A New Algorithm for the Closest Pair of Points for Very Large Data Sets using Exponent Bucketing and Windowing ⋆

Vaclav Skala[1] [0000−0001−8886−4281],
⋆⋆ Alejandro Esteban Martinez[1,2], David Esteban Martinez[1,2], and Fabio Hernandez Moreno[1,3]

[1] Department of Computer Science and Engineering
University of West Bohemia, Czech Republic
skala@kiv.zcu.cz www.VaclavSkala.eu
[2] University of A Coruña, Spain
[3] University of Las Palmas de Gran Canaria, Spain

Abstract. In this contribution, a simple and efficient algorithm for the closest-pair problem in E^1 is described using the preprocessing based on exponent bucketing and exponent windowing respecting accuracy of the floating point representation. The preprocessing is of the $O(N)$ complexity. Experiments made for the uniform distribution proved significant speedup. The proposed approach is applicable for the E^2 case.

Keywords: Closest pair · minimum distance · uniform distribution · data structure · bucketing sort · computational geometry · large data sets · data windowing

1 Introduction

The closest pair problem is a problem of finding two points having minimum mutual distance in the given data set. Brute force algorithms with $O(N^2)$ complexity are used for a small number of points, and for higher number of points algorithms based on sorting have $O(N \lg N)$ complexity. In the case of large data sets[4], the processing time with $O(N \lg N)$ complexity might be prohibitive.

In this contribution, a proposed simple preprocessing of the data set based on the bucketing and exponent windowing is described; similar strategy as in Skala[13, 12, 14, 15] and Smolik[16]. The extension to the E^2 case is straightforward using already developed algorithms. The closest pair problem was address in Shamos[11], Kuhller [7], Golin[4] and Mavrommatis[8], Pereira[9], Roumelis[10] used space subdivision and Bespamyatnikh[1] used a tree representation, Daescu[3,

⋆ Supported by the University of West Bohemia - Institutional research support
⋆⋆ These students contributed during their Erasmus ACG course at the University of West Bohemia
[4] Data set, where $N \ggg 10^6$. Note that 2 147 483 648 $\doteq$ 2.147 10^9 unsigned distinct values only can be represented in single precision.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

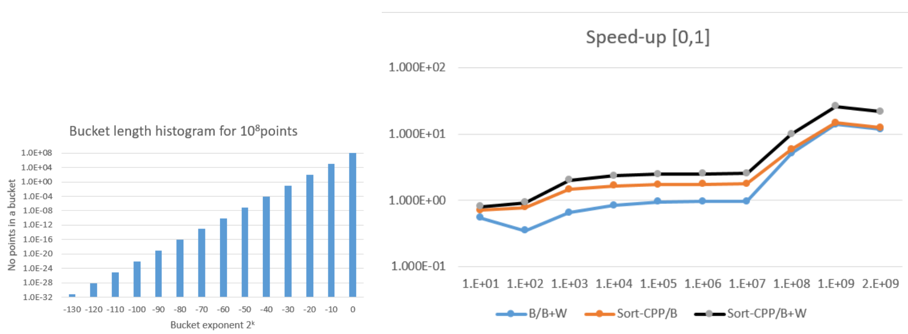
2] and Katajainen[6] used divide & conquer strategy, Kamousi[5] used a stochastic approach.

In the following, basic strategies for finding a minimum distance of points in E^1 are described.

Brute-force algorithm - The naive approach leads to a brute-force algorithm, which is simple and easy to implement. However, it has $O(N^2)$ computational complexity as it requires $N(N-1)/2$ computational steps. This algorithm cannot be used even for relatively small N due to the algorithm complexity, but can be also used for a higher dimensional case. The algorithm can be speed-up a bit, as d can be computed as $d := \|\mathbf{x}[i] - \mathbf{x}[j]\|^2$ and d_0 set as $d_0 := \sqrt{d_0}$ as the function $f(x) = x^2$ is a monotonically growing function[5].

Algorithm with sorting - In the one dimensional case, i.e. the E^1 case, the given values x_i might be re-ordered into the ascending order. It leads to $O(N \log N)$ computational complexity. Then the ordered data are searched for the minimum distance of two consecutive numbers, i.e. $x_{i+1} - x_i$ as $x_{i+1} \geq x_i \ \forall i$, with $O(N)$ complexity. The above-mentioned algorithms are correct and can be used for computation with "unlimited precision".

However, in real implementations, the used floating point representation has a limited mantissa representation and range of exponents.



Bucket length histogram (a) | Speed-up for the interval $[0, 1]$ (b)

Table 1. Bucket length distribution and speed-up of the proposed algorithm

Algorithm with limited mantissa - The computational complexity of the algorithm is $O(N \log N)$ due to the ordering and finding the minimum distance is $O(N)$ complexity as all N values have to be tested, as the smallest difference might be given by the last binary digits in the mantissa.[6]

[5] It actually saves $O(N^2)$ computations of the $\sqrt{*}$ function.

[6] Let us consider a sorted sequence $1.01, 1.05, \ldots, 10.0001, 10.0005$, then the minimum distance is 0.0004 not 0.04.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

However, in the real data cases, the expected complexity will be smaller, due to values distribution over several binary exponents. Tab.1.a presents[7] a histogram of values according to their binary exponent; the uniform distribution $[0, 1]$ is used.
In reality, the IEEE 754 floating point representation or a similar one with a limited mantissa precision is used. It means, that if $d = |x_{i+1} - x_i|$ is the currently found minimum distance, i.e. $d = [m_d, E_d]$, where m_d is the mantissa and E_d is the binary exponent of d, then the stopping criterion for searching the ordered $\mathbf{x}$ values is $x_i > d_0 * 2^{p+1}$, where p is the number of bits of the mantissa, d_0 is already found minimum.

It can be seen, that the limited mantissa precision reduces the computational requirements significantly for the larger range of the binary exponents of values. Tab.1.a presents an expected number of points having exponents within exponent buckets for 10^8 of points, if the uniform distribution is used.

2 Proposed algorithm with $O_{exp}(N)$ complexity

The bottleneck of the algorithm standard $O(N \log N)$ complexity is the ordering step. However, instead of "standard" sorting algorithms, e.g. heap sort, shall sort, quick sort etc., it is possible to use bucketing by the exponent values[8] in stead, which has $O(N)$ complexity, see Fig.1. The data structure is similar to the standard hashing structure. All values having the same binary exponent are stored in an array-list or in a similar data structure[9], see Fig.1. The values E_{min}

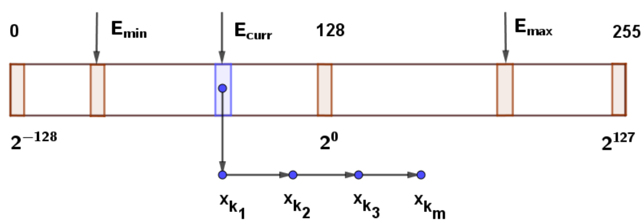


Fig. 1. Bucketing structure - 32-bits

and E_{max} are the *minimum* and *maximum* binary exponents found. It means, that all values are sorted according to their binary exponents, but *unsorted* within the actual bucket, i.e. unordered, if the exponents are equal. The table length is 256, resp. 2048 according to the precision used, i.e. 32-bits, resp. 64-bits.

It should be noted, that even for 10^8 points, the probability for small values of exponents is extremely low. As the mantissa precision is limited the bucket length for very low exponents will be zero or very small.

[7] Note: 130-120 means exponents interval [-120,...,-111]
[8] the binary exponent is shifted, i.2. a value 2^{-128} has the shifted exponent 0.
[9] The array list, i.e. extensible arrays were used in the actual implementation.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

Algorithm 1 Minimum distance with bucketing in E^1

```
procedure MinDistBucket(x, N, d_0);
                                   ▷ given set of N points x, x[i] ≥ 0, distance d_0 found
    E_min := maxint; E_max := minint;                                   ▷ initial setting
    p := 24;                                       ▷ 32-bits: p = 24; 64-bits: p := 53
    E_range := 256;            ▷ exponent range E_range := 2048, if double precision
    # preprocessing - buckets construction #
    for i := 1 to N do
        Ex := Exponent(x[i]);                          ▷ binary exponent [0, E_range]
        Add(x[i],Bucket(Ex));             ▷ add the value to the bucket Bucket(Ex)
        E_min := min {E_min, Ex};
    end for                                             ▷ all bucket are constructed

    d_0 := ∞ ;                                    ▷ setting a min. distance estimation
    temp := −∞;                                                    ▷ initial setting
    i := E_min;                                                  ▷ Bucket[E_min] ≠ ∅
    E_max := E_min + p;                      ▷ E_max - upper bound for the windowing
    InWindow := true;
    PairFound := false;
                         ▷ case if the only one point is inside of the exponent window
    while (i ≤ E_max or not PairFound) and i ≤ E_range do
        if (Bucket[i] ≠ ∅) and InWindow then
            SORT_Bucket(i);                       ▷ sorts values in the i-th Bucket
            # [d, temp]:=ProcessOneBucket(i,temp); #
                  ▷ finds a minimum distance d of temp and values in the Bucket[i]
                                     ▷ temp is last value in the Bucket[i − 1]
            # find a minimum distance in a {temp, Bucket[∗]}, if exists #
            for k := 1 to Bucket.length[i] do
                xx := Bucket[i][k];                           ▷ get the current value
                d := xx − temp; temp := xx;
                if d_0 > d then
                    d_0 := d;
                    PairFound := true;                ▷ at least one valid pair found
                end if
            end for
            Ex := Exponent(d_0);                          ▷ Windowing the exponent
            InWindow := Ex + p < i;
                ▷ STOP, if the exponent Ex of (d_0 + p) ≥ i; the current exponent i
                                                     ▷ p is the mantissa length+1
        end if
    end while
end procedure
```

#**SOLVED** - A sequence 10^{23}, 0.1 10^{0}, 10.001 10^{23} is handled properly

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

Taking above into consideration, the proposed algorithm based on bucketing and exponent windowing is given by the algorithm Alg.1, where sorting of buckets is made on a request, i.e. when needed.

The function ProcessOneBucket(i, *temp*) finds a minimum distance within the **sorted** Bucket[i], taking *temp* value as the element before the first element in the Bucket[i]. It should be noted, that there is a "window" in the exponent table long 24 in the case of 32-bits, resp. 53 in the case of 64-bits, in which data are to be processed due to the mantissa limited precision. It leads to significant speed-up, especially for large data interval range.

3 Algorithm analysis

Let us consider uniform distribution on the interval $[0,1]$, e.g. using the standard $random(*)$ function. The exponent bucketing is a non-linear space subdivision as the space of values is split non-linearly, i.e. the interval length grows exponentially. In this case, values have the *power distribution* 2^k, $k=-128,\ldots,-1$, or $k=-128,\ldots,127$, if data generated from the interval $[0,\infty)$. It means, that for small exponent values, fewer elements are stored in a bucket, while for higher exponent values more values are stored in the relevant bucket.

As all values are generated within the interval $[0,1]$ and the 32-bits precision is used, then each sub-interval is of the length 2^k. It means that if N points are generated uniformly within the interval [0,1], then the interval k contains m_k values, see Eq.1, where $m_k = 2^k N$, $k=-128,\ldots,-1$ as:

$$\sum_{k=1}^{k=128} 2^{-k} = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^{128}} \doteq 1 \tag{1}$$

As the distance between small values is smaller, there is higher probability, that the minimum distance, i.e. the closest pair will be found faster, see Tab.1.a.

However, not the whole range of exponents will be used in a real situation and the interval of exponents $[E_{min}, E_{max}]$ can be expected. As the precision of the mantissa is limited to 24 bits in the 32-bits case, a window of 24 exponents is to be processed instead of 128, resp. 256; similarly in the case of 64-bits, the window of 53 exponents is to be processed instead of 1024, resp. 2048.
It should be noted that for $N=10^{10}$ values, the number of values having the exponent 2^{-128}, i.e. for $\xi=0$, is $2.9\ 10^{-29}$ only. The lowest non-empty bucket will probably have the non-shifted exponent 33 and the shifted exponent $Ex=95$ [10]. It means that the expected E_{max} will be $E_{max}=117$, i.e. $k=-33+24=-9$, if the single precision used.

Therefore, in the case of the $[0,1]$ uniform distribution interval, the last 8 shifted (physical) exponents, i.e. $Ex=[120,127]$, will not be evaluated. Those

[10] The value $k \doteq 33$ is obtained by solving $10^{10} * 2^k = 1$, which is $k \doteq -33$ and the shifted exponent $Ex = 128 - 33 = 95$.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

buckets contain N_0 values, i.e. approx. 99.609% points, see Eq.2.

$$N_0 = N \sum_{i=1}^{8} \frac{1}{2^i} \qquad \frac{N_0}{N} = \sum_{i=1}^{8} \frac{1}{2^i} \doteq 99.609375 \tag{2}$$

where i is the non-shifted exponent. Therefore, efficiency of the proposed algorithm grows with the exponent range in the uniform distribution case. The proposed algorithm can be modified for the Gaussian distribution easily.

4 Experimental results

The implementation of the algorithm described in Alg.1 is based on some data profile assumptions. There are two significant factors to be considered:

- the range of data exponents should be higher; if all the data would have the same binary exponent, only one very long bucket would be created,
- the algorithm is intended for larger data sets, i.e. number of points $N > 10^6$.

In the actual implementation an equivalent of the **array-list** was used, which is extended to a double length if needed and data are copied to the new position[11]. It might lead the *copy-paste* extensive use resulting to slow-down. In the case of the uniform distribution, the initial length of a bucket should be set to a recommended length $N\ 2^{-k} * 1.2$ setting used in the experimental evaluation, i.e. $length.Bucket[k] \geq N\ 2^{-k} * 1.2$.

Evaluation Uniform distribution of values was used with different intervals from $[0, 1]$ to $[0, 10^4]$. Up to $2\ 10^9$ points were generated and efficiency of the proposed algorithm was tested.

Obtained results for the interval $[0, 1]$ are summarized in Tab.1.b, where ratios of time spent are presented. Notation: $Sort - CPP/B = \frac{time_{Sort-CPP}}{time_{Bucketing}}$,
$Sort - CPP/B + W = \frac{time_{Sort-CPP}}{time_{Bucketing+Windowing}}$, and
$B/B + W = \frac{time_{Bucketing}}{time_{Bucketing+Windowing}}$
The ratio $Sort - CPP/B$ gives reached speed-up using the exponent bucketing over sort[12] used in C++. It can be seen that there is a speed-up over 1.2 for more than 10^3 points and grows with the range of generated data.

The ratio $Sort - CPP/B + W$ gives reached speed-up using the exponent bucketing over sort used in C++ and the speed-up is over 1.4.

[11] In the case of 10^6 values, over 10^3 bucket extensions were called, but with the 20% additional memory allocation, the extension was called only 7 times.

[12] Sort-CPP - standard Shell sort in C++

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

It should be noted, that

- speed-up for 10^9 points is over 16 times against if the sort method is used.
- ratio $B/B+W$ clearly shows significant influence of the windowing, which reflects the limited precision of numerical representation.

Notes - As the number of the processed values N is high, there are possible modifications of the algorithm leading to further efficiency improvements, e.g.:

- finding the E_{min} and E_{max} can be done after the buckets construction; it saves $O(N)$ floating point comparisons, or it can be removed with initial setting $E_{min} := 0$,
- some heuristic strategies can be used, e.g. pick up m values, find the smallest and its exponent Ex, and the second smallest one and determine a first minimum distance estimation. Set E_{max}:= $E_x + p$ (E_x is the exponent of already minimum found) as a stopping criterion for building buckets (it eliminated long bucket constructions for higher exponents, which cannot contribute to the minimum distance[13]).

5 Conclusions

In this contribution, an efficient improvement of the minimum distance algorithm E^1 case is presented. It takes a limited precision of the floating point representation into consideration and uses bucketing sort based on exponent's baskets. The presented approach is intended for larger data sets with a higher exponent range. Experiments made proved a significant speed-up of the proposed approach for the uniform distribution. The proposed approach can be extended to the E^2 case using algorithms as proposed in Daescu[3, 2], Golin[4], etc. Extensions of the proposed approach for the E^2 and E^3 cases are future work.

Acknowledgments

The author would like to thank colleagues at the University of West Bohemia, Plzen for their comments and suggestions, comments and hints provided, especially to Martin Cervenka and Lukas Rypl for some additional additional counter tests. Thanks also belong to anonymous reviewers for their critical view and recommendations that helped to improve this manuscript.

Responsibilities

Skala,V.: theoretical part, algorithm design, algorithm implementation and verification, manuscript preparation; Esteban Martinez,A., Esteban Martinez,D., Hernandez Moreno,F.: algorithm implementation and experimental verification.

[13] Note, that p depends on the FP precision used.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40

References

1. Bespamyatnikh, S.: An optimal algorithm for closest-pair maintenance. Discrete and Computational Geometry **19**(2), 175–195 (1998). https://doi.org/10.1007/PL00009340
2. Daescu, O., Teo, K.: 2D closest pair problem: A closer look. CCCG 2017 - 29th Canadian Conf. on Computational Geometry, Proceedings pp. 185–190 (2017)
3. Daescu, O., Teo, K.: Two-dimensional closest pair problem: A closer look. Discrete Applied Mathematics **287**, 85–96 (2020). https://doi.org/10.1016/j.dam.2020.08.006
4. Golin, M.: Randomized data structures for the dynamic closest-pair problem. SIAM Journal on Computing **27**(4), 1036–1072 (1998). https://doi.org/10.1137/S0097539794277718
5. Kamousi, P., Chan, T., Suri, S.: Closest pair and the post office problem for stochastic points. Computational Geometry: Theory and Applications **47**(2 PART A), 214–223 (2014). https://doi.org/10.1016/j.comgeo.2012.10.010
6. Katajainen, J., Koppinen, M., Leipälä, T., Nevalainen, O.: Divide and conquer for the closest-pair problem revisited. International Journal of Computer Mathematics **27**(3-4), 121–132 (1989). https://doi.org/10.1080/00207168908803714
7. Khuller, S., Matias, Y.: A simple randomized sieve algorithm for the closest-pair problem. Information and Computation **118**(1), 34–37 (1995). https://doi.org/10.1006/inco.1995.1049
8. Mavrommatis, G., Moutafis, P., Corral, A.: Enhancing the slicenbound algorithm for the closest-pairs query with binary space partitioning. ACM International Conference Proceeding Series pp. 107–112 (2021). https://doi.org/10.1145/3503823.3503844
9. Pereira, J., Lobo, F.: An optimized divide-and-conquer algorithm for the closest-pair problem in the planar case. Journal of Computer Science and Technology **27**(4), 891–896 (2012). https://doi.org/10.1007/s11390-012-1272-6
10. Roumelis, G., Vassilakopoulos, M., Corral, A., Manolopoulos, Y.: A new plane-sweep algorithm for the k-closest-pairs query. Lecture Notes in Computer Science **8327 LNCS**, 478–490 (2014). https://doi.org/10.1007/978-3-319-04298-5_42
11. Shamos, M., Hoey, D.: Closest-point problems. Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS **1975-October**, 151–162 (1975). https://doi.org/10.1109/SFCS.1975.8
12. Skala, V.: Fast $oexpected(n)$ algorithm for finding exact maximum distance in E2 instead of $o(n^2)$ or $o(nlgN)$. AIP Conference Proceedings **1558**, 2496–2499 (2013). https://doi.org/10.1063/1.4826047
13. Skala, V., Cerny, M., Saleh, J.: Simple and efficient acceleration of the smallest enclosing ball for large data sets in e2: Analysis and comparative results. LNCS **13350**, 720–733 (2022). https://doi.org/10.1007/978-3-031-08751-6_52
14. Skala, V., Majdisova, Z.: Fast algorithm for finding maximum distance with space subdivision in E2. LNCS **9218**, 261–274 (2015). https://doi.org/10.1007/978-3-319-21963-9_24
15. Skala, V., Smolik, M.: Simple and fast $oexp(n)$ algorithm for finding an exact maximum distance in E2 instead of $o(n^2)$ or $o(n \lg N)$. LNCS **11619**, 367–380 (2019). https://doi.org/10.1007/978-3-030-24289-3_27
16. Smolik, M., Skala, V.: Efficient speed-up of the smallest enclosing circle algorithm. Informatica **33**(3), 623–633 (2022). https://doi.org/10.15388/22-INFOR477

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_40