Foundations of Geometric Methods in Data Analysis
Search algorithms in metric trees

Côme Clément, Florez de la Colina Inès

Mars 2021

1 Introduction

In this project report we will present our work on the search algorithms used for the search of nearest neighbors in metric trees and metric forests. We will present our implementations of metric trees and metric forests and we will focus on two main search algorithms, the exact search with pruning and the defeatist search. We will see the advantages and disadvantages of both methods especially when it comes to the number of visited nodes during the search and the accuracy of the results.

2 Implementation of Metric trees and Searching Algorithms

2.1 Building a Metric Tree

Before being able to build a metric tree, we first need to properly define metric spaces. Indeed, the data we use to build our metric tree is data that lives in a metric space. A metric space is defined as a set associated with a metric. Thus, a metric space is a pair of a set and a distance measure (M, d) where M is the set and d the distance, a.k.a the metric. The metric is defined as $d : M \times M \to \mathbb{R}$ such that $\forall (x, y, z) \in M$:

- identity of indiscernibles: $d(x, y) = 0 \iff x = y$
- symmetry: $d(x, y) = d(y, x)$
- triangle inequality: $d(x, y) \leq d(x, y) + d(y, z)$

Now that we have defined our metric space, we can build a metric tree. Let's consider a set of N points $(x_1, x_2, ..., x_N)$. To initialize the metric tree, we need to choose a pivot v. In this section, the pivot will be chosen randomly amongst a set of candidate points from the data set. Nevertheless, we will see in the last section of this report how to choose a pivot to make the search for nearest neighbors more efficient. We subdivide the space into two sub-spaces as follows:

- right tree: $\forall i \in [\![1, N]\!], d(v, x_i) \geq \mu$
- left tree: $\forall i \in [\![1, N]\!], d(v, x_i) < \mu$

where μ defines the cut distance between the two sub-trees and is chosen to be the median distance between the pivot and all the other points from the tree. To keep building the metric tree we choose a pivot in both sub-trees and reiterate the first step until we reach the leaves of the tree.

Metric Tree Class

Our metric tree class has several arguments: the root, the median value μ, the minimal and maximal distances from the sub-tree to the root and the size of the sub-tree, i.e the number of points in the sub-tree. Here is an example of metric tree built using our class for a total number of 10 points:

```
>>> full_tree = MetricTree(points)
    MetricTree: root=[0.83244264 0.21233911], size=10
>>> full_tree.right
    MetricTree: root=[0.37454012 0.95071431], size=5
>>> full_tree.right.mu
    0.35447574406213617
>>> full_tree.left.min_left
    0.1705893848218244
```

Moreover, you can see on Figure 1 the different steps of building the metric tree for the first subdivision into two sub-trees.

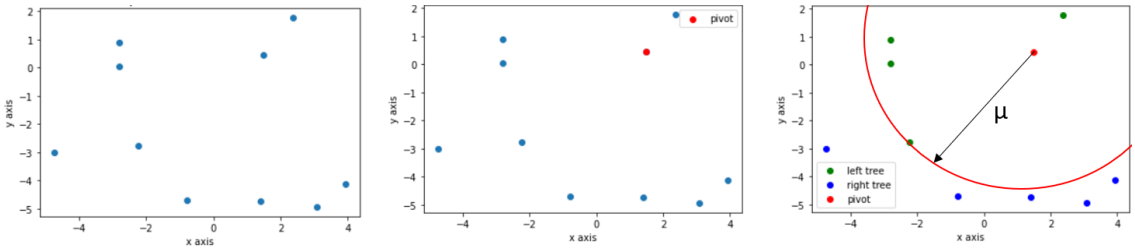


Figure 1: Metric tree points (left), Pivot chosen (middle), Division into two sub-trees (right)

We can see on Figure 1 that the choice of the pivot separates the metric space into two sub-spaces with 4 (left) and 5 (right) points each.

We will now see two important nearest neighbors search algorithms.

2.2 Exact Search with Pruning condition

The nearest neighbor (NN) search consists of finding the closest point amongst the given data points, to a given query. Here, the closeness of the point depends on the metric chosen. The exact search is very easy to implement but is computationally very expensive as we compute the distance between the query point and all the points from the metric tree. Thus we can use a pruning condition to reduce the number of visited nodes during the search.

The pruning condition takes advantage of the triangle inequality of the metric. Let's consider a set of N+1 points and denote by v, the chosen pivot. The remaining N points are then divided into two sub-trees. As stated in the previous section, we have computed, for each sub-tree the minimum and maximum distances to the pivot. Let's denote by (dl_{min}, dl_{max}) the minimum and maximum distances from the left tree to the pivot and (dr_{min}, dr_{max}) those of the right tree. We consider a length τ and define two intervals $l_l = [dl_{min} - \tau, dl_{max} + \tau]$ and $l_r = [dr_{min} - \tau, dr_{max} + \tau]$. For a given l, we get the following properties:

- if $l \notin l_l$ then the left sub-tree can be pruned.
- if $l \notin l_r$ then the right sub-tree can be pruned.

For the implementation of this algorithm, we have followed the implementation presented in the lecture slides (course 1, slide 55). The advantages of using the exact search with pruning condition is that we have the certainty of finding the correct nearest neighbor and by using the pruning condition we can decrease the number of visited nodes, hence reduce the computational complexity of our algorithm. Of course, this reduction of complexity depends on the number of times we can use pruning and this depends on the choice of the pivot. We will present in the last section of this report a method to choose the pivot to minimize the number of visited nodes.

2.3 Defeatist Style Strategy

The defeatist style strategy consists of only visiting one sub-tree at each iteration. We can immediately see that this strategy highly reduces the complexity of the algorithm, but unfortunately it is prone to errors. The idea of this algorithm is to recursively visit the sub-tree containing the query point. Once again, for our implementation, we have implemented the algorithm presented in the lecture slides (course 1, slide 22).

2.4 Comparing both searching methods

To compare the accuracies of both searching methods, we generated 1000 points drawn randomly and computed the average accuracy over the search for the nearest neighbors of 10000 random queries, just to get an idea of the performance of both searching methods from the accuracy point of view. The results can be see in Table 1.

As expected, the pruning search has an accuracy of 1, while the defeatist search has an accuracy of 0.59. This is due to the fact that we only search one of the sub-trees at each iteration. We will compare in section 4 the average number of visited nodes for both methods.

Table 1: Metric Tree: Average accuracy of both searching methods for 10000 queries performed on randomly drawn data (1000 data points)

Search Method	**Average accuracy**
Pruning search	1.0
Defeatist search	0.59

3 Implementation of Metric forests and Searching algorithms

For the implementation of the metric forest we have decided to follow the implementation presented in [2]. This time, instead of dividing the space into two sub-spaces (left and right) we will divide it into three sub-spaces. This division is called an m-split with left, middle and right subsets. These subsets are defined as follows: for an ordered set $X = \{x_1, ..., x_N\}$ and $m \in [0, 1]$ we consider: $w = \lfloor mN \rfloor$ and $a = \lfloor (N - w)/2 \rfloor$ and define the m-split as:

$$\begin{aligned} L &= \{x_i | i \leq a\} \\ M &= \{x_i | i > a, i \leq a + w\} \\ R &= \{x_i | i > a + w\} \end{aligned} \tag{1}$$

We will build our metric forest by performing m-splits to divide the initial space into three sub-spaces and reiterate the process until we obtain leaves of size 1 for the left and right sub-trees. We will concatenate all the points contained in the Middle sub-spaces and reiterate the process until no more middle sub-spaces are created. An example of a metric forest can be seen on Figure 2.

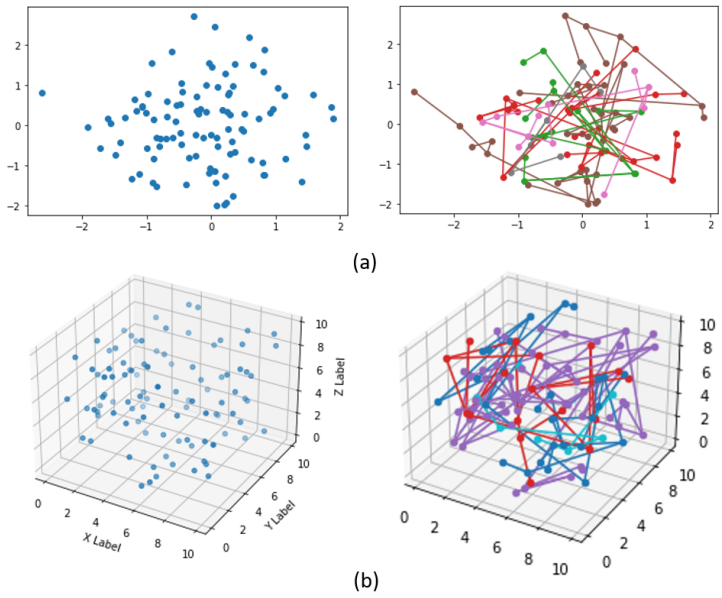


Figure 2: Initial points (left) and built forest (right): Metric forest in 2D (a), Metric forest in 3D (b)

For the searching algorithms in the metric forest, we compute the nearest neighbor in each tree and then find, from all the nearest neighbors already computed, the actual nearest neighbor to the given query. Hence we can use the algorithms implemented in the previous section.

Once again we can compare the accuracies between both search methods.

3.1 Comparing both searching methods

As in the previous section, we compared the accuracies of both searching methods, with the exact same set up, 1000 data points drawn randomly and we performed 10000 random queries. The results can be see in Table 2.

We can make the same observations as for the metric tree. Nevertheless it is interesting to notice that defeatist search has better accuracy for the metric forest than for the metric tree. This is due to the fact that we compute the nearest neighbors of all the trees of the forest and then take the nearest of those neighbors. This decreases the error of the defeatist search.

Table 2: Metric Forest: Average accuracy of both searching methods for 10000 queries performed on randomly drawn data (1000 data points)

Search Method	Average accuracy
Pruning search	1.0
Defeatist search	0.84

4 Experiments

In this section we will convey experiments on two types of data, data drawn randomly and data drawn from a mixture of Gaussians in euclidean spaces of dimension D. On Figure 3, we can see an example of such data drawn from a euclidean space of dimension D=2.

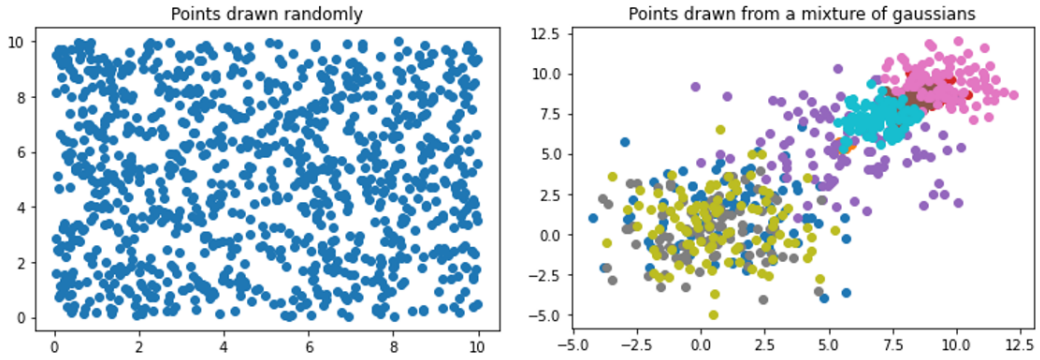


Figure 3: Data drawn randomly (left), Data drawn from a mixture of Gaussians (right)

4.1 Varying the dimension of the Ambient Space

For the experiments, we have chosen to compare the average number of visited nodes for both search algorithms, for data drawn randomly and data drawn from a mixture of Gaussians. First of all, we focused on varying the dimension of the euclidean ambient space. For this we carried out several experiments. First we looked, for a given number of points in the metric forest, the effects of increasing the dimension of the ambient space. The results of this experiment can be seen on Figure 4.

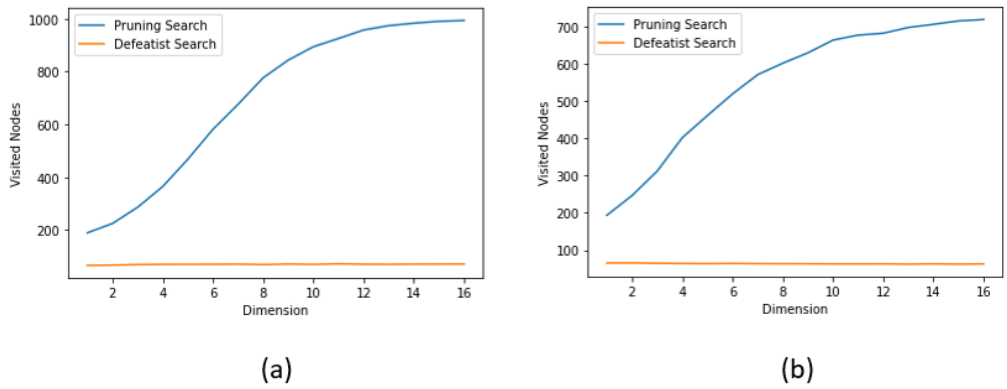


Figure 4: Variations of the average number of visited nodes according to the dimension of the ambient space for: (a) Data drawn randomly, (b) Data drawn according to a mixture of Gaussians

From this Figure we can see that the number of visited nodes stays constant for the defeatist search, which was expected as the number of trees in the forest stays constant. On the other hand, the number of visited nodes increases, with the dimension, for the pruning search. We observe that when the data is drawn from a Gaussian distribution, the number of visited nodes increases more slowly than for the data drawn randomly. The average number of visited nodes starts at around 200 for both datasets, nevertheless, it increases much more for the data drawn randomly.

Our second experiment was based on varying the number of trees of the Metric Forest for a fixed dimension of the ambient space and see how this parameter influences the average number of visited nodes. The results can be seen on Figure 5.

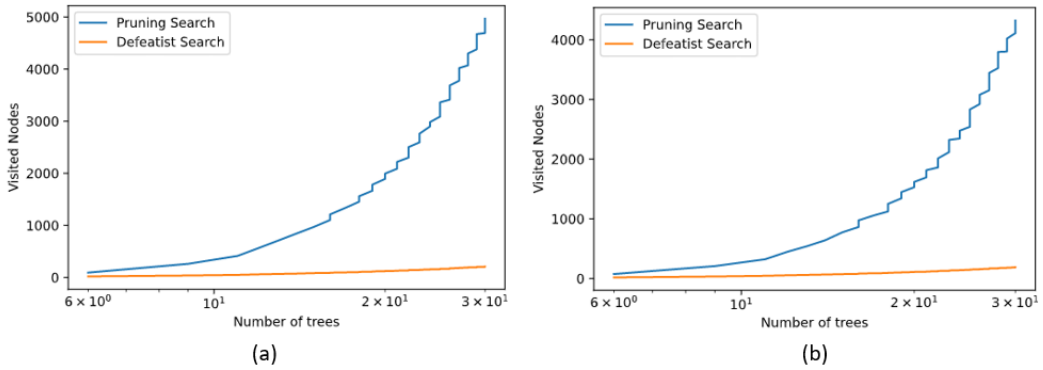


Figure 5: Variations of the average number of visited nodes according to the number of trees in dimension 8 for: (a) Data drawn randomly, (b) Data drawn according to a mixture of Gaussians

On this Figure, we can see that the number of visited nodes increases very slowly for the defeatist search compared to the pruning search where it increases quite quickly. The increase in the number of visited nodes for the defeatist search was expected as the number of trees increases with the size of the database. The difference between the average number of visited nodes for the pruning search, for the data drawn randomly and drawn according to a mixture of Gaussians is less than in the previous experiment. Here, both curves increase in a very similar way.

4.2 Varying the dimension of the ambient space while keeping the intrinsic dimension data constant

For this subsection, we decided to convey experiments for data drawn from a euclidean space of dimension d for search queries drawn from a euclidean space of dimension D (where $d \leq D$). d is called the intrinsic dimension of the data. An example can be seen on Figure 6 where the intrinsic dimension of the data is d=2, and the dimension of the ambient space is D=3.

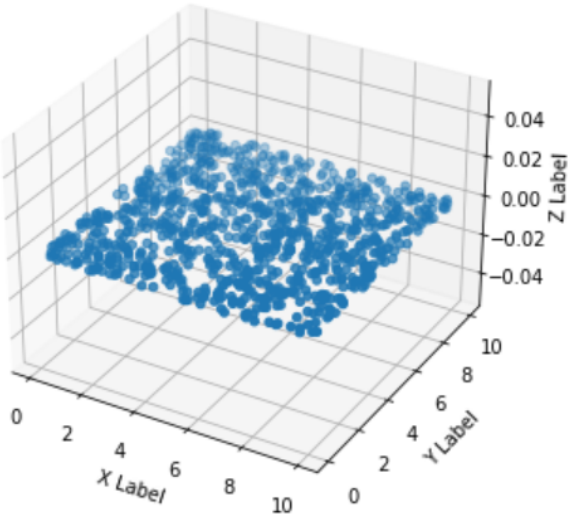


Figure 6: Data drawn from a space of intrinsic dimension equal to 2

In the run experiment, we wanted to see how the average number of visited nodes varied, for a fixed intrinsic dimension d and a varying ambient dimension D. To do so, we run queries of dimension, the dimension of the ambient space. The results can be seen on Figure 7.

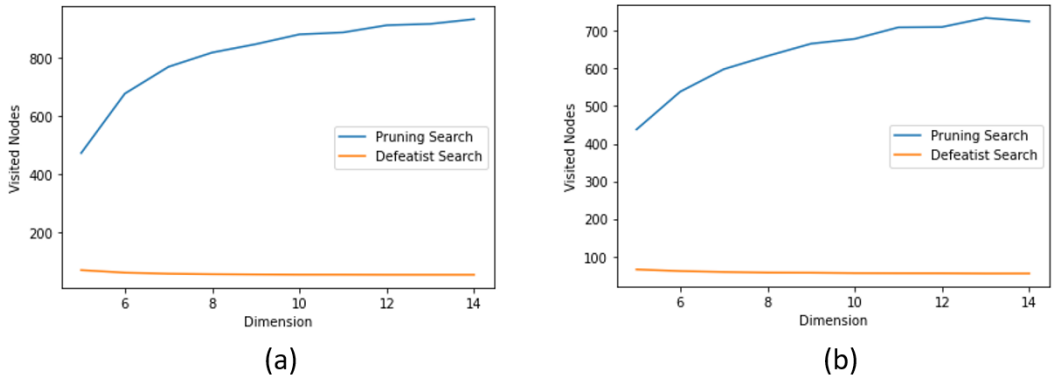


Figure 7: Variations of the average number of visited nodes according to the the dimension of the ambient space for a constant intrinsic dimension d=5 for: (a) Data drawn randomly, (b) Data drawn according to a mixture of Gaussians

We can see from this Figure that the average number of visited nodes stays almost constant for the defeatist search when the dimension of the ambient space increases while the intrinsic dimension stays constant. On the other hand it increases for the pruning search. Once again, we can see that fewer nodes are visited for data drawn from a mixture of Gaussians. What is interesting here, is to compare these results with the ones obtained in Figure 4. Indeed, for the same dimension D of the ambient space, if the intrinsic dimension of the drawn data is smaller than D, then the average number of visited nodes is smaller.

Overall, from the experiments presented in this section, we can conclude that the average number of visited nodes is much less for the defeatist search than for the exact pruning search. This was expected as we only visit one sub-tree at each iteration for the defeatist search. The average number of visited nodes for the defeatist search doesn't seem to vary whether the data shows some kind of structure or not. Moreover, for the pruning search, if the data follows some structure, like being drawn from a mixture of Gaussians, then the average number of visited nodes is smaller, i.e the search algorithm is able to perform more pruning.

5 Running experiments for comparing images

In this section, we decided to focus on comparing images using the Earth Mover distance. The Earth Mover Distance (EMD), is a measure of distance between two probability distributions over a certain region, in our case this region is a metric space M. Another name for this distance is the Wassertein metric. EMD is based on the minimal cost that must be paid to transform one distribution into the other [3]. The advantages of the EMD is that it is based on a solution to the transportation problem from linear optimization for which many very efficient algorithms are available. Moreover it allows for partial matching. The EMD is more robust than histogram-based matching techniques as it can operate on variable-length representations of the distributions. Indeed, in the bin-to-bin method, the computed distance tends to overestimate the true distance as neighboring bins are not considered. On the contrary, mixing methods tend to underestimate distances. Hence, the use of the EMD for image retrieval.

The EMD can be formulated and solved as a transportation problem. To properly define it, lets consider the following weighted point sets (cf lecture slides): $P = \{(p_1, w_{p1}), ...(p_m, w_{pm})\}$ and $Q = \{(q_1, w_{q1}), ...(q_m, w_{qm})\}$. Moreover we consider a distance d(.,.) such that $d_{ij} = d(p_i, q_j)$ and a transport plan where f_{ij} correspond to the non-negative flows circulating on the edges of the bipartite graph PxQ. Hence we have the following optimization problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{ij} f_{ij} d_{ij} \\
\text{subject to} \quad & f_{ij} \geq 0, \\
& \sum_{j} f_{ij} \leq w_{p_i}, \forall i \\
& \sum_{i} f_{ij} \leq w_{q_j}, \forall j \\
& \sum_{i} \sum_{j} f_{ij} = min(W_P, W_Q)
\end{aligned}
\tag{2}
$$

Finally, as previously stated, the EMD is a measure of the cost that must be paid to go from one distribution to the other, hence we we:

$$
d_{EMD} = \frac{\sum_{ij} f_{ij} d_{ij}}{\sum_{ij} f_{ij}} \tag{3}
$$

The EMD is computed thanks to the opencv implementation. Although this implementation is highly time-consuming (∼3s to compare 2 grayscale images of size 32x32), as it is the most straightforward solution, we decided to stick with it.

The dataset we used for this image retrieval task is the well-known CIFAR-10. In order to have reasonable execution time, we chose the 30 first images of this dataset in grayscale, to construct the Metric Tree. Building the tree takes approximately 5 minutes and performing a query takes around 15 seconds on our computer.

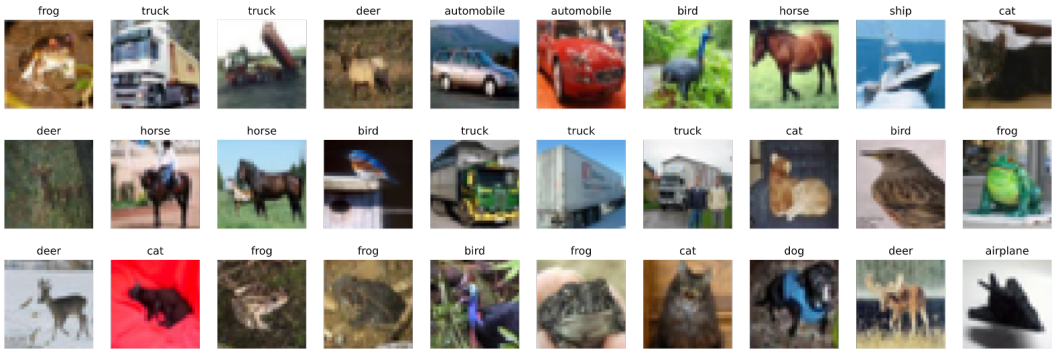


Figure 8: CIFAR-10 samples used to build a metric tree

We get both surprising and expected results (Figures 9 and 10). It is not surprising as we use very few images due to the time constraint on the computation of the Earth Mover's Distance. If we used a less straightforward but more efficient implementation of the EMD for images we could expect to get more expected results than surprising ones. Still, in the Figure 10 we can still recognize that the two images are somehow similar as there is a white background and a darker object in the center.

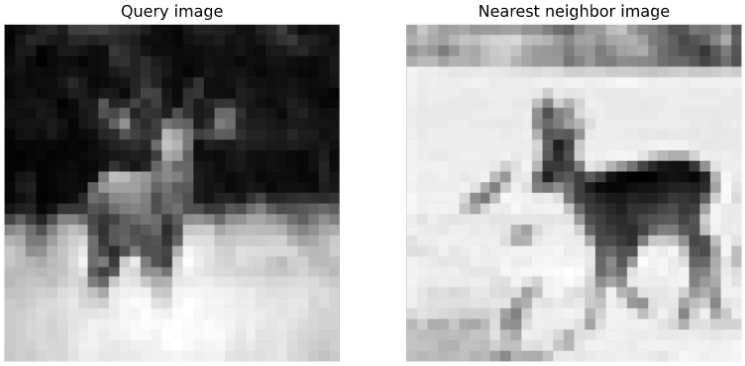


Figure 9: Example where the nearest neighbour search gives a coherent result

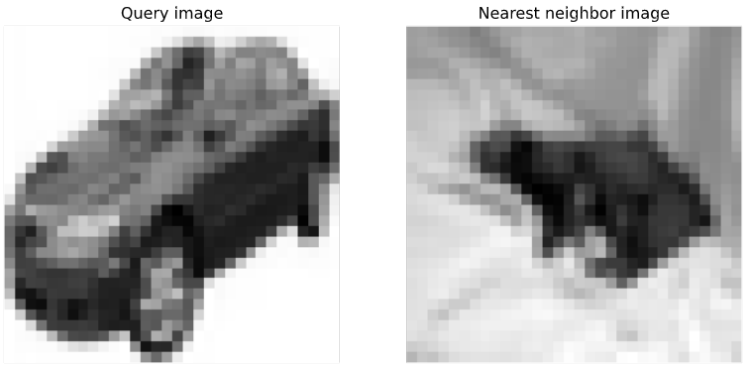


Figure 10: Example where the nearest neighbour search gives a surprising result

6 Deterministic Strategy for choosing the pivot

In the pruning exact search, the method used to choose the pivot can drastically change the number of visited nodes during the search for nearest neighbors. Instead of using the randomized choice, we decided to implement the deterministic method presented in [4]. The first pivot point of the tree, i.e the root is chosen as follows: we choose a set of candidate points from the data set (in our implementation we choose a proportion p=20% of the data set). For each of the chosen points, we compute the distances between this point and all the remaining sample points. Then, we compute the mean and the standard deviation of the distances found and to find the pivot, we choose the point with the maximum standard deviation. Then at each iteration of building the metric tree, we choose for the next pivot, the point that is the farthest away from the root of the sub-tree, and we iterate this process. In [4], the new pivot chosen is supposed to be the one farthest away from all of the pivots already computed. This would increase the complexity of building the tree as we have to compute more distances. Hence we decided to take the point the farthest away from the root of the sub tree considered. To show the efficiency of this method, we have generated sets of randomly chosen data points and we have plotted the average number of visited nodes for an increasing size of the dataset. The average is done over 1000 queries. The results of this method can be seen on Figure 11.

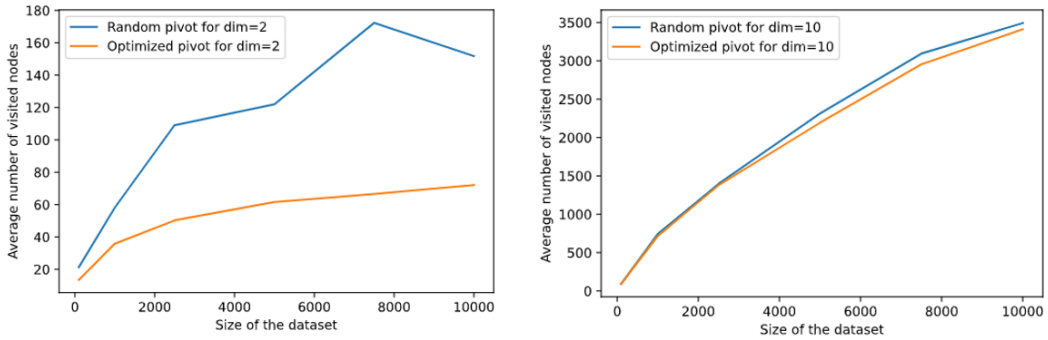
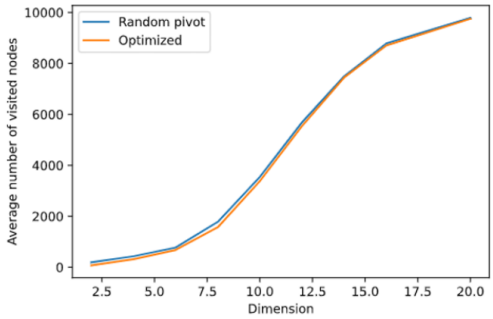


Figure 11: Variations of the average number of visited nodes for randomly drawn data according to the choice of the pivot for different dimensions of the Euclidean space: (left) dimension 2, (right) dimension 10

From both of these graphs, we can see that the average number of visited nodes is lower with the method proposed than with the random pivot choice. Overall, with the increase of the size of the data set, the gap between the number of visited nodes for both methods increases. We can now take a look at how both methods behave when we increase the dimension of the Euclidean space, for a fixed number of data points (we have chosen 10000 data points). The results can be seen on Figure 12.

We can see from this figure that when the dimension of the Euclidean space increases, the gap between the average number of visited nodes for both methods gets smaller. Indeed, in dimension 2, the number this number with the optimized method only represents 38% of the average number of visited nodes with the random pivot choice. On the contrary, for dimension 20, it represent 99.6%. Hence, this optimized method is very beneficial in low dimensions, and looses its benefits in higher dimensions.

Overall, we saw from both Figures 11 and 12, that the average number of visited nodes is smaller when we use the optimized pivot choice presented in this section.



Method for pivot choice	Average visited nodes for dimensions 2, 4, 6, 8, 10, 12, 14, 16 and 20
Random choice	193.415, 429.106, 769.479, 1780.958, 3543.998, 5676.926, 7491.278, 8783.949, 9788.153
Optimized choice	74.62, 311.198, 676.729, 1568.469, 3379.982, 5530.895, 7443.78, 8699.212, 9752.183
Ratio between optimized and random pivot choice	38%, 72%, 87%, 88%, 95%, 97%, 99%, 99%, 99.6%

Figure 12: (left) Variations of the average number of visited nodes according to the dimension of the Euclidean space, (right) Average number of visited nodes and ratio

7 Conclusion

By doing this project we deepened our knowledge on metric trees and discovered how to build metric forests. We also compared two different nearest neighbor search algorithms and saw that the exact pruning search presents an accuracy of 100%, whereas this is not the case for the defeatist search. On the other hand, the defeatist search is much quicker as it visits significantly less nodes than the exact pruning search. Hence the use of one of the two methods depends on the user's wishes. Does the user need a method that will predict the exact nearest neighbor or does the user only need to find a quick approximation of this neighbor?

References

[1] Peter N Yianilos *Data structures and algorithms for nearest neighbor search in general metric spaces*. In ACM SODA, volume 93, pages 311–321, 1993

[2] Peter N Yianilos *Excluded Middle Vantage Point Forests for Nearest Neighbor Search*. July 20, 1998

[3] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas *The Earth Mover's Distance as a Metric for Image Retrieval*.

[4] Ada Wai-chee Fu, Polly Mei-shuen Chan, Yin-Ling Cheung, Yiu Sang Moon *Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances*. January 31, 2000