Dummy title

Jane Open Access
Dummy University Computing Laboratory, [optional: Address], Country
My second affiliation, Country

Joan R. Public[1]
Department of Informatics, Dummy College, [optional: Address], Country

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Jane Open Access*: (Optional) author-specific funding acknowledgements
Joan R. Public: [funding]

Acknowledgements I want to thank ...

1 Typesetting instructions – Summary

LIPIcs is a series of open access high-quality conference proceedings across all fields in informatics established in cooperation with Schloss Dagstuhl. In order to do justice to the high scientific quality of the conferences that publish their proceedings in the LIPIcs series, which is ensured by the thorough review process of the respective events, we believe that LIPIcs proceedings must have an attractive and consistent layout matching the standard of the series. Moreover, the quality of the metadata, the typesetting and the layout must also meet the requirements of other external parties such as indexing service, DOI registry, funding agencies, among others. The guidelines contained in this document serve as the baseline for the authors, editors, and the publisher to create documents that meet as many different requirements as possible.

Please comply with the following instructions when preparing your article for a LIPIcs proceedings volume.

Minimum requirements

- Use pdflatex and an up-to-date LaTeX system.
- Use further LaTeX packages and custom made macros carefully and only if required.
- Use the provided sectioning macros: `\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\paragraph*`, and `\subparagraph*`.
- Provide suitable graphics of at least 300dpi (preferably in PDF format).
- Use BibTeX and keep the standard style (`plainurl`) for the bibliography.
- Please try to keep the warnings log as small as possible. Avoid overfull `\hboxes` and any kind of warnings/errors with the referenced BibTeX entries.
- Use a spellchecker to correct typos.

---

[1] Optional footnote, e.g. to mark corresponding author

Mandatory metadata macros

Please set the values of the metadata macros carefully since the information parsed from these macros will be passed to publication servers, catalogues and search engines. Avoid placing macros inside the metadata macros. The following metadata macros/environments are mandatory:

- `\title` and, in case of long titles, `\titlerunning`.
- `\author`, one for each author, even if two or more authors have the same affiliation.
- `\authorrunning` and `\Copyright` (concatenated author names)
  The `\author` macros and the `\Copyright` macro should contain full author names (especially with regard to the first name), while `\authorrunning` should contain abbreviated first names.
- `\ccsdesc` (ACM classification, see `https://www.acm.org/publications/class-2012`).
- `\keywords` (a comma-separated list of keywords).
- `\relatedversion` (if there is a related version, typically the "full version"); please make sure to provide a persistent URL, e. g., at arXiv.
- `\begin{abstract}...\end{abstract}` .

Please do not ...

Generally speaking, please do not override the `lipics-v2021`-style defaults. To be more specific, a short checklist also used by Dagstuhl Publishing during the final typesetting is given below. In case of **non-compliance** with these rules Dagstuhl Publishing will remove the corresponding parts of LaTeX code and **replace it with the `lipics-v2021` defaults**. In serious cases, we may reject the LaTeX-source and expect the corresponding author to revise the relevant parts.

- Do not use a different main font. (For example, the `times` package is forbidden.)
- Do not alter the spacing of the `lipics-v2021.cls` style file.
- Do not use `enumitem` and `paralist`. (The `enumerate` package is preloaded, so you can use `\begin{enumerate}[(a)]` or the like.)
- Do not use "self-made" sectioning commands (e. g., `\noindent{\bf My Paragraph}`).
- Do not hide large text blocks using comments or `\iffalse` ... `\fi` constructions.
- Do not use conditional structures to include/exclude content. Instead, please provide only the content that should be published – in one file – and nothing else.
- Do not wrap figures and tables with text. In particular, the package `wrapfig` is not supported.
- Do not change the bibliography style. In particular, do not use author-year citations. (The `natbib` package is not supported.)

This is only a summary containing the most relevant details. Please read the complete document "LIPIcs: Instructions for Authors and the `lipics-v2021` Class" for all details and don't hesitate to contact Dagstuhl Publishing (`mailto:publishing@dagstuhl.de`) in case of questions or comments: `http://drops.dagstuhl.de/styles/lipics-v2021/lipics-v2021-authors/lipics-v2021-authors-guidelines.pdf`

2 Overview

Bichromatic closest pair was first introduced in [1] where Pankaj et al. give a reduction of euclidean minimum spannig tree to two colored bichromatic closest pair and compare their hardness. They also provide a randomized algorithm for computing the bichromatic closest pair which runs in expected $O((nm)^{1-\frac{1}{\lceil \frac{d}{2} \rceil+1}+\varepsilon} + m \log n + n \log m)$ for any $\varepsilon > 0$ and n the size of the first color's set and m the size of the second color's set. Later Aggrawal et al. [2] give an optimal algorithm to solve the k-colored bichromatic all nearest neighbors problem in plane. Their aproach uses a Voronoi diagaram to compute the neighbors and has a running time of $O(n \log n)$ for any number of colors. This in turn gives a $O(n \log n)$ solution to the bichromatic closest pair. Khuller and Matias [13] give a randomized algorithm for the closest pair problem that runs in expected $O(n)$ time which, they argue, can be extended to the bichromatic closest pair problem in higher dimensions with a running time still in expected $O(n)$. In the same year Eppstein [10] gives important algorithms which solve the dynamic Euclidean minimum spanning tree and dynamic bichromatic closest pair. The running time of maintaining the Euclidean minimum spannig tree and two colored bichromatic closest pair is $O(n^{1-\varepsilon})$ where $\varepsilon > 0$ depends on the dimension d. For rectilinear metrics, the MST can be maintained in time $O(\sqrt{n} \log^d n)$. Dumitrescu and Guha [9] gave an MST approach for the k-colored bichromatic closest pair in plane with running time $O(n \log n)$ and reduced the k-colored bichromatic closest pair problem to $\lceil \log k \rceil$ two-colored bichromatric closest pair problems. They also give an algorithm that maintains a bichromatic closest pair under color change with running time $O(\log n)$ per update in any dimension. In the same manner, Borgelt and Borglet [4] give an algorithm that maintains in plain a k-colored bichromatic all nearest neighbors with running time $O(n \log n)$ to construct a Voronoi diagram and amortized $O(\log n)$ time to update it. With Impagliazzo nad Paturi [12] stating the Strong Exponential Time Hypothesis (SETH) we have conditional lower-bounds on some problems. Williams [18] proved that SETH implies Orthogonal Vectors Hypothesis (OVH). O'Donnell et al. [14] gave strict bounds on the locality sensitive hashing approach. Alman et al. [3] give a randomized $(1+\varepsilon)$-approximate all nearest neighbor with expected running time $O(dn + n^{2-1/\Omega(\sqrt[3]{\varepsilon}/\log \frac{1}{\varepsilon})})$ for l_1 and Euclidean metrics. With Bringmann's introduction of Fine-grained complexity theory [5, 6] bichromatic closest pair was researched from a purely theoretical standpoint. Improving on to Alman et al. [3], Rubinstein [16] proved a better conditional lower-bound on the $(1+\varepsilon)$-approximate bichromatic closest pair with time $O(n^{2-\varepsilon})$, which in turn implies better conditional lower-bounds for approximate bichromatic closest pair queries and approximate bichromatic all nearest neighbors. He also pointed out similar results for Hamming distnace and Fréchte distance, whereas for l_1 and l_∞ exist subquadratic algorithms. Pagh et al. [15] proved similar results for $(1+\varepsilon)$-approximate bichromatic closest pair with Jaccard similarity. In a different approach, Flores-Velazco and Mount [11] studied the classification and give a datastructure that can query the most occuring color of the k ε-nearest neighbors of a given point. Horst et al. [17] also provide a datastructure that can answer the most occuring color of the exact k nearest neigbors of a given point.

3 Reduction proof

▶ **Theorem 1.** *The k-colored bichromatic closest pair problem can be reduced to $\lceil \log k \rceil$ 2-colored bichromatic closest pair problems.*

Proof. Let k be the number of different colors and let $A_1, A_2, \ldots, A_k$ be sets of points coresponding to different colors. To make the proof cleaner, let us relable the sets to $A_0, A_1, \ldots, A_{k-1}$. Now construct $\lceil \log k \rceil$ pairs p_i with

$$p_i := \left(\bigcup_{j=0..(k-1) \wedge i\text{-th bit of } j \text{ is } 0} A_j, \bigcup_{j=0..(k-1) \wedge i\text{-th bit of } j \text{ is } 1} A_j \right).$$

For $i > \lceil \log k \rceil$, pairs will be of form $(\bigcup A_j, \emptyset)$ and so they are not needed. By computing the 2-colored bichromatic closest pair between the left and right set of each pair p_i and taking the minimal solution, we get a solution to the k-colored bichromatic closest pair. To prove this, assume the k-colored bichromatic closest pair is achieved between sets A_{r_1} and A_{r_2}. Since $r_1 \neq r_2$ there has to exist a bit in which r_1 and r_2 differ. Let that be the s-th bit. Since they differ in the s-th bit, they are split appart in the pair p_s. Now the 2-colored bichromatic pair on this pair p_s will return this solution (or one equally good). And the minimum over all of the pairs should be as good as this solution.

◀

4 Bichromatic closest pair

▶ **Definition 2** (bichromatic closest pair). *For a given dimension d, and two sets of points $A, B \subset \mathbb{R}^d$ where A are red colored points and B are blue colored points, compute the*

$$\min_{p \in A,\ q \in B} d_2(p, q)$$

where $d_2(p, q)$ is the Euclidean distance.

▶ **Definition 3** (bichromatic closest pair with k-colors). *For a given dimension d, number of colors k, and sets $S_1, S_2, \ldots S_k \subset \mathbb{R}^d$, compute*

$$\min_{p \in S_i,\ q \in S_j,\ i \neq j} d_2(q, p).$$

This problem was introduced in [▪ **TODO:** Geometry book ▪] as a two colored version. Later it was studied from a theoretical aspect because of its hardness [▪ **TODO:** (see SETH and OVH) ▪]. All bichromatic nearest neighbor has its use in practice in a fine number of problems such as [▪ **TODO:** LIST SOME PROBLEMS ▪]. Generally there is no well known algorithm that solves it and is speculated that there can not exist an algorithm that solves it efficiently for general dimension. Special cases such as dynamic bichromatic nearest neighbor queries on plane, reverse bichromatic all nearest neighbors, or the most occuring color in some points k-neighborhood, have algorithms that solve them. Special cases on plane, such as the above mentioned dynamic bichromatic nearest neighbor queries, are solvable in optimal time. [▪ **TODO:** See reduction to Does there exist a copy of two real numbers in multiset ▪] In this paper we provide a simple-to-implement algorithm that solves the general d-dimensional k-colored all bichromatic nearest neighbor with acceptable running time that is also adaptable to solving its variants. The whole setup would look like a framework, so to say, and the ideas behind it are the following.

▶ **Idea 1.** *We would like to reduce the general k-colored variant of this problem to a all-nearest neighbor problem or to a 2-colored bichromatic nearest neighbor problem, since these problems are well known.*

▶ **Idea 2.** *For each point in some set A_i we would like to decompose the whole dataset into some (possibly small) number of disjoint sets $\bigcup_j B_j = A \backslash A_i$ so we could use algorithms for the aforementioned problems.*

The idea 2 leads to a well known algorithm and data structure used in information theory.

▶ **Theorem 4** (Huffman coding). *Given a distribution $p_1, p_2, \dots p_k$, we can find in time $O(k \log k)$ a complete binary tree such that*

$$\sum_{i=1}^{k} p_i \log_2 \frac{1}{p_i} \leq \sum_{i=1}^{k} p_i l_i \leq 1 + \sum_{i=1}^{k} p_i \log_2 \frac{1}{p_i}$$

where l_i is depth of a leaf assigned to the i-th distribution.

Theorem 4 is a well known result in information theory and its proof is located in **[▪ TODO:** Cormen et al ▪ **]**. Lower bound of 4 is due to Shannon, and its upper bound and greedy construction algorithm are due to Huffman. Using this data structure we will try to "encode" each color such that the decomposition from idea 2 has at most $O(\log n)$ disjoint subsets, and we will try to do if for each color. We can do that thanks to the above theorem in the following way.

×

▶ **Theorem 5** (Encoded colors tree). *For a given dimension d, number of colors k and sets $A_1, A_2, \dots A_k \subset \mathbb{R}^d$ we can construct a complete binary tree with leaves associated to sets A_i, $\forall i = 1 \dots n$ and internal nodes associated to union of sets in its subtree in time $O(d\, n \log n)$, where $n = \sum_{i=1}^{k} |A_i|$ is the number of all points.*

Proof. Let us denote the size of each set with $w_i = |A_i|$ and note that w_i is the number of times set A_i will be copied. For these w_i we define $p_i := \frac{w_i}{n}$. p_i make up a distribution which means we can use theorem 4 to construct a binary tree with leaves associated to distributions p_i i.e. sets A_i. During construction we copy each point from the current node to its parent so that root of each subtree has a union of points of its leaves. Each point from set A_i we need to copy l_i times and this will take $O(d\, l_i)$ operations. The set A_i we can copy in time $O(d\, w_i l_i)$. For all sets our running time is $\sum_{i=1}^{k} O(d\, w_i l_i)$ By theorem 4 these l_i are such that

$$\sum_{i=1}^{k} p_i l_i \leq \sum_{i=1}^{k} p_i \log_2 \frac{1}{p_i}.$$

By expanding p_i we have

$$\begin{aligned}
\sum_{i=1}^{t} \frac{w_i}{n} l_i &\leq 1 + \sum_{i=1}^{t} \frac{w_i}{n} \log_2 \frac{n}{w_i} \\
&\leq 1 + \sum_{i=1}^{t} \frac{w_i}{n} \log_2 \frac{n}{\min\limits_i w_i} \\
&= 1 + \frac{1}{n} \log_2 \frac{n}{\min\limits_i w_i} \sum_{i=1}^{t} w_i \\
&\leq 1 + \frac{1}{n} \log_2 \frac{n}{\min\limits_i w_i} n \\
&= 1 + \log_2 \frac{n}{\min\limits_i w_i} \\
&\leq 1 + \log_2 n, \\
\sum_{i=1}^{t} w_i l_i &\leq n \left(1 + \log_2 n\right).
\end{aligned}$$

From this we can see that our running time is $O\left(d\,n\log n\right)$. ◀

▶ Remark 6. For a balanced dataset with $|A_i| = n/k$ we have a running time of $O\left(d\,n\log k\right)$. If we want to remove the d from the running time we can copy just the indices. If we want to optimize it even more we can sort the whole dataset of points by the size of each colored set, and use $O(n)$ Huffman coding algorithm in which we will save only indexes of the first and last point of that subset.

To use this data structure in the way we planed to we need a little bit of additional information. The whole process of additional processing and finding is quite easy and is described in the following theorem.

▶ **Theorem 7.** *With the data structure described in theorem 5 we can for each point from a set A_i find a decomposition of disjoint sets $\bigcup\limits_j B_j = A \backslash A_i$ in amortized $O(\log n)$ time.*

Proof. The idea is to precompute hashes using a *dfs* in which we always descend into the left child first and in each leaf we save a number, at which time we visited it. For each color we have its visit time $time_i$ saved in its leaf. Then for every node we save a number $maxTime = \max\left(leftChild.maxTime, rightChild.maxTime\right)$ that denotes the max $time_i$ in its subtree. For leaves $maxTime = time_i$. This number we can compute while returning from the *dfs* recursion and overall cost will be $O(t)$ which is in worst-case $O(n)$. Now to find a target leaf for point i we do the following:

Algorithm 1 Find(T, i)

```
if isLeaf(T) then
    return ∅.
else if T.leftChild.maxTime < time[i] then
    return {T.leftChild.set} ⋃ Find(T.rightChild, i).
else
    return Find(T.leftChild, i) ⋃ {T.rightchild.set}.
end if
```

By theorem 4 we will descend exactly l_i times for each point. For all points we will descend overall $\sum\limits_{i=1}^{k} w_i l_i = O\left(n\log n\right)$ times which gives $O(\log n)$ amortized time per point. We can easily verify by induction that this algorithm 1 returns a set of sets who in union give $\bigcup\limits_j B_i = A \backslash A_i$ for a target point i, and as such is left as an exercise for the reader. ◀

We have a data structure, now we need to make it usefull. Given theorems 5 and 7 we can go in two directions: online algorithm and offline algorithm. Since this data structure can be view as a binary search tree it seems natural to first explain the online solution for this problem. Now the first thing we need to have for this solution is an algorithm or data structure that has $n \cdot P\left(n\right)$ precompute time and can answer **Which point in this set of points is nearest to some given point not living in this set?** in time $Q(n)$ where n is the size of mentioned set. Using this we get the following two theorems.

▶ **Theorem 8** (Construction time). *Given the Encoded Colors Tree from theorem 5 and a data structure with construction time $O\left(nP_d\left(n\right)\right)$, $P_d(n)$ monotonically increasing, we need overall $O\left(nP_d(n)\log n\right)$ time to build this data structure inside every node of the Encoded Colors Tree.*

Proof. Since Huffman trees are complete binary trees we have $2k-1$ nodes which gives a naive upper bound of $O\left(n^2 P_d(n)\right)$ construction time. Having Huffman tree properties we can do better. Let again $w_i := |A_i|$ and let b_j be the size of the set inside node j, $\forall j = 1 \dots 2k-1$. Our construction time of building the given data structure inside all of the nodes is asymptotically bounded above by

$$\sum_{j=1}^{2k-1} b_j P_d(b_j)$$

which by monotonicity of $P_d(n)$ is

$$\sum_{j=1}^{2k-1} b_j P_d(b_j) \leq P_d(n) \sum_{j=1}^{2k-1} b_j.$$

By definition of b_j and construction of Encoded Colors tree we have

$$\sum_{j=1}^{2k-1} b_j = \sum_{i=1}^{n} w_i l_i$$

which in turn gives us an asymptotic upper bound of

$$\sum_{j=1}^{2k-1} b_j P_d(b_j) = O\left(P_d(n) n \log n\right).$$

◀

▶ Remark 9. A slight optimization can be made if the datastructure can be split on merged when building the tree.

▶ **Theorem 10** (Query time)**.** *Given the augmented Encoded Colors Tree from theorem 8 whose data structure can answer queries of the form* **Find the point inside this set that is closest to the given query point** *in time $O\left(Q_d(n)\right)$, $Q_d(n)$ monotonically increasing, we can answer in amortized $O\left(Q_d(n) \log n\right)$ time* **Find the nearest bichromatic neighbor of the given query point**.

Proof. Proof is trivial and follows from 7 and 8. Let again $w_i := |A_i|$ and b_j be the size of set in node j. Using algorithm 1 from 7 we find l_i disjoint sets for our query point in time $\Theta\left(l_i\right)$. Each of these sets can find the closest point to our query point in time $Q_d\left(b_{ij}\right)$ which means we have an asymptotic upper bound of

$$\sum_{j=1}^{l_i} Q_d\left(b_{ij}\right) \leq \sum_{j=1}^{l_i} Q_d\left(n\right) = Q\left(n\right) l_i.$$

Now since we found the closest point in each of disjoint sets B_j, who in union give $A \backslash A_i$, we know that the closest point of those l_i points is the nearest bichromatic neighbor to our query point, thus we have our answer. Now the overall running time to query all points is bounded above by

$$\sum_{i=1}^{n} w_i \sum_{j=1}^{l_i} Q_d\left(b_{ij}\right) \leq \sum_{i=1}^{n} w_i Q_d\left(n\right) l_i = Q_d\left(n\right) \sum_{i=1}^{n} w_i l_i = O\left(Q_d(n) n \log n\right)$$

which in turn gives us an amortized running time of $O\left(Q_d(n) \log n\right)$ per query. ◀

▶ Remark 11. An optimization can be made by passing information from the above node to the below nodes when doing a query. Information may help in some cases for example when passing the current best distance we can limit the search space for other calculations. Information passing can be done in the reverse manner where we go from leaf to root and propagate results upwards.

We summarize the above two theorems into the following corollary.

▶ **Corollary 12.** *For some dimension d, given a data structure with precompute time $O\left(n\, P_d(n)\right)$ that can answer find the nearest neighbor in time $O\left(Q_d(n)\right)$, we can solve the k-colored all bichromatic nearest neighbor in time $O\left(n \log n \left(P_d(n) + Q_d(n)\right)\right)$, independent of the number of colors.*

Proof. Theorems 8 and 10 ◀

▶ Remark 13. Given superadditive $P_d(n)$ and $Q_d(n)$ we can easily show the running of the above method to be $O(nQ_d(n) + nP_d(n))$ which can be easily proven to be $\Omega(n^2)$.

▶ Remark 14. In terms of Master Theorem we can view the running time of the whole procedure as a recursion of the form

$$T(n) = 2T(n/2) + O(P(n) + Q(n)),$$

which is the case for datasets with same sized colored sets.

Corollary 12 has a constructive proof and, in terms of fine-grained complexity theory, gives a reduction to Nearest Neighbors Search problem. In mathematical terms that would mean the following.

▶ **Corollary 15.** *If nearest neighbor search can be solved in time $O(n^{1-\varepsilon_1} poly(d))$ then k-colored all bichromatic nearest neighbor can be solved in time $O(n^{2-\varepsilon_2} poly(d))$, independent of the number of colors.*

This information isn't anything new. Nearest neighbor search was extensively research in the past decade and a lot of results were proven for conditional hardness of it and similar problems. **[▪ TODO:** CITE A LOT OF WORK from Rubinstein. ▪ **]** Our framework given above is also extendable to k-nearest bichromatic neghbors, bichromatic closest pair, approximate bichromatic nearest neighbor, randomized bichromatic nearest neighbor, in general l_p metric bichromatic nearest neighbor, and so on, given the appropriate data structure. This fact implies a lot of hardness results e.g. approximate nearest neighbor search in $O(n^{1-\varepsilon})$ implies approximate bichromatic nearest neighbor search in $O(n^{1-\varepsilon})$**[▪ TODO:** cite Rubinstine ▪ **]**.

One that we explained the online approach of this framework, we can easily adapt it to an offline approach in which we in bulk calculate the nearest neighbors for sets of points in the following manner.

▶ **Theorem 16** ($k \to 2$)**.** *Given the Encoded Colors Tree form theorem 5 and an algorithm $Alg(S_1, S_2)$ that can solve the 2-color all bichromatic nearest neighbor, we can solve the k-color all bichromatic nearest neighbor.*

Proof. The algorithm is straightforward and is similar to the one in 10. Each left child is disjoint to the right child and in union give the whole subset located in the respective subtree. ◀

▶ Remark 17. Using Alg with running time $f(b_1, b_2)$, we can have a running time of $O(f(n,n) \log n)$ or $O(f(n,n))$ for the k-color all bichromatic nearest neighbor, depending on the properties of $f(b_1, b_2)$.

Algorithm 2 Calculate(T)

if $isLeaf(T)$ **then**
  return $\emptyset$.
else
  $rec\_results := Calculate(T.leftChild) \bigcup Calculate(T.righChild)$,
  return $\min\left(rec\_results, Alg\left(T.lefChild.set, T.righChild.set\right)\right)$.
end if

The offline version of this framework is in principal the same as the online version. The same generalization can be made for this case i.e. we can extend this offline version to k-color bichromatic closest pair problem, approximate bichromatic closest pair problem, randomized

▶ **Corollary 18.** *k-color all bichromatic nearest neighbor is as hard as 2-color all bichromatic nearest neighbor and the running time doesn't depend on the number of colors.*

Approximate bichromatic nearest neighbor

Let $A_1, A_2, \ldots, A_t \subset \mathbb{R}^d$. In *all bichromatic approximate nearest neighbor* problem, for each $i = 1 \ldots t$ and each $q \in A_i$, we want to find a point $z \in \bigcup_{i \neq j} A_j$ such that $d_2(q,p) \leq d_2(q,z) \leq (1+\varepsilon) d_2(q,p)$, where $p \in \underset{p \in \bigcup_{i \neq j} A_j}{\arg\min}\, d_2(q,p)$. We will show that all bichromatic approximate nearest neighbor problem can be computed in $O(n \log n)$ time with the help of the well-separated pair decomposition, a well-know geometric decomposition introduced in a seminar work of Callahan and Kosaraju in 1995 [7, 8].

4.0.0.1 Well-separated pair decomposition.

Let S be a set of n points in $\mathbb{R}^d$. For any $A \subseteq S$, let $R(A)$ denote the minimum enclosing axis-aligned box of A. Let C_A be the minimum enclosing ball of $R(A)$, and let $r(A)$ denote the radius of C_A. Let $C^{r(A)}$ be the ball with the same center as C_A, but with radius $r(A)$. Furthermore, for two sets $A, B \subseteq S$, let $r = \max(r(A), r(B))$, and let $d(A,B)$ denote the minimum distance between C_A^r and C_B^r. For example, if the C_A intersects C_B, then $d(A,B) = 0$.

▶ **Definition 19.** *A pair of sets A and B are said to be well-separated if $d(A,B) > s \cdot r$, for any given separation constant $s > 0$ and $r = \max\{r(A), r(B)\}$.*

▶ **Definition 20** (WSPD)**.** *A well-separated pair decomposition of $S \subset \mathbb{R}^d$, for a given $s > 0$, is a sequence $(A_1, B_1), \ldots, (A_k, B_k)$, where $A_i, B_i \subseteq S$, such that*
1. *A_i, B_i are well-separated with respect to separation constant s, for all $i = 1, \ldots, k$;*
2. *for all $p \neq q \in S$ there exists a unique pair (A_i, B_i) such that $p \in A_i, q \in B_i$ or $q \in A_i, p \in B_i$.*

Note that WSPD always exists since one could use all singleton pairs $(\{p\}, \{q\})$, for all pairs $p, q \in S$. However, this would yield a sequence of dumbbells of size $k = \Theta(n^2)$. The question is whether one could do better than that. The answer to that question was given by the following theorem.

▶ **Theorem 21** ([8])**.** *Given a set S of n points in $\mathbb{R}^d$ and a separation constant $s > 0$, a WSPD of S with $O(s^d d^{d/2} n)$ many dumbbells can be computed in $O(dn \log n + s^d d^{d/2} n)$.*

We will choose the separation constant s and modify the construction of WSPD such that it can be reused in the context of our problem.

Let (A_i, B_i), $i = 1, \ldots, k$, denote the WSPD for some set of points $S \subset \mathbb{R}^d$. For $a, a' \in A_i$ and $b, b' \in B_i$ for some dumbbell i, we make the following observations:

1. Points within the sets A_i and B_i can be made 'arbitrarily close' as compared to points in the opposite sets by choosing the appropriate separation $s > 0$, i.e.

$$d(a, a') \leq 2r < \frac{2}{s} d(A_i, B_i) \leq \frac{2}{s} d(a, b). \tag{1}$$

2. Distances between points in the opposite sets can be made 'almost equal', by choosing the appropriate $s > 0$, i.e.

$$d(a', b') \leq d(a, a') + d(a, b) + d(b, b') < (1 + \frac{4}{s}) d(a, b). \tag{2}$$

Thus, for $s = \frac{4}{\varepsilon}$, we have that $d(a', b') \leq (1 + \varepsilon) d(a, b)$, for any $\varepsilon > 0$.

4.0.0.2 Construction of WSPD with an additional color information.

For the construction of WSPD the split tree of a set S is computed by Algorithm 3. Even

Algorithm 3 SplitTree(S)

```
if size(S) = 1 then
    return leaf(S).
else
    Partition S into sets S_1 and S_2 by halving R(S) with hyperplane along its longest side.
    Return a node with children (SplitTree(S_1), SplitTree(S_2)).
end if
```

though such a tree might have linear depth and therefore a naive construction of the split tree by Algorithm 3 in the worst case takes quadratic time. However, the work of [8] showed how to compute such a binary tree in $O(n \log n)$ time. With every node u of that tree we can conceptually associate the set S_u of all points contained in its subtree. Node u is called colorful node if S_u contains points from two or more colors. Otherwise, we say that S_u is monochromatic.

▶ **Definition 22** (Colorful edge). *Let u and w denote two nodes in a split tree such that the corresponding set of points (S_u, S_w) for a well-separated pair. If either S_u or S_w is colorful, then the (S_u, S_w) is called colorful edge. If both S_u and S_w are monochromatic, but colored with different colors, then (S_u, S_w) is called colorful edge.*

In order to compute the subset of WSPD consisting only of colorful edges, for each internal node u of the split tree T with children v, w and v_l, v_r and w_l, w_r denoting left and right child of v, w respectively, we invoke Algorithm 4. Let C_S denote the set of colorful edges computed by Algorithm 4. Note that C_S is a subset of WSPD of S. Furthermore, for any node u of the split tree T, let $C_{S_u} \subseteq C_S$ denote the set of all colorful edges such that $(S_w, S_v) \in C_S$, for nodes $v, w \in T$, and v ancestor of u. We consider u to be an ancestor of itself. Let $(S_{w'}, S_u) = \arg\min_{(S_w, S_v) \in C_{S_u}} d(S_w, S_v)$, i.e. $(S_{w'}, S_u)$ is the 'shortest' colorful edge in C_{S_u}. We save that information with every node $v \in T$.

▶ **Theorem 23.** *Given a set S of n points in R^d, AND k COLORS, the approximate all-bichromatic nearest neighbors problem can be solved in $O(n \log n)$ time.*

Algorithm 4 FindColorfulEdges(v, w)

Require: Split tree T of S, separation constant $s > 0$
if S_v and S_w are well-separated and (S_v, S_w) colorful **then**
  add colorful edge (v, w) to the tree T.
else if $L_{\max}(S_v) > L_{\max}(S_w)$ **then**
  FindColorfulEdges(v_l, w), FindColorfulEdges(v_r, w)
else
  FindColorfulEdges(v, w_l), FindColorfulEdges(v, w_r)
end if

Proof. Let p by any point in S and let $q \in S$ be its bichromatic nearest neighbor. Let u denote the leaf in the split tree that stores q, and let $(S_{w'}, S_u$ is the shortest colorful edge saved with u. Then $p \in$ ◀

References

1 Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, page 203–210, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/98524.98567.

2 Alok Aggarwal, Herbert Edelsbrunner, Prahakar Raghavan, and Prasoon Tiwari. Optimal time bounds for some proximity problems in the plane. *Information Processing Letters*, 42(1):55–60, 1992. URL: https://www.sciencedirect.com/science/article/pii/002001909290133G, doi:10.1016/0020-0190(92)90133-G.

3 Josh Alman, Timothy M. Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications, 2016. arXiv:1608.04355.

4 Magdalene G. Borgelt and Christian Borgelt. Notes on the dynamic bichromatic all-nearest-neighbors problem. 2008.

5 Karl Bringmann. Fine-grained complexity theory. 2019.

6 Karl Bringmann. Fine-grained complexity theory: Conditional lower bounds for computational geometry. In *Lecture Notes in Computer Science*, pages 60–70. Springer International Publishing, 2021. URL: https://doi.org/10.1007%2F978-3-030-80049-9_6, doi:10.1007/978-3-030-80049-9_6.

7 Paul B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its application*. PhD thesis, Dept. Comput. Sci., Johns Hopkins University, Baltimore, Maryland, 1955.

8 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, jan 1995. doi:10.1145/200836.200853.

9 Adrian Dumitrescu and Sumanta Guha. Extreme distances in multicolored point sets. In Peter M. A. Sloot, Alfons G. Hoekstra, C. J. Kenneth Tan, and Jack J. Dongarra, editors, *Computational Science — ICCS 2002*, pages 14–25, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

10 D. Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Discrete & Computational Geometry*, 13(1):111–122, Jan 1995. doi:10.1007/BF02574030.

11 Alejandro Flores-Velazco and David M. Mount. Boundary-Sensitive Approach for Approximate Nearest-Neighbor Classification. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14625, doi:10.4230/LIPIcs.ESA.2021.44.

12 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. URL: `https://www.sciencedirect.com/science/article/pii/S0022000000917276`, `doi:10.1006/jcss.2000.1727`.

13 S. Khuller and Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. *Information and Computation*, 118(1):34–37, 1995. URL: `https://www.sciencedirect.com/science/article/pii/S0890540185710498`, `doi:10.1006/inco.1995.1049`.

14 Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality sensitive hashing (except when q is tiny), 2009. `arXiv:0912.0250`.

15 Rasmus Pagh, Nina Stausholm, and Mikkel Thorup. Hardness of bichromatic closest pair with jaccard similarity, 2019. `arXiv:1907.02251`.

16 Aviad Rubinstein. Hardness of approximate nearest neighbor search, 2018. `arXiv:1803.00904`.

17 Thijs van der Horst, Maarten Löffler, and Frank Staals. Chromatic k-nearest neighbor queries. *CoRR*, abs/2205.00277, 2022. `arXiv:2205.00277`, `doi:10.48550/arXiv.2205.00277`.

18 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). URL: `https://www.sciencedirect.com/science/article/pii/S0304397505005438`, `doi:10.1016/j.tcs.2005.09.023`.

A Styles of lists, enumerations, and descriptions

List of different predefined enumeration styles:

- `\begin{itemize}...\end{itemize}`
- ...
- ...

1. `\begin{enumerate}...\end{enumerate}`
2. ...
3. ...

(a) `\begin{alphaenumerate}...\end{alphaenumerate}`
(b) ...
(c) ...

(i) `\begin{romanenumerate}...\end{romanenumerate}`
(ii) ...
(iii) ...

(1) `\begin{bracketenumerate}...\end{bracketenumerate}`
(2) ...
(3) ...

Description 1 `\begin{description} \item[Description 1] ...\end{description}`
Description 2 Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.
Description 3 ...

Proposition 27 and Proposition 27 ...

B Theorem-like environments

List of different predefined enumeration styles:

▶ **Theorem 24.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Lemma 25.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Corollary 26.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Proposition 27.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Conjecture 28.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Observation 29.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Exercise 30.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Definition 31.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Example 32.** Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Note 33. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Note. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Remark 34. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Remark. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▷ Claim 35. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▷ Claim. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

Proof. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque. ◀

Proof. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque. ◁