be satisfied by approximate solutions provided by the reasonably fast locality sensitive hashing technique, for example [17].

The *all nearest-points pairs problem* represents a special form of the recurring NPS, where the set of target points S and the set of query points coincide. For each point $p_i \in S$, we have to find $p_j \in S$, such that $i \neq j$ and the distance $|p_i p_j|$ is minimal. Two different approaches can be distinguished.

(a) The two-phase *preliminary points arrangement* approach firstly arranges all the points into the adequate regions of the space partition data structure. Then, in the second phase, this preliminary points arrangement is utilized to determine the nearest point for every query point.
(b) The *incremental nearest-point search* successively inserts points p_1, ..., p_n into the space partition data structure and simultaneously searches for their nearest points. The nearest point for p_i, $1 < i \leq n$, is determined among previously inserted points from $S_i = \{p_1, ..., p_{i-1}\}$ only. Note that $S_1 = \emptyset$.

In this paper, we introduce an original dynamic plane partition into parallel strips and utilize it to handle the incremental NPS in 2-D space. The incremental search adequately models interactive processing of database queries where the results of previous queries are usually irrelevant for processing the current one. In an expressive everyday life example, a new house is connected to the

electrical grid, water supply, road and telecommunication systems in a currently optimal way, although this solution could prove far from optimal after ten more houses are built in the neighbourhood. In computational geometry, a remarkably fast incremental Delaunay triangulation algorithm is based on the incremental NPS [20].

Details on the new algorithm and the problem itself are described in Section 2. Section 3 analyses the time complexity, compares the method with an older algorithm based on static strips, and introduces some additional improvements. Finally, the presented work is briefly summarized and some challenges for further research are discussed in Section 4.

2 The DP-DSL approach to the incremental nearest-point search

Subquadratic-time methods for the preliminary points arrangement approach are well-known. Utilization of the Voronoi diagram, together with efficient solutions of the point-location problem, for example, leads to an obvious $O(n \log n)$ time solution [1, 11], where n is the number of points in S. On the other hand, utilization of the Voronoi diagram in the incremental NPS requires some of the incremental Voronoi diagram construction algorithms which all, although fast on average, require quadratic time in the worst case [8]. For this reason and because of a relatively complex maintenance of the Voronoi diagrams, we preferably study other space partitioning techniques. First of all, we wish to keep practical advantages of the HT-DSL approach [19] and, simultaneously, to improve its theoretical behaviour. The pioneering HT-DSL approach represents even nowadays the only work where the incremental NPS is explicitly considered. It is based on a uniform plane subdivision into parallel strips. These static strips are directly accessible in $O(1)$ time through a *hash table* (*HT*). On the other hand, our method named the DP-DSL approach uses a dynamic partition (DP) into evenly populated strips. In both methods, the points in a particular strip are stored in (a, b)-*deterministic skip list* (*DSL*) [13], providing a point insertion in $O(\log n)$ time and, on the average, efficient NPS inside the strip. The DP-DSL approach must additionally provide the functionality of DSL splitting as an over-populated strip has to be split into two (or three) strips.

2.1 Deterministic skip lists

Our implementation of (a, b)-DSL, inherited from [19], consists of a doubly linked list of points sorted in non-descending order on the x-coordinate. If more points share the same x-coordinate, then the y-coordinate is decisive. Double connectivity assures that the move from an arbitrary point to its direct predecessor or successor takes $O(1)$ time. This list represents the basic level (level 1) of the DSL. Its nodes (*leaves*) are accessible from simply linked lists of the *internal nodes* at higher levels. Each parent node P (see Fig. 1) at level h, $h > 1$, points to a single child node C followed by the remaining children nodes of P, forming a *gap*. The first node C′ after the gap represents the leading (the one with the lowest x-coordinate) child node of the successive parent node (P′) of P. The gap size must be in range [a, b], except the gap behind the last child node. Access to a particular leaf requires $O(b \log n)$ worst time. By keeping b small, the logarithmic access time is provided. Typical pairs (a, b) in practice are (1, 2), (1, 3), (2, 5), and (3, 7). Fig. 1 shows an example of (1, 3)-DSL. Values stored in a gap are lower or equal to the value in the parent node. Consequently, M at the root level must be set to some "safely" high value.

The actual search for the nearest point to the query point p was also inherited from [19]. Once it determines a candidate of the nearest-point to p and its distance d to p, it limits the search for better candidates to the interior of the circle with the centre p and radius d. The search consists of the *local search* in the strip where p was inserted, and the *inter-strip search* which progresses up and/or down through the adjacent strips within the distance of current d. The *local search time* is the time needed to perform local search for a single query point, while the *total local search time* is the time spent for local search operations for all query points. In an analogous manner, the *inter-strip search time* and the *total inter-strip search time* can be introduced, while the *total search time* refers to the sum of both, the total local search time and the total inter-strip search time. Finally, the *total time* is the sum of the total search time and the time spent for the dynamic partition construction.

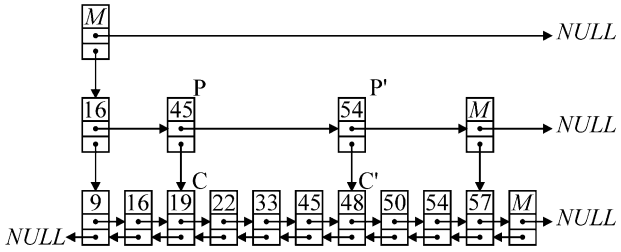

Figure 1: An example of (1, 3)-deterministic skip list.

2.2 Dynamic partition

The HT-DSL is remarkably fast for nearly uniform point distributions. However, examples with much slower performance and also strongly affected by the points ordering can effortlessly be constructed and, not rarely, also met in practice. Example in Fig. 2 consists of a cloud of points with rather favourable Gaussian distribution which alone does not result in highly over-populated strips. However, an isolated point concentrates all other points on the opposite end of the region of interest, resulting in a considerable number of empty strips and in the increased population density of those few strips containing the entire point cloud (Fig. 2a). The DP-DSL approach is directly designed to prevent from such situations. The idea is straightforward: when a particular strip contains too many points (the number of points in a strip is labelled q in continuation), the algorithm splits it into a pair of strips, each containing half of the points of the original strip. Under certain conditions, splitting may also result in three strips. The point cloud in Fig. 2b is cut by many narrow strips, while a wide undivided strip is left around the isolated point. Another, more realistic

demonstration of the advantageous behaviour of the DP-DSL approach is given in Fig. 3.

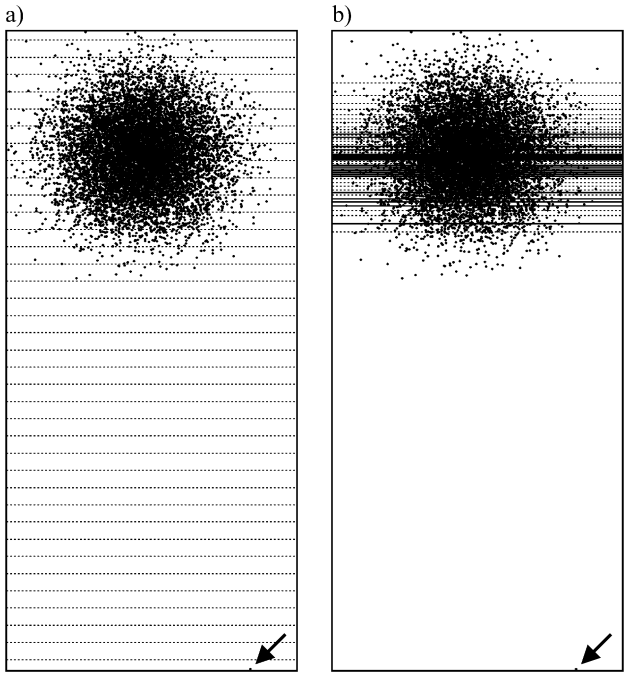

Figure 2: Gaussian distribution with additional point in: a) HT-DSL static partition with uniform strip width, and b) DP-DSL approach with evenly populated strips of variable widths.

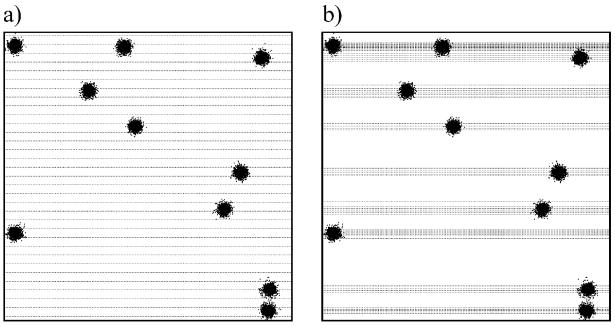

Figure 3: a) HT-DSL and b) DP-DSL approach employed on clusters of points.

The DP-DSL approach requires additional data structure to store the strips' borders i.e. minimum *y*-coordinates for each strip. We use an additional DSL named *Borders* for this purpose. It plays the same role as the hash table in the HT-DSL approach, but requires longer search time (logarithmic instead of constant) and dynamic construction.

Two types of strips are stored in *Borders*. A *line strip* SoP_i is the horizontal line $y = Borders_i$, and an *interval strip* SoP_j is a region between two horizontal lines. The lower borderline is also a part of the interval strip i.e. $SoP_j = \{p(x, y); Borders_j \leq y < Borders_{j+1}\}$. Exceptionally, the unbounded first and last strip are also the interval strips. Another exception is met when the interval strip SoP_j lies directly above a line strip SoP_{j-1}. Then the lower borderline cannot be a part of SoP_j since it represents an independent line strip. A line strip is introduced when the *y*-coordinates of two or more points correspond to the splitting threshold.

Points in each DSL are sorted according to *x*-coordinates, but an over-populated strip should be split with regard to *y*. Splitting is only sensible for the interval strips. A line strip may therefore have O(*n*) points, but the local search has to examine only the predecessor and successor of the query point while, on the other hand, the entire interval strip should be examined in the worst case. The role of the line strips is, therefore, to keep the sizes of the interval strips limited.

The splitting algorithm must firstly determine the splitting threshold. We utilize the well-known SELECT algorithm [5] which finds the *i*-th largest element in the set with *q* points in $\Theta(q)$ time. Simultaneously, the types of the output strips and the numbers of points in each of them are determined. Three diverse output situations can be met: (1) two interval strips, (2) two interval strips and the separating line strip, and (3) an interval strip above a line strip where the latter coincides with the bottom line of the input strip.

The strip splitting is completed by physically splitting the *DSL* into two or three separate skip lists. An intuitive solution rests on the $O(\log n)$-time skip list splitting algorithm [15] which cuts the input DSL at the determined splitting threshold into two separate DSLs. Of course three DSLs may be obtained, when necessary, by performing two cuts. The structure of the input DSL is mostly preserved in the separated DSLs, except that the gaps on the right side along the cut usually require some minor $O(b)$-time corrections. The values of *q* nodes are maintained in $\Theta(q)$ time afterwards. The method has several desired properties, including the aforementioned inherited structure of higher levels and the ability to reuse allocated nodes of the input DSL. However, the inherited gaps in the output DSLs, varying in size from *a* to *b*, are often too short for optimal further exploitation. Furthermore, additional short gaps on both sides along a cut are typically produced. In continuation, we propose an original approach, which gives full control over the gap size to the user.

The bricklaying approach firstly constructs level 1 for each of the two or three separate DSLs. This is achieved by moving the leaves of the input DSL, one after another, to the end of the corresponding separate list. Upper levels are then built from the elements of the simply linked *global list of recyclable nodes*. At each level, the nodes are grouped into gaps of size $b - gsc$, where *gsc* is a user-selected gap size correction parameter. A compromise must be found since shorter gaps accelerate later insertions (less gap rearrangements needed), while longer gaps reduce the numbers of nodes at higher levels and consequently decrease the search times.

The global list of recyclable nodes may contain nodes from three different sources, as shown in Fig. 4a. Firstly, eventual unused nodes from previous splitting operations are included. Although the output lists usually contain more internal nodes in total than the input DSL did, the opposite is also possible because of typically longer gaps in split DSLs. The second source consists of the input DSL's internal nodes. They are firstly organized into the

linked list, such that the last element of level k is connected to the first one at level k - 1, and this is then appended to the end of the global list. Finally, the third part contains eventual additional nodes allocated just before the actual splitting operation starts. Fig. 4b shows how the nodes of the global list are distributed across three output DSLs. The situation with two output strips is handled in nearly the same manner.

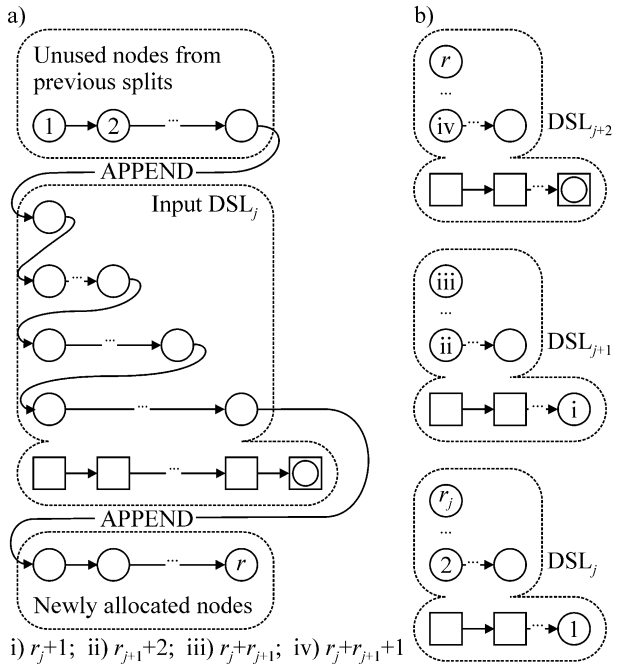

Figure 4: Bricklaying DSL splitting approach: a) global list of recyclable nodes, and b) three output DSLs.

3 Results and analysis

The number of strips in the HT-DSL approach was experimentally set in the range $m = \Theta(\sqrt{n})$. Consequently, the number of points in a strip is $q = O(\sqrt{n})$ in an optimal uniform point distribution. We have retained this result in the DP-DSL approach, and experimentally determined the best performance by splitting the strips reaching $q = \lceil 3\sqrt{n} \rceil$ points. We use (1, 3)-DSLs in the HT-DSL approach, and (2, 5)-DSLs in the DP-DSL approach. The best long-term performance was achieved by using the gap size correction parameter gsc = 1.

In Table 1, expected time complexities for handling the considered representative cases by the HT-DSL and DP-DSL approach are given. We consider Gaussian distribution of points with an additional isolated point (Fig. 2), few non-uniformly distributed clusters (Fig. 3), uniform distribution (Fig. 5a), grid (Fig. 5b), two GIS datasets (Figs. 5c and 5d), and the so-called *ladder* with an additional isolated point (Fig. 6b). The time ratios in the second column were obtained for configurations of 5.000.000 points. Much lower cardinalities were used in figures (except Fig. 5) to maintain clarity. The realistic examples in Figs. 5c-d consist of 70.334 and 193.360 points, respectively.

By choosing the number of strips in the range $m = \Theta(\sqrt{n})$, the average horizontal distance between two successive points in the DSL of an interval strip may be considered similar to the average strip width. Consequently, the local search mostly examines only a few nearest-point candidates, while the inter-strip search also traverses only a few strips. Both numbers may be considered O(1) and thus, the expected total search time for n query points does not exceed $O(n)$. The expected total times $O(n \log n)$ for the first six examples in Table 1 are therefore determined by the construction phase (see Table 2). Exceptionally, the HT-DSL approach in the first two examples (Figs. 2 and 3) collects n - 1 points in a single strip, and the local search time can be hardly considered $O(1)$. Significantly slower performance can be noticed in comparison to the DP-DSL approach, although the theoretical worst-case time complexity $\Theta(n^2)$ is not reached in this two cases.

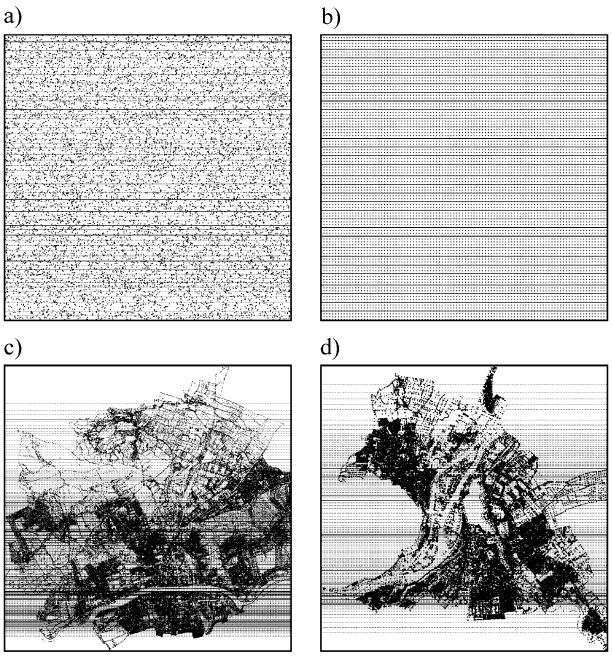

Figure 5: Dynamic partition into strips for: a) uniform points distribution, b) grid, and c-d) two GIS datasets.

While the order of inserting the points was considered random in the above six examples, the ladder illustrated in Fig. 6 and analysed in the last two rows of Table 1 was synthetically generated and represents the worst-case for the local search, which requires $\Theta(r)$ time if there are $2r$ points in an interval strip (see Fig. 6a). The condition $w < h$ assures that the nearest point of any $p_{r+i} \in l_2$ is exactly the other end of the same ladder rung i.e. $p_i \in l_1$. Another requirement $w > x_r - x_1$ provides the arrangement $x_j < x_{j+1}$ for every $j < 2r$. Consequently, exactly r points p_{r+i-1}, p_{r+i-2}, …, p_i have to be examined for every query point p_{r+i} on l_2, $0 < i \leq r$. Thus, the HT-DSL approach requires $\Theta(n^2)$ total local search time to handle the case in Fig. 6b with $2r = n - 1$ points. The DP-DSL approach handles the same case in $\Theta(n\sqrt{n})$ time in a similar manner as the example from Fig. 6c is handled.

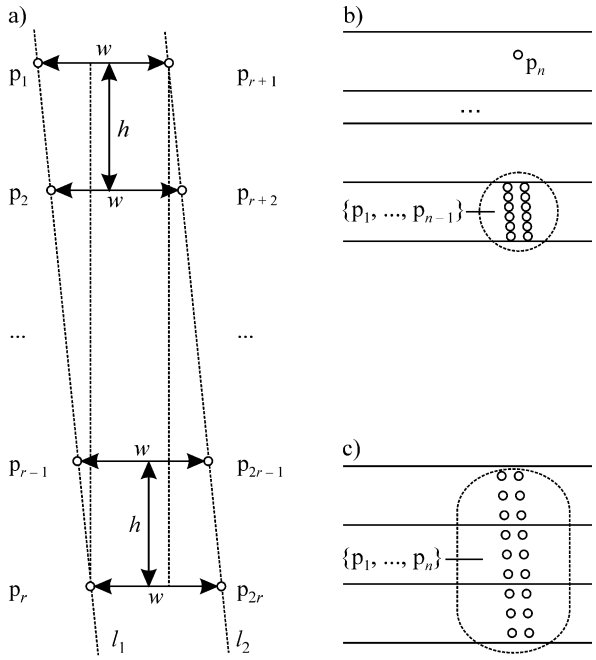

Figure 6: A ladder: a) point organization, b) HT-DSL case with $\Theta(n^2)$ total local search time, and c) case requiring $O(n\sqrt{n})$ time in both approaches.

Fig.	HT-DSL/DP-DSL	HT-DSL time	DP-DSL time
2	9.91	$\Theta(n \log n)$ to $O(n^2)$	$\Theta(n \log n)$
3	7.21	$\Theta(n \log n)$ to $O(n^2)$	$\Theta(n \log n)$
5a	0.79	$\Theta(n \log n)$	$\Theta(n \log n)$
5b	0.84	$\Theta(n \log n)$	$\Theta(n \log n)$
5c	0.50	$\Theta(n \log n)$	$\Theta(n \log n)$
5d	0.47	$\Theta(n \log n)$	$\Theta(n \log n)$
6b	203.73	$\Theta(n^2)$	$\Theta(n\sqrt{n})$
6c	0.59	$\Theta(n\sqrt{n})$	$\Theta(n\sqrt{n})$

Table 1: Expected time complexities of HT-DSL and DP-DSL approaches in considered representative examples.

3.1 Theoretical time complexity analysis

Table 2 lists theoretical worst-case time complexities of all phases of both approaches. The construction of strips and maintenance of DSLs are optimal in both cases, while the total local and total inter-strip search time do not provide the desired $O(n \log n)$ total time and require further consideration.

Besides other interesting cases, we have managed to construct one which requires $\Theta(n^2)$ inter-strip search time in the DP-DSL approach. In Fig. 7, points $p_1, \ldots, p_r$ are placed on slightly descending vectors in strips SoP_2 to SoP_m, and $p_{r+1}, \ldots, p_n$ are arranged from left to right in SoP_1. These latter (p_i, $i > r$) further fulfil the following conditions.

1. In each SoP_j, $j > 1$, distances to p_i descend from left to right along the bold line segment (the nearest point to p_i is the rightmost and the bottommost one).
2. $p_u \in SoP_{j+1}$, $p_v \in SoP_j$, $j > 1 \Rightarrow |p_i p_u| < |p_i p_v|$ i.e. upper bold segments are closer to p_i than lower ones.
3. Point p_{i+1} is closer to any point in S_{r+1} than to p_i.

We use auxiliary pairs of circular arcs to graphically emphasize the above conditions 1 and 2. The left and right arc in each pair are centred in p_n and p_{r+1}, respectively. The third condition is fulfilled by halving the horizontal distance between the query point and the farthest target point candidate p_{FAR} from S_{r+1} in each iteration. For each point from SoP_1, all r points from S_{r+1} have to be examined until the nearest point p_{NEAR} is found, thus the total time is raised for $\Theta((n - r)r)$. Selection $r \approx n/2$ obviously leads to $\Theta(n^2)$ time. Note that SoP_1 represents a line strip in the DP-DSL approach and thus it is allowed to contain $O(n)$ points. Anyway, x-coordinates of $p_{r+1}, \ldots, p_{n-1}$ in the considered example represent a geometric progression with ratio 2. Even with relatively low n and really small d, the exponential growth quickly produces x-coordinates out of the range of the IEEE 754 floating-point specification. If we choose $d = 1$, $n = 1000$, $r = 500$ and $x_{501} = 0$, for example, then we get $x_{1000} = 2^{499} \approx 1.6*10^{150}$, which is usually far beyond the expected range in industrial, GIS and other practical applications.

Phase	HT-DSL	DP-DSL
Construction		
Strip identification	$O(n)$	$O(n \log n)$
Point insertion	$\Theta(n \log n)$	$\Theta(n \log n)$
DSL splitting	0	$O(n)$
Maintenance of *Borders*	0	$O(\sqrt{n} \log n)$
Quering		
Local search	$\Theta(n^2)$	$\Theta(n\sqrt{n})$
Inter-strip search	$\Theta(n^2)$	$\Theta(n^2)$

Table 2: Worst-case time complexities of particular phases in the HT-DSL and DP-DSL approach.

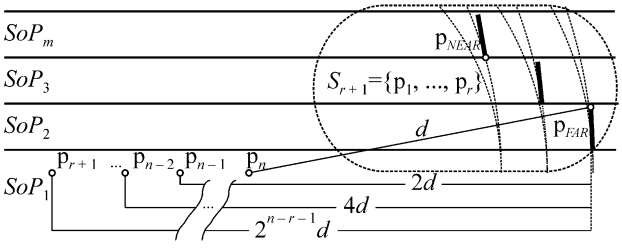

Figure 7: Construction of the $\Theta(n^2)$ time example based on geometric progression.

Note that the total local search time of the DP-DSL approach could be improved from $O(n\sqrt{n})$ to $O(n \log n)$ by splitting the strips of size $q = O(\log n)$ instead of current $q = \lceil 3\sqrt{n} \rceil$. Although this change does not critically increase theoretical worst-case time complexities of other phases, it usually results in slower practical performance due to the increased number of DSL splits and initial positioning operations in much more DSLs during the inter-strip search.

3.2 DP-4DSLs approach

We have recently developed an engineering solution which handles the considered problematic examples in the desired (optimal) time bounds. It additionally performs the vertical DP. In each strip, two orthogonal DSLs are constructed, the horizontal one sorted by the *x*-coordinate, and the vertical one sorted by the *y*-coordinate. Each point is therefore placed into four DSLs: XH-DSL (also in the DP-DSL approach) and YH-DSL are assigned to each horizontal strip, and XV-DSL and YV-DSL are constructed in each vertical strip (see Fig. 8). Note that the strip splitting threshold can be found by help of the orthogonal DSL in a quicker way than with the aforementioned SELECT algorithm.

In each iteration of the local search, the method performs one move in each DSL which are all addressing the same radius *d*. The nearest point is found when the first DSL (the winner) examines all the points within the distance *d* around the query point. We have not managed to theoretically prove optimal time complexity but the performance in the considered cases (I to VI in Table 3) appears promising.

- The cases I and II were already considered in Table 1. In the DP-4DSLs approach, the nearest-point of any point p is its direct predecessor in the YH-DSL. The total local search and inter-strip times are both $\Theta(n)$ and thus the total time $\Theta(n \log n)$ is determined by the construction phase.
- In the ladder example rotated 90 degrees (case III), the local search in the HT-DSL and the DP-DSL approach examines at most two points, but the inter-strip search traverses $\Theta(\sqrt{n})$ strips for half of the query points, resulting in $\Theta(n\sqrt{n})$ total time. In the DP-4DSLs approach, XV-DSL has the same role as YH-DSL has in cases I and II, resulting in $\Theta(n \log n)$ total time.
- Case IV was addressed by the HT-DSL and DP-DSL approach in Section 3.1 already. In the DP-4DSLs approach, however, XV-DSL provides $O(1)$ local search and inter-strip search times and thus the total time $\Theta(n \log n)$ is determined by the construction phase.
- In case V where the configuration from case IV is rotated 90 degrees, YH-DSL has the same role as XV-DSL has in case IV, and optimal $\Theta(n \log n)$ is again achieved. For the HT-DSL and the DP-DSL approaches, the same conclusions can be made as in case II.
- "Regular" cases (VI) refer to those configurations, where an optimal $\Theta(n \log n)$ time complexity is expected (see Fig. 5 and Table 1) within both, HT-DSL and DP-DSL approach. Of course, the same optimal time complexity is expected by the DP-4DSLs approach because the winner can either be XH-DSL (also used in the HT-DSL and DP-DSL approaches) or some other DSL outperforming XH-DSL. Note that the HT-DSL and DP-DSL approaches usually outperform the DP-4DSLs approach in "regular" cases as maintenance of two partitions and four DSLs is quite expensive.
- Note that the remaining YV-DSL, which is not met in the considered examples, is also necessary. It is for example the winner if a "regular" case won by XH-DSL is rotated 90 degrees.

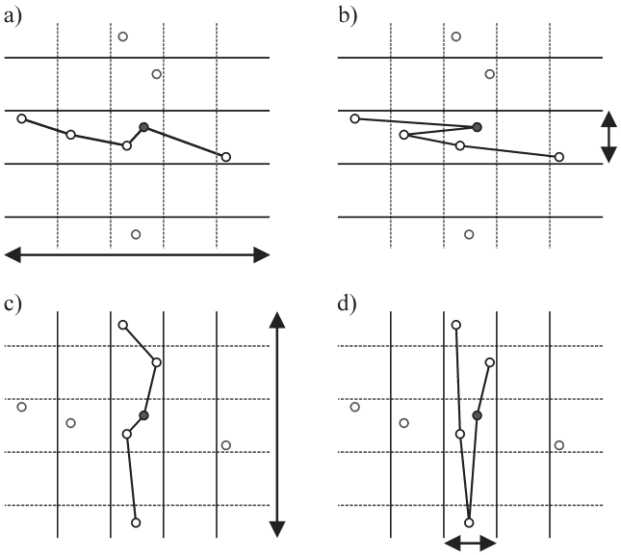

Figure 8: A query point (filled grey) is in four DSLs: a) XH-DSL, b) YH-DSL, c) YV-DSL, and d) XV-DSL.

Case	HT-DSL	DP-DSL	DP-4DSLs	Winner
I	$\Theta(n^2)$	$\Theta(n\sqrt{n})$	$\Theta(n \log n)$	YH
II	$\Theta(n\sqrt{n})$	$\Theta(n\sqrt{n})$	$\Theta(n \log n)$	YH
III	$\Theta(n\sqrt{n})$	$\Theta(n\sqrt{n})$	$\Theta(n \log n)$	XV
IV	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n \log n)$	XV
V	$\Theta(n\sqrt{n})$	$\Theta(n\sqrt{n})$	$\Theta(n \log n)$	YH
VI	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	various

Table 3: Comparison of the three approaches in the considered cases: I – ladder with an isolated point, II – ladder, III – ladder rotated 90 degrees, IV – geometric-progression-based case from Fig. 7, V – case from Fig. 7 rotated 90 degrees, and VI – "regular case".

4 Conclusion

The paper considers a new (DP-DSL) approach to the incremental nearest-point search in 2-D. It guarantees $\Theta(\sqrt{n})$ interval strips, each containing $\Theta(\sqrt{n})$ points and, therefore, successfully prevents situations with over-populated interval strips and decreases the total local search time from $O(n^2)$ to $O(n\sqrt{n})$. In our opinion, this is an important acceleration, although the algorithm still fails to achieve an optimal $O(n \log n)$ time performance characteristic for the preliminary points arrangement approach. In addition, examples can be constructed (although hardly met in practice) which, just as the "traditional" HT-DSL approach still achieve quadratic inter-strip search time. The DP-4DSLs variant seems to solve the considered problematic examples in optimal time, but a formal proof is still missing. Construction of the Voronoi diagram on $\Theta(\sqrt{n})$ points and utilization of

two perpendicular DSLs in each Voronoi cell could have a potential, but one should first prove that such dynamic partition is generally possible, and then provide an efficient region splitting algorithm.

Acknowledgement

This work was partially supported by the Slovenian Research Agency (research programme P2-0041).

References

[1] F. Aurenhammer F. (1991). Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Computing Surveys*, ACM, vol. 23, iss. 3, pp. 345-405.

[2] Chan T. M.; Rahmati Z. (2017). Approximating the minimum closest pair distance and nearest neighbor distances of linearly moving points. *Computational Geometry*, Elsevier Science, vol. 60, Jan. 2017, pp. 2-7. https://doi.org/10.1016/j.comgeo.2016.04.001

[3] Chaurasia V.; Chaurasia V. (2016). Statistical feature extraction based technique for fast fractal image compression. *Journal of Visual Communication and Image Representation*, Elsevier Science, vol. 41, Nov. 2016, pp. 87-95. https://doi.org/10.1016/j.jvcir.2016.09.008

[4] Cleary J. G. (1979). Analysis of an Algorithm for Finding Nearest Neighbors in Euclidean Space. *ACM Transactions on Mathematical Software*, ACM, vol. 5, iss. 2, pp. 183-192. https://doi.org/10.1145/355826.355832

[5] Cormen T. H.; Leiserson C. E.; Rivest R. L.; Stein C. (2001). *Introduction to Algorithms, 2nd Edition*, MIT Press and McGraw-Hill.

[6] de Gomensoro Malheiros M.; Walter M. (2016). Spatial sorting: an efficient strategy for approximate nearest neighbor searching. *Computers & Graphics*, Elsevier Science, vol. 57, June 2016, pp. 112-126. https://doi.org/10.1016/j.cag.2016.03.006

[7] Gómez-Ballester E.; Micó L.; Oncina J. (2006). Some approaches to improve tree-based nearest neighbour search algorithms. *Pattern Recognition*, Elsevier Science, vol. 39, iss. 2, pp. 171-179. https://doi.org/10.1016/j.patcog.2005.06.007

[8] Guibas L. J.; Knuth D. E.; Sharir M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, Springer, vol. 7, iss. 4, pp. 381-413. https://doi.org/10.1007/BF01758770

[9] Han Y.; Tang J.; Zhou Z.; Xiao M.; Sun L.; Wang Q. (2014). Novel itinerary-based KNN query algorithm leveraging grid division routing in wireless sensor networks of skewness distribution. *Personal and Ubiquitous Computing*, Springer, vol. 18, iss. 8, pp. 1989-2001. https://doi.org/10.1007/s00779-014-0795-y

[10] Kanda T.; Sugihara K. (2002). Comparison of various trees for nearest-point search with/without the Voronoi diagram. *Information Processing Letters*, Elsevier Science, vol. 84, iss. 1, pp. 17-22. https://doi.org/10.1016/S0020-0190(02)00221-1

[11] Kirkpatrick, D. G. (1983). Optimal search in planar subdivisions, *SIAM J. Comput.*, vol 12, iss. 1, pp. 28-35. https://doi.org/10.1137/0212002

[12] Long Y.; Zhu F.; Shao L. (2016). Recognising occluded multi-view actions using local nearest neighbour embedding. *Computer Vision and Image Understanding*, Elsevier Science, vol. 144, March 2016, pp. 36-45. https://doi.org/10.1016/j.cviu.2015.06.003

[13] Munro J. I.; Papadakis T.; Sedgewick R. (1992). Deterministic skip lists. *Proceedings of the Third ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, Orlando, USA, pp. 367-375.

[14] Nagasue J.; Konishi Y; Araki N.; Sato T.; Ishigaki H. (2009). Slope-Walking of a Biped Robot with K Nearest Neighbor Method. *Proceedings of the 2009 Fourth International Conference on Innovative Computing, Information and Control*, IEEE Computer Society, Kaohsiung, Taiwan, pp. 173-176. https://doi.org/10.1109/ICICIC.2009.333

[15] Pugh W. (1990). Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, ACM, vol. 33, iss. 6, pp. 668-676. https://doi.org/10.1145/78973.78977

[16] Savchenko A. V. (2017). Maximum-likelihood approximate nearest neighbor method in real-time image recognition. *Pattern Recognition*, Elsevier, vol. 61, Jan. 2017, pp. 459-469. https://doi.org/10.1016/j.patcog.2016.08.015

[17] Wang. H.; Cao J.; Shu L.; Rafiei D. (2013). Locality sensitive hashing revisited: filling the gap between theory and algorithm analysis. *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, ACM, San Francisco, USA, pp. 1969-1978. https://doi.org/10.1145/2505515.2505765

[18] Wei-Kleiner F. (2016). Tree decomposition-based indexing for efficient shortest path and nearest neighbors query answering on graphs. *Journal of Computer and System Sciences*, Elsevier Science, vol. 82, iss. 1, part A, pp. 23–44. https://doi.org/10.1016/j.jcss.2015.06.008

[19] Zadravec M.; Brodnik A.; Mannila M.; Wanne M.; Žalik B. (2008). A practical approach to the 2D incremental nearest-point problem suitable for different point distributions. *Pattern Recognition*, Elsevier Science, vol. 41, iss. 2, pp. 646-653. https://doi.org/10.1016/j.patcog.2007.06.031

[20] Zadravec M.; Žalik B. (2005). An almost distribution-independent incremental Delaunay triangulation algorithm. *Visual Computer*, Springer, vol. 21, iss. 6, pp. 384-396. https://doi.org/10.1007/s00371-005-0293-3

[21] Zheng R. Y. (2010). *Machine Learning Approaches to Bioinformatics*, World Scientific Publishing Co., Inc. https://doi.org/10.1142/7454