Near neighbor search in nonmetric space

Joannès Vermorel

Near Neighbor Search in Nonmetric Spaces

Joannès Vermorel
Computational Biology Group, École des Mines de Paris
joannes.vermorel@ensmp.fr

August 30, 2005

Abstract

We consider the computational problem of the Near Neighbor Search (NNS) in nonmetric spaces. Nonmetric spaces are the generalization of the metric spaces because they do not require the triangular inequality assumption. Nonmetric spaces are important because many similarity measures (between images, proteins, etc) do not verify the triangular inequality.

We show the nonmetric situation calls for different evaluation criterions of NNS algorithms. As a first attempt, to our knowledge, to perform general nonmetric NNS, we introduce such evaluation criterions. The insights provided by those criterions lead us to introduce a new category of search structures, called *densitrees*, that extend the classical metric tree algorithm for the nonmetric NNS.

Against well established datasets, our preliminary empirical results lead us to a counter-intuitive conclusion: *the triangular inequality has only a secondary contribution on the efficiency metric NNS*. Additionally, the densitrees, although very naively implemented, performs reasonably well both in metric and nonmetric situations.

1 Introduction

A Near Neighbor Search (NNS) consists of finding points similar to a given query point within a given dataset. Performing fast NNS is critical for many applications: pattern recognition or statistical learning [13], multimedia information retrieval, protein database search, audio and video compression.

Historically, the methods used to perform search operation were leveraging the *structure* of the data, often numerical or alphabetical. In such a simple case where a total order exist between the data points, NNS are performed in logarithmic time of the size of dataset (see [9]). Traditional databases, structured around the notion of relational algebra, deal with the notion of *exact structured search*. Language like SQL emphasizes the combination of explicit total ordering criterions. This approach has the clear advantage of being computationally

very efficient. Nevertheless, the main drawback of the structured search lies in the tight bound of the method itself to a certain type data. This bound between the search method and the data structure raises two issues:

- How do we deal with data that does not fit a relational representation? For example, proteins are encoded sequences of variable length of amino acids.
- How do we deal with data when structured search criterions cannot express the desired similarity? For example, images (of identical dimensions) can be represented as fixed length real vectors, but dimension-wise similarity criterion will lead to a poor notion of similarity.

Those issues have led to a more general and unifying approach of NNS problem: *searching in a dataset for the closest points to a given query point according to a similarity function.* Although this paper focuses on the *nonmetric* NNS, we will first introduce the metric NNS that has been the subject to a much more extensive literature (see [6] for a review).

Definition 1 (Metric function and metric space) *Let E be a set. A function $d : E^2 \rightarrow \mathbb{R}^+$ is said to be a metric over E if d verifies the separation assumption $\forall (x, y) \in E^2, d(x, y) = 0 \Leftrightarrow x = y$, the symmetry assumption $\forall (x, y) \in E^2, d(x, y) = d(y, x)$ and the triangular inequality $\forall (x, y, z) \in E^3, d(x, y) + d(y, z) \leq d(x, z)$. A couple (E, d) is called a metric space if d is a metric over E.*

Intuitively, if a dataset contains points drawn from a *metric space*, then those assumptions ensure that the distance d has a behavior somewhat similar to the simple geometric euclidian distance. In the following, let E be the dataspace, X be the dataset with a cardinal $n = |X|$ and $d : E^2 \rightarrow \mathbb{R}^+$ a metric.

The NNS literature is mainly devoted to algorithms performing (exactly or approximated) those two types of neighbor queries: range queries and near neighbors queries. The **range query** is the selection within X of all points included in a ball of center q the query point and of a given radius r, $\mathcal{B}_d(q, r) = \{x \in X | d(q, x) \leq r\}$. The **near neighbors query** is the selection within X of the k nearest points of the query point q such as $\mathcal{N}_d(q, k) = \{x_1, \ldots, x_k | \forall x \in X, d(q, x) < d(q, x_k) \Rightarrow \exists j, x = x_j\}$. Because of the size of the dataset, the analysis of the computational costs of such algorithms can take many factors into account such as raw CPU, I/O and memory [7]. Nevertheless, the most classical cost criterion consists simply of counting the number of calls to the metric function to achieve the query [6].

In order to achieve better performance than the naive exhaustive search, the metric NNS algorithms relies on the metric space assumptions. The strongest of the three assumptions (see Definition 1), is the *triangular inequality.* Intuitively the classical metric NNS algorithms, like the *metric tree* [21, 19, 7], makes an intensive use of the triangular inequality to prune the search in order to achieve a sub-linear computational cost (in term of number of call to the similarity

function). The two other assumptions are weaker: as described in [6], simple yet general methods exists to adapt any metric NNS algorithm if the two other assumptions are removed.

In the other hand, the metric assumptions (in particular the triangular inequality) do not hold in the nonmetric NNS case. In order to distinguish from the usual metric notation, let (E, s) be the general (nonmetric) space considered in the following. The similarity function $s : E^2 \to \mathbb{R}_+$ simply reflects the notion of point proximity (the smaller $s(p, q)$ the closer the points p and q). Let us motivate the need for a *nonmetric* NNS. Many similarity criterions (the well-known Smith-Waterman distance [20] between sequences for example, or various distances between images [14]), although being widely used do not verify the the triangular inequality[1]. The need for efficient NNS algorithms is such nonmetric cases is often even stronger than in the metric case because of the tremendous costs such similarity functions compared to the low computational cost of the norms $\|.\|_2$ or $\|.\|_\infty$ that are very common in metric spaces [16].

The paper is organized as follow. The Section 2 introduces our evaluation framework for the nonmetric NNS. This framework is compared to the classical evaluation framework used in the metric NNS. The Section 3 introduces a new data structure called *densitree* (contraction of "density tree") that performs approximate nonmetric NNS. Densitrees along other methods are empirically evaluated in Section 4 against well established datasets.

2 Nonmetric NNS framework

To our knowledge, no general approach[2] have been proposed so far for the nonmetric case. In our opinion, the main reason of this situation is the most immediate consequence of the nonmetric assumptions: there is no "hard" assumption that could be used to ensure any guaranteed search pruning. Therefore the only "exact" algorithm that performs queries in nonmetric space is the naive exhaustive search algorithm. Therefore the discussion in the following is narrowed by necessity to the *approximate* nonmetric NNS algorithms.

We will first discuss in Section 2.1 how the nonmetric NNS algorithm can be evaluated. This discussion shows the weaknesses of the classical evaluation approach for approximate metric NNS algorithms and introduces a more specific evaluation criterion for the nonmetric situation. The insights provided by this framework lead us to introduce the notion of a generic nonmetric NNS algorithm in Section 2.2, that, in turn, leads us to refine our evaluation criterion in the Section 2.3. Based on this refined criterion, this section ends with the discussion in Section 2.4 of the termination criterion of a generic nonmetric NNS algorithm.

[1] Various relaxations of the triangular inequality have been proposed (see [6] for more details). Nevertheless those relaxations are far too restrictive to cover functions such as the Smith-Waterman distance.

[2] To our knowledge, the nonmetric NNS problem has been studied once by Zhang and Srihari [25] (2002), but restricted to a particular nonmetric similarity function. Additionally, contrary to the opinions presented in this paper, we will show in Section 4 that a classical metric NNS algorithm can be well suited to the nonmetric case.

2.1 Approximate queries in nonmetric space

As discussed here above, the query methods in the nonmetric situation are either naive or approximate. The most commonly used approximation criterion in the NNS literature is the *geometric* approximation[3]: the algorithm ensures that the distance from the query point to the returned point is smaller than $1+\epsilon$ times the distance of the query point its nearest neighbor (see [16, 10, 5] as recent examples).

Nevertheless, as pointed out in [4], such geometric criterion suffers from intrinsic instabilities as the dimension increase. As a simple example to illustrate this point, let us a consider points uniformly drawn from the unit cube in $\mathbb{R}^k$. When the dimension k tends to infinity, the normalized random distance between points $AVG[d/\sqrt{k}]$ tends to a constant. Strong empirical evidence of this phenomenon can also be found in the NNS literature. For example the recent benchmark [16] shows that taking $\epsilon = 0.1$ leads to more than 50% CPU costs saving on non-approximate methods by simply sampling the initial datasets. Additionally, the values provided by nonmetric similarity functions (Smith-Waterman [20] being a typical example) have little meaning in themselves. The only purpose of those similarity functions is to provide a *ranking* criterion, i.e. to indicate if a pair of points can be considered as closer than another pair. Considering this, one would expect that the NNS algorithm execution and its associated performance, at least in nonmetric case, to be *invariant against monotone transformations*[4], i.e. the substitution of $\phi(s)$ to the similarity function s ($\phi : \mathbb{R}_+ \to \mathbb{R}_+$ being an arbitrary increasing function) should have no impact on the algorithm. All those elements call for *a rank-based rather than geometric-based approximation criterion* for the nonmetric NNS problem.

Following the discussion here above, we now introduce the notion of **accuracy** as a rank-based evaluation criterion of nonmetric NNS algorithms. To our knowledge, we are not aware of previous use of such a criterion for the purpose of NNS. Formally, let (E, s) be the considered nonmetric space. Let $\widetilde{\mathcal{B}}(q, r)$ (resp. $\widetilde{\mathcal{N}}(q, k)$) be the approximate result provided by a given NNS algorithm to the range query $\mathcal{B}(q, r)$ (to the near-neighbor query $\mathcal{N}(q, k)$).

Definition 2 (Accuracy of a NNS query) *For a given near-neighbor query (q, k), we define the accuracy of the result returned by an arbitrary NNS algorithm with*

$$v(\widetilde{\mathcal{N}}(q,k)) = \frac{|\widetilde{\mathcal{N}}(q,k) \bigcap \mathcal{N}(q,k)|}{|\mathcal{N}(q,k)|} = \frac{1}{k}\left|\widetilde{\mathcal{N}}(q,k) \bigcap \mathcal{N}(q,k)\right|$$

The accuracy value can be interpreted as the ratio of correct query points (true positive) returned by the algorithm for the near-neighbor query (q, k).

[3] This criterion has been declined in various ways such as the *effective distance error* defined as $E = \frac{1}{|X|}\sum_{x \in X} \frac{\widehat{d}}{d^*} - 1$ [3, 11]. Although, the declinations of the geometric may be expressed in many different ways, they all share the fact than they rely intrinsically on the *absolute* distance values between the points.

[4] Indeed both range and near-neighbor queries are invariant against monotone transformation of the similarity function.

Assuming that the algorithm does not return more than k neighbor to a such a query, the range of accuracy value is the interval $[0, 1]$, *zero* being a total wrong answer, *one* being the exact answer. Note that the accuracy as presently defined answers to the issues discussed here above. In particular, it is clear that the accuracy is invariant to monotone transformation of the similarity function. The accuracy is straightforward to transpose in the range query case; nevertheless, since the range query does not constrain the returned number of points, the false positive ratio (dual of the accuracy interpreted a true positive ratio) would be required to evaluate the answer quality. In the following, in particular in the experiments presented in Section 4, we will focus on the near-neighbor queries that are easier to interpret.

Although being very simple and invariant to monotone similarity transformation, the accuracy is not entirely satisfying as a quality criterion. Indeed, the accuracy is not sensitive to the quality of the incorrect points. Intuitively, returning very close but incorrect neighbors will not lead to a higher accuracy than returning very distant neighbors. This issue will be solved in Section 2.3, but for the sake of simplicity we will first introduce a generic nonmetric NNS algorithm template.

2.2 Generic nonmetric NNS algorithm template

We introduce here a generic NNS algorithm template. Indeed, the nonmetric NNS problem is conceptually a *simpler* problem than its correspondent metric one because of the lack of assumptions. The purpose of this template is to outline the key aspects of nonmetric NNS.

Intuitively, any nonmetric NNS algorithm $\mathcal{A}$ relies on a particular data structure that contains X the dataset. When performing a query with $\mathcal{A}$, one can distinguish two parts in X: the set of explored points (the similarity between any of those points and the query has been computed) and the set of unexplored points. Based on the similarity values associated to the set of explored points, the algorithm $\mathcal{A}$ chooses which point is explored next. Some data structure $\mathcal{G}$ usually restraints the set of candidate points. In other words, all unexplored points are not necessary candidates to be explored next. Notice that, here, the term "exploration" refers to process of choosing the next point to be compared to the query point based on the information previously acquired.

More formally, let $\mathcal{G} = (V, \mathcal{E})$ be an oriented graph, $\preceq_{\mathcal{K}}$ be a ranking function and $r \in V$ be a root vertex. As detailed in the pseudo-code of the Algorithm 1, the tuple $(\mathcal{G}, \preceq_{\mathcal{K}}, r)$ is a generic representation of the algorithm $\mathcal{A}$. The set V (respectively $\mathcal{E} \subset V \times V$) represents the vertices (respectively the edges) of the graph $\mathcal{G}$[5]. For any subset $C \subset V$, we refer to $\mathcal{N}(C)$ as the "neighbors" of C, formally defined by

$$\mathcal{N}(C) = \{y \in V | (x, y) \in \mathcal{E} \text{ and } x \in C\}$$

[5]We adopt the notation V for the vertices to be more consistent with the graph literature, but please note that here $V = X$.

Let $\mathcal{K}$ be a set of pairs $\{x, s\}$ where $x \in X$ is a point and $s \in \mathbb{R}_+$ is a similarity value. The set $\mathcal{K}$ is an abstraction of the set of explored points. The sign $\preceq_{\mathcal{K}}$ refers to the relation order induced by $\mathcal{A}$ on the set C of candidate points for exploration. The vertex r represents the "root" of the algorithm $\mathcal{A}$, i.e. the point that will be compared first to the query. Using those notations, left us now provide the detail of the general representation of the algorithm $\mathcal{A}$ with the following pseudo-code.

Algorithm 1 NonMetricQuery(q)

1: $\mathcal{K} \leftarrow \{r, s(q, r)\}$
2: **while** SomeTerminationConditions **do**
3: $\quad C \leftarrow \mathcal{N}(\mathcal{K}) \setminus \mathcal{K}$
4: $\quad v \leftarrow \min_{\preceq_{\mathcal{K}}} C$
5: $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{v, s(q, v)\}$
6: **end while**
7: Return $\mathcal{K}$

The Algorithm 1 is quite simple. It begins with the exploration of the root at line 1. The main loop (lines 2-6) consists of finding the set C of candidate vertices at line 3, of choosing within C the vertex v to be explored next at line 4 and finally to explore v at line 5. The algorithm returns directly $\mathcal{K}$ the set of explored vertices. Note that the constraints induced by the model underlying the Algorithm 1 are very reasonable ones. This model implies that no similarities are computed except the ones between the query point and the dataset points (forbidding the computation of the similarity between two points of the dataset for example). This model also implies that each allowed similarity computed once at most. This template, although not providing any actual implementation, outlines the following aspects of nonmetric NNS:

- Heuristic stopping criterion.
- Accurate exploration is the key of nonmetric NNS.
- No difference between range query and near-neighbor query.

As discussed here above, the only exacts nonmetric NNS algorithm is actually the exhaustive search through the whole dataset. Therefore, an heuristic stopping criterion (as expressed line 2 of the Algorithm 1) is required. Since any NNS algorithm $\mathcal{A}$ requires to keep $\mathcal{K}$ the set of already encountered vertices (or at least a subset of $\mathcal{K}$ associated to the closest points), an algorithm $\mathcal{A}$ can be interrupted at any time, the set $\mathcal{K}$ being returned. Intuitively the more exploration is allowed, the more accurate will be the results extracted from $\mathcal{K}$. This point will be the object of Section 2.3 because it provides an new insight on NNS algorithms performance evaluation; but also of the Section 2.4 that discusses what termination criterion can be used in practice.

This initial remark leads us to the second aspect: an accurate exploration is the key of nonmetric NNS. Indeed, since the algorithm $\mathcal{A}$ has be stopped before

having actually explored the whole dataset, it is critical, for the accuracy (see Definition 2) of the results, to explore as fast as possible the nearest points of the query. Therefore the main issue in nonmetric NNS is not to design elaborate pruning criterions (as opposed to the metric case), but is to design a ranking $\preceq_{\mathcal{K}}$ that will explore as fast as possible to the nearest neighbors of the query. This point is the main motivation underlying the densitrees introduced in Section 3.

Finally, the nonmetric NNS is conceptually more simple than the metric NNS. Indeed, the lack of "hard constraints" like triangular inequality implies that there is no practical difference between a range query and a near-neighbor query. It is known for the metric case, that there are tight relationships between the two query types. General methods are available to convert efficiently a range query into near-neighbor query and vice-versa (see [6] for the detail). But in the nonmetric situation, the relationships are even stronger. The process described in Algorithm 1 remains the same independently of the query type; only the query point matters. In both cases, the result will be a subset of the returned set $\mathcal{K}$. As a consequence, we have restricted the experiments of the Section 4 to near-neighbor queries.

2.3 Query accuracy profile

This section introduces the notion of *query accuracy profile* that is an extension of the "accuracy" criterion introduced in Section 2.1. As we have seen, the accuracy indicates the ratio of correct items returned by a certain algorithm $\mathcal{A}$ for a given query. The Algorithm 1 provides a new insight to the notion of accuracy. Indeed it becomes clear that the accuracy value is bound to the termination condition (see line 2) of the Algorithm $\mathcal{A}$. Nevertheless, as discussed previously, the termination condition is a simple tradeoff, whereas the exploration is the critical issue of nonmetric NNS. Those elements call for *a criterion that could be used to evaluate the quality of exploration rather than the termination condition.*

Intuitively, since the Algorithm 1 can be stopped after each call to the similarity function, a corresponding accuracy can also be computed. The query accuracy profile simply gathers all successive accuracies. More formally, the query accuracy profile is a function $p : \#call \mapsto accuracy$ ($\#call$ is the number of calls) where $p(k)$ is equal to the accuracy of the best subset of $\mathcal{K}$ (taking the notations of the Algorithm 1) after the k^{th} call to the similarity function (the similarity function is called iteratively at line 5 in Algorithm 1). If $\mathcal{N}$ is the result from the exact computation of the query and $\mathcal{K}$ the set of encountered points after the k^{th} call to the similarity function, then $p(k) = \frac{|\mathcal{K} \cap \mathcal{N}|}{|\mathcal{N}|}$.

Since the query accuracy profiles are the core of the empirical results presented in Section 4, let us now discuss how query accuracy profiles can be efficiently computed in practice. The main issue is the fact that the profile computation actually requires the exact query result. Therefore, the most simple solution consists of a two-passes method: first compute the query result using an exhaustive exploration; second, perform the exploration and record at each similarity function call the reached accuracy. When correctly implemented, this method requires $2n$ similarity calls which is the best asymptotical bound that

can be achieved here (note that the *exact* query result is required for the accuracy computation). Nevertheless, this method can be easily be improved into a single-pass method. The detail is given by the pseudo-code of the Algorithm 2 which is a slightly modified version of the Algorithm 1.

Algorithm 2 ProfileQuery(q)

1: $\mathcal{K} \leftarrow \{r, s(q,r)\}$
2: $i \leftarrow 0$
3: $\mathcal{L} \leftarrow \{r, s(q,r), i\}$
4: **while** $\mathcal{N}(\mathcal{K}) \setminus \mathcal{K} \neq \emptyset$ **do**
5: $\quad C \leftarrow \mathcal{N}(\mathcal{K}) \setminus \mathcal{K}$
6: $\quad v \leftarrow \min_{\preceq_{\mathcal{K}}} C$
7: $\quad \mathcal{K} \leftarrow \mathcal{K} \cup \{v, s(q,v)\}$
8: $\quad i \leftarrow i + 1$
9: $\quad \mathcal{L} \leftarrow \{v, s(q,v), i\}$
10: **end while**
11: Return ExtractProfile($\mathcal{L}$)

Note that the termination condition of the Algorithm 2 is equivalent to "until there are no more unexplored vertices". Additionally, a list $\mathcal{L}$ gathers the tuples $(v, s(q,v), i)$ added at each iteration (where i counts the number of previous calls to the similarity function). The profile is extracted as post-query process by calling ExtractProfile whose detail is not given here because being quite straightforward.

2.4 Exploration stopping criterion

The Algorithm 1 refers to an abstract SomeTerminationConditions criterion to stop the exploration and return $\mathcal{K}$. Intuitively, for a given exploration method, there are two opposing forces for the client of the NNS algorithm **CPU cost vs. result accuracy**. In such situation, the most flexible solution is simply to provide the tradeoff curve $u : \mathit{accuracy} \mapsto \#\mathit{call}$ to the client and let him chose the desired level of accuracy. Actually, we have already introduced, although from a different perspective, such a tradeoff curve through the concept of "query accuracy profile" in Section 2.3.

A empirical way of constructing the tradeoff curve u is to take a random sample of points from the dataset, to run the Algorithm 2 with those points being the queries, and to compute the average query accuracy profile. From such average query profile, the reverse function (providing the number of similarity calls based on a chosen accuracy) can easily be computed.

3 Densitrees

As a consequence of the insights on nonmetric NNS this section introduces a class of particular data structures that we call *densitrees*. Intuitively, the core

insight underlying the densitrees is to rely on classifiers to efficiently guide the exploration process.

The densitrees along with other concepts such as *density estimator* are defined in Section 3.1. The densitree query algorithm is given in Section 3.2, followed by the densitree construction algorithm in Section 3.3. This part ends with an extensive discussion of the density estimator in Section 3.4 that proves to be a critical component of the densitrees.

3.1 Definition of densitrees

Intuitively, densitrees are a class of decorated trees that hold the points of the dataset in a way similar to the metric tree [21, 19, 7]. The critical difference lies in the nature of tree decoration; instead of having one or several real values reflecting some bounds on the triangular inequality attached to every tree node, each densitree node is associated to a particular classifier called here a *density estimator*[6].

A density estimator estimates the number of points contained within a ball specified by its center and radius. When the query is performed, the densitree exploration, which starts at the root, follows greedily the paths of greatest densities. The insights offered by the Algorithm 1 motivates the idea of exploring the nearest points as fast as possible because it will lead to a better accuracy for any given query termination condition.

Formally, nodes and leaves in the densitree are treated indifferently. A densitree node is a tuple defined by $\alpha = (x, \widehat{\phi}, \alpha_L, \alpha_R)$ where $x \in X$ is point, $\widehat{\phi}$ is a density estimator, α_L and α_R are respectively left and right nodes (that may be null). By convention, the root node of the densitree is refered as α_0. The density estimator $\widehat{\phi}$ is a function $\widehat{\phi} : E \times \mathbb{R} \rightarrow [0; 1]$.

In order to clarify the need for density estimator, let us first introduce the more simple notion of density function. We define the (exact) density function for a dataset X with

$$\phi_X(q, r) = \frac{1}{|X|} |X \bigcap \mathcal{B}(q, r)|$$

The function ϕ_X associates the ratio of the dataset contained in the specified ball. Since X always represents the dataset, in the following, the density function will be refered as ϕ for the sake of simplicity. From an exploration perspective, the density function ϕ is obviously the optimal function to guide the exploration. Nevertheless the density computation in term of call to the similarity function prove to be as expensive as the naive exhaustive search. In this respect, the density estimators, that can be viewed as efficient approximation of the density function, are be used instead.

[6] We speak of densitrees (plural) rather than densitree (singular) because in practice many variations based on the density estimator definition are possible within the densitrees framework. This point will be the object of the Section 3.4.

3.2 Queries in densitrees

For the sake of clarity, we begin our densitree discussion by introducing the densitree query algorithm. Based on the generic template introduced previously (see the Algorithm 1), the Algorithm 3 provides the detail of the behavior of the densitree query process[7].

Algorithm 3 DensitreeQuery(q)

1: $\mathcal{K} \leftarrow$ EmptyStack()
2: $r \leftarrow$ CurrentRange($\mathcal{K}$)
3: $\mathcal{K}$.Push($\alpha_0, \phi_{\alpha_0}(q, r)$)
4: **while** SomeTerminationConditions **do**
5: $\quad \alpha \leftarrow \mathcal{K}$.Pop()
6: $\quad r \leftarrow$ CurrentRange($\mathcal{K}$)
7: $\quad$ **if** $left(\alpha) \neq null$ **then** $\mathcal{K}$.Push($left(\alpha), \phi_{left(\alpha)}(q, r)$)
8: $\quad$ **if** $right(\alpha) \neq null$ **then** $\mathcal{K}$.Push($right(\alpha), \phi_{right(\alpha)}(q, r)$)
9: **end while**
10: Return $\mathcal{K}$

Let us give some details about the pseudo-code of the Algorithm 3. The notation ϕ_α refers to the density estimator associated to the node α. The left branch (resp. the right branch) of the node α is refered by $left(\alpha)$ (resp. $right(\alpha)$). We assume that the set of explored points $\mathcal{K}$ is designed as a heap, the points being ordered by decreasing estimated densities. The density estimations are based on the ball radius provided by the method CurrentRange that takes $\mathcal{K}$ as argument. Like we did for the Algorithm 1, the termination criterion at line 4 is left unspecified (see Section 2.4 for a discussion of this issue).

The query process in densitrees is conceptually rather simple, but several issues are left undiscussed. The most critical one is the definition of the density estimator which will be extensively discussed in Section 3.4. Let us now provide some details about the CurrentRange function.

Estimating the final range. In the proposed definition for the density estimators (see Section 3.1), both the query point and a radius are required to estimate the density. In the case of a range query, the range value r is given, and CurrentRange is trivially defined as a constant function returning r. The difficulty arises from the near-neighbor query case where there are no obvious ways to define CurrentRange. Here below, we first describe the approach typically adopted in the metric NNS algorithm to solve a very similar problem and we explain why this approach leads to poor performance in the nonmetric context. In order to overcome this issue we introduce a solution that will be empirically evaluated in Section 4.

[7]For reasons that have been discussed in Section 2.2, we do not distinguish here range queries from near-neighbor queries.

The problem of the definition of the CurrentRange function is somewhat similar to the notion of "current radius" is classical metric NNS algorithms such as the metric tree [21, 19, 7] and the vantage point tree [22, 23, 24] (among many other, see [6] for a survey). The usual approach suggest to return an infinite radius until $|\mathcal{K}| = k$ (k being the number of neighbor specified by the query), and then to return the radius associated the k^{th} nearest neighbor in $\mathcal{K}$. If the metric assumptions hold, this approach typically guarantees the correctness of the algorithm ensuring an exact result. Unfortunately, strong empirical evidence (yet not included in Section 4 for the sake of concision) suggests that this approach is inefficient in the nonmetric context of densitrees. Intuitively, since $\phi_\alpha(x, \infty)$ is always equal to 1, an initial infinite radius implies a random exploration of the k first points[8] which leads to poor overall performance.

This calls for a deeper analysis of the Algorithm 3 in its nonmetric context. Since the exploration requires, to be efficient, the minimization of the bias of the density estimator toward the (exact) density function, the function CurrentRange should ideally return the *final radius* (being defined as the distance of k^{th} closest point of the query). The approach used in the experiments of Section 4 is a simple heuristic that consists of performing a sample of random queries (just after the densitree construction, see Section 3.3) for a given number of neighbors and of storing the average achieved final range. This approach has the advantage of requiring no additional metric call at query-time. Empirical evidence indicates that this approach lead to strong improvements over the decreasing radius approach presented here above. Nevertheless, this approach is naive, and we expect further research to provide better solutions to estimate the final range.

3.3 Building the densitrees

As stated in Section 3.1, the densitree is a binary tree decorated with density estimators. Here below, we propose a very classical top-down approach to build the densitree that involves two key tasks: (i) a recursive dataset partitioning method, (ii) a density estimator learning method. The point (i) is discussed through the pseudo-code of the Algorithm 4. For being more complicated, the point (ii) is left to the Section 3.4.

Algorithm 4 BuildDensitree(X)

1: $\phi \leftarrow$ LearnDensity(X)
2: $(X_L, X_R) \leftarrow$ Split(X)
3: $\alpha_L \leftarrow$ BuildDensitree(X_L)
4: $\alpha_R \leftarrow$ BuildDensitree(X_R)
5: **Return** $(\phi, \alpha_L, \alpha_R)$

The Algorithm 4 has a simple recursive structure. For the sake of simplicity,

[8] Empirical evidence suggests that the radius remains largely inaccurate for a much longer time.

the Algorithm 4 does not include the detail of the trivial stopping criterions. The function SPLIT, as the name suggests, splits the dataset into a left and a right part (respectively X_L and X_R following the notations of the pseudo-code). The detail of the SPLIT function is given here below. The discussion of the function LEARNDENSITY is left to the next section.

Let us note that this top-down approach is almost identical to the classical metric tree or vantage point tree (or spill-tree [16] for a more recent example) construction algorithms, except that density estimators should be additionally learned in order to decorate the tree. Many heuristics have been proposed in metric NNS literature to find a good split function (see [6] for a review). An interesting point to note is that the metric assumptions have little importance here since the split heuristics do not rely explicitly on them. In the experiments of the Section 4, we have adopted for SPLIT a classical heuristic[9] that computes an approximation of two most distant points (the pivots) with linear CPU requirement and then splits the remaining points based on their nearest pivot.

3.4 Density estimators

The purpose of the density estimators is to efficiently approximate the density function (see Section 3.1 for density estimator motivation). This implies that two main opposing forces apply on density estimators: (i) the density estimator must be as accurate as possible, (ii) the density estimator must have a computational cost as low as possible.

Based on the the insights provided by Algorithm 1, it can be noticed that the *absolute* value returned by the density estimators no importance in itself since only the *ranking* matters. Therefore, there is no necessity to estimate the density *directly* if the ranking can be approximated through other means. In this respect, the most naive approach to guide the densitree exploration simply consists of aiming for the nearest point. Formally, this idea takes the form of a density estimator $\widehat{\phi} : E \times \mathbb{R} \to \mathbb{R}$ defined by $\widehat{\phi}(x, r) = -s(x, q)$ where q is the query point (the *minus* sign is required because highest densities are explored first). In the following, we will call this classifier the *greedy* density estimator[10]. Although being naive, this solution leads to efficient results in practice (see Section 4 for more details).

Let us now introduce a more complicated density estimator, called PATRIOT[11], that corresponds to the initial insight of approximating the density function. We will begin with an intuitive description of the PATRIOT. The densitree is, as we have seen, a tree where each node is decorated by a density estimator. When a node is explored, the corresponding the density estimator is called. First, let us see what information (i.e. computed similarity values)

[9] Take a random point a. Compute b the most distant point of a. Compute c the most distant point of b. Return the pair (b, c) as an approximation of the two most distant points.

[10] A densitree based on the greedy density estimator is nothing more than a metric tree that skips, at query-time, the triangular inequality pruning.

[11] PAth To Root densIty estimaTOr

can be used for the purpose of the density estimator. Since the computation of the similarity value between the query point is required for the node to be explored (see Algorithm 3), we can assume this value to be available to the density estimator[12]. Additionally, the exploration process ensures that all the nodes, along the path that leads from the root to the current node, have already been compared to the query point. Therefore, we propose a density estimator that exploits all similarity values along the path to the root[13].

As a second part of the intuitive description of the PATRIOT, let us give some insights on the inner workings of the density estimation. As preliminary remark, let us note that any point a can be projected[14] into a real-valued vector against a set of points (the dimensions being the similarity values between the point a and the points of the set). The PATRIOT includes a "projection" of the points associated to the node subtree. The projection is performed against the path-to-root path described here above. The PATRIOT estimates the density of a ball centered around the query point using this sample of projected points. The main advantage of the PATRIOT approach is the exploitation of a path-to-root vector of similarity values (rather than the single current-node-to-query similarity value) without requiring any additional to call to the similarity function neither at construction-time nor at query-time.

We will now provide a more formal definition of the PATRIOT. Let α be the node of interest (associated to the considered PATRIOT). Let $X_\alpha \subset X$ be the subset of points associated to the subtree defined by the node α. Let B be the list of points $B = (x_1, \ldots, x_d) \in X^d$ that constitutes the path to the root (x_1 is the root point and x_d the point associated to the node α). Any point $x \in X$, can be "projected" on B with $\bar{x} = (s(x, x_1), \ldots, s(s, x_d))$. Let $\bar{X}_\alpha$ be the point-wise projection of X_α over B. Let $\| \, . \, \|$ be a norm over $\mathbb{R}^d$ (more details will be given later about this norm). Let $\kappa : \mathbb{R} \to \mathbb{R}$ be a real function; κ is the "scaling function" (more details will be given later about this function). We have now gathered all the elements required to define the PATRIOT $\widehat{\phi}$. For a point $q \in E$ and a range $r \in \mathbb{R}$, we have

$$\widehat{\phi}(q, r) = \frac{1}{|X_\alpha|} \left| \mathcal{B}_{\|.\|, X_\alpha}(\bar{q}, \kappa(r)) \right| \tag{1}$$

Let us give a word of explanation about this definition. The notation $\mathcal{B}_{\|.\|, X_\alpha}$ refers the the ball defined over the set X_α based of the norm $\| \, . \, \|$ (used as the similarity function). Intuitively the query point q is simply projected as $\bar{q}$, then the density is evaluated within the projected space which required to scale the range r into $\kappa(r)$ as well. The value of $\widehat{\phi}$ can be considered as an estimation

[12] If we try to rely solely on the node-to-query similarity value to infer a density estimation, then we are likely obtain an estimator pretty much equivalent to the greedy density estimator.

[13] The path-to-the-root is the only set of points guaranteed to be already explored when the node is reached by the Algorithm 3. Additional similarity values may nevertheless by available when the node is explored; and more complicated methods can possibly be devised to exploit such uncertain information.

[14] Such projection is a well known technique in the metric NNS literature, see [12] as an example.

Figure 1: PATRIOT list of issues

$$\text{The proposed formula is } \widehat{\phi}(q,r) = \frac{1}{|X_\alpha|}\left|\mathcal{B}_{\|.\|,X_\alpha}(\bar{q},\kappa(r))\right|$$

- How to choose $\|\ .\ \|$?
- How to choose κ?
- CPU and memory costs of $\widehat{\phi}$ at construction-time?
- Idem at query-time?
- The empirical accuracy of $\widehat{\phi}$ vs other estimators?

of the density ϕ. The PATRIOT method raises several practical issues (see the Figure 3.4) that will be discussed in the following.
The definition of the norm $\|\ .\ \|$ is critical for the accuracy of of PATRIOT estimator $\widehat{\phi}$. A natural idea would be use $\|\ .\ \|_1$ or $\|\ .\ \|_2$ as very classical metrics over $\mathbb{R}^d$. Those metrics have been empirically evaluated (the experiments are not listed in Section 4 for the sake of concision) and lead unfortunately to very poor accuracies. Let us provide the insight of this poor behavior. To simplify largely the problem, we can say that the exploration (if not stopped by any termination criterion) has three phases: (i) descending down the densitree to the query close neighborhood (ii) visiting the query close neighborhood (iii) visiting the remaining remote neighborhoods. In practice, the average similarity values between the query point and the points explored the phase (i) are much greater than the similarity values between the query point and the points explored during the phase (ii). As we have designed PATRIOT, this phenomenon leads to a considerable overweighting of the points at the top of the densitree (points close to the densitree root). This issue suggests that more complicated norms for $\|\ .\ \|$ should be considered. A natural solution to overcome the overweighting mentioned here above is to introduce linear coefficients such as

$$\|\ \bar{x}\ \| = \sum_{i=1}^{d} \lambda_i |\bar{x}_i|$$

This option raises an other issue that is the actual choice of the coefficients λ_i. In Section 4, we have opted to normalize each dimension dividing by the mean similarity value. Thus we ensure that $\mathrm{E}[\lambda_i|\bar{x}_i|] = 1$ for all i. This choice is motivated by the insight provided here above and suppress the overweighting problem. In practice, this option leads to large improvements over the use of uniform norms like $\|\ .\ \|_1$. Nevertheless, this option is certainly not definitive, and we suspect that improvements can be obtained by a better "tuning" of this norm. In particular, empirical observations lead us to suggest that the closer the pivot point is from the query point, the more accurate is the prediction

associated to the resulting dimension. Better approaches would certainly take into account such phenomenon. A more general would approach would even suggest that the $\{\lambda_i\}_i$ coefficients should be directly optimized (learned) in order to fit the proposed purpose. Such approach goes beyond the scope of the present study.

The values of the scaling function κ is tightly bound to the actual metric $\| \ . \ \|$. Nevertheless we propose a generic method to estimate the scaling function κ that can be used along any arbitrary metric $\| \ . \ \|$. The method consists simply in drawing random pair of points (x, y) within X_α and computing the pair of values $(s(x, y), \| \ \bar{x} - \bar{y} \ \|)$. Let $(u_i, v_i)_i$ be a list of such sample pairs ordered against the similarity values (ie. $i \leq j \Rightarrow u_i \leq u_j$). Considering the list $(u_i, v_i)_i$, a simple definition for κ would be $\kappa(u) = v_{\arg\max_i \{u_i < u\}}$. Such choice would be motivated by the intuition that choosing the norm value associated to a similar similarity value would be an accurate option. Unfortunately, empirical observations suggest that such an option leads to a very poor overall accuracy for the PATRIOT estimator. Indeed, the use of a single pair does not provide an accurate estimate. The empirical solution to overcome this issue that have been used in the Section 4 is to smooth the previously proposed κ function by clustering the list $(u_i, v_i)_i$ (gathering the values based on the u_i similarity, and taking the mean of the u_i and v_i values for each cluster). In the Section 4, the number of random pairs is fixed to $|X_\alpha|$ and the resulting list is clustered into $\sqrt{|X_\alpha|}$ (in order to assure an unbiased yet convergent estimator for κ).

A efficient implementation for $\widehat{\phi}$ is required to ensure low CPU and memory cost for the densitrees both at construction and query-time. Let us begin our analysis at construction-time. The top-down densitree construction algorithm (see Algorithm 4) ensures that the projection of each subtree (noted X_α in Equation 1) requires not additional call to the nonmetric similarity function. Indeed, all subtree points have been compared to each node along the path to the root (see the discussion of the Split function in Section 3.3). On this projected subtree, the PATRIOT method induces an additional range-query problem. Although our overall objective is to perform a nonmetric NNS, it can be noticed that the task performed by a PATRIOT, is actually a *metric* range query. Therefore, most of the solutions proposed in the NNS literature can be used here. In Section 4, we have opted for the metric tree algorithm. The additional CPU cost as construction-time for the whole densitree is null in term of calls to the similarity function. In term of raw CPU operations, the additional cost for learning the PATRIOTs is $\mathcal{O}(n \ln(n)^2)$ (n being the number of points) which asymptotically outgrows the base CPU cost of $\mathcal{O}(n \ln(n))$ of Algorithm 4. In practice this additional cost may be preponderant, if the similarity function CPU costs are negligible, or negligible if the similarity function is expensive. In this study, we focus on the later case. In term of memory, the additional cost of storing the density estimators is a $\mathcal{O}(n \ln(n))$ with small constants in practice. Once again, the interpretation of this cost should take into account the context of the nonmetric NNS. Indeed, the dataset although theoretically requiring $\mathcal{O}(n)$ of memory may very-well be predominant (typical nonmetric

situations deal with images or protein sequences). The discussion of the CPU costs of the PATRIOT that involves many elements such as the resulting query accuracy profiles is to the Section 4.

The accuracy of the PATRIOT estimator determines the overall exploration quality of the densitree. The PATRIOT estimator aims to a better accuracy than the greedy estimator previously described. Intuitively, the PATRIOT estimator exploits more information than the greedy estimator (several similarity values are used to "score" a node vs a single value). The drawback of this approach is that the PATRIOT estimator is less "robust" (from a statistical viewpoint) than the greedy estimator. Empirical observations (yet not included in the Section 4 for the sake of concision) indicates that the PATRIOT estimator is less accurate than the greedy estimator when the number of points contained in the node subtree is low. Therefore the ranking $\preceq_K$ (using the notation of Algorithm 1) used in the experiments of Section 4 is a hybrid method that mixes greedy estimators and PATRIOTs. The choice is of the classifier to be used is determined as follow: if both subtree sizes are greater than an given threshold then the comparison is based on the PATRIOT estimates; if not, the comparison is based on the respective node similarity values to the query point, i.e. the greedy estimator. The threshold used in the experiments in Section 4 is $\sqrt{n}$ where $n = |X|$ is the number of points in the dataset. This threshold is mostly arbitrary (although it seems to perform reasonably well, see the results of Section 4), nevertheless it ensures that the PATRIOT variance and bias tend to zero when n tends to infinity.

4 Experiments

The objective of this section is to empirically evaluate the quality of the exploration strategy of the various algorithms that have been introduced so far. In particular we provide experimental results based on the datasets already used in [16] for a benchmark of metric near neighbor search algorithms, plus a biological sequence dataset associated to a nonmetric similarity function. Let us first review those datasets and the corresponding similarity functions.

- `aerial`: Texture feature data contain 275.000 feature vectors of 60 dimensions representing texture information of aerial photograph [17, 18, 15]. The similarity function associated to this dataset is the $\mathcal{L}_1$ norm.
- `corel_hist`: 20.000 histograms with 44 non-zero dimensions of color images taken from the COREL STOCK PHOTO Library [16, 12, 1, 15]). The similarity function associated to this dataset is the $\mathcal{L}_1$ norm.
- `corel_uci`: 68.000 histograms with 64 dimensions of color images from the COREL library [16, 15]. The similarity function associated to this dataset is the $\mathcal{L}_1$ norm.

- `disk_trace`: 40.000 content traces of disk-write operations, each being a 1kb block. The traces are generated from a desktop computer running SuSe Linux during daily operation [16, 15]. The similarity function associated to this dataset is the $\mathcal{L}_1$ norm.

- `sprot40`: 101601 sequences extracted from the Swiss-Prot database version 40 [2]. The similarity function associated to this dataset is the Smith-Waterman distance [20]. The Smith-Waterman distance does not verify the metric assumptions.

Please note that the only true nonmetric dataset is `sprot40`. Nevertheless, since the nonmetric NNS is a generalization of the metric one, it appeared reasonable to us to include those datasets (which have already been used in NNS benchmark contrary to `sprot40`). Those datasets have been tested against the following three algorithms.

- `M-Tree`: the metric tree structure [21, 7]. An efficient implementation can be found at [8].

- `Greedy Densitree`: the greedy densitree (associated to the greedy density estimator as defined in Section 3.4).

- `PATRIOT Densitree`: the PATRIOT densitree (associated to the PATRIOT density estimator as defined in Section 3.4).

The empirical results, presented in Figure 2 to 6, reflect the query accuracy profile criterion introduced in Section 2.3. Those results are the average query accuracy profiles obtained for each algorithm against each dataset with 1000 random queries (only 100 queries for the `sprot40` dataset). The number of neighbors k varies from $k = 2$ (minimal neighborhood[15]) to $k = 1024$ (large neighborhood). We have excluded the M-Tree from the `sprot40` experiment because of the reliance on the metric assumptions.

Several empirical conclusions can be drawn from the empirical results of this section. Surprisingly, our most important conclusion is quite counter-intuitive: *the triangular inequality has only a secondary contribution to metric NNS efficiency.* Indeed, if metric tree uniformly outperforms the greedy densitree, the improvement brought by triangular inequality pruning is only a secondary effect on the exploration quality compared to greedy estimator effect. In term of exploration improvement over the (naive) random exploration, the impact of the greedy estimator is clearly the primary factor.

Our second empirical conclusion is about the densitrees performance. Both greedy and PATRIOTperform reasonably well against the metric dataset. The PATRIOT densitree uniformly outperform the greedy densitree and is often very

[15]Since the sample queries were drawn from the same dataset that has been used to build the NNS data structures, the results for $k = 1$ are highly biased in favor of the NNS algorithms. In such situation, the profiles for $k = 2$ could be interpreted as a good approximation of the *nearest neighbor search* profile for a foreign query point (not previously used to build the NNS data structure).

Figure 2: `aerial` dataset

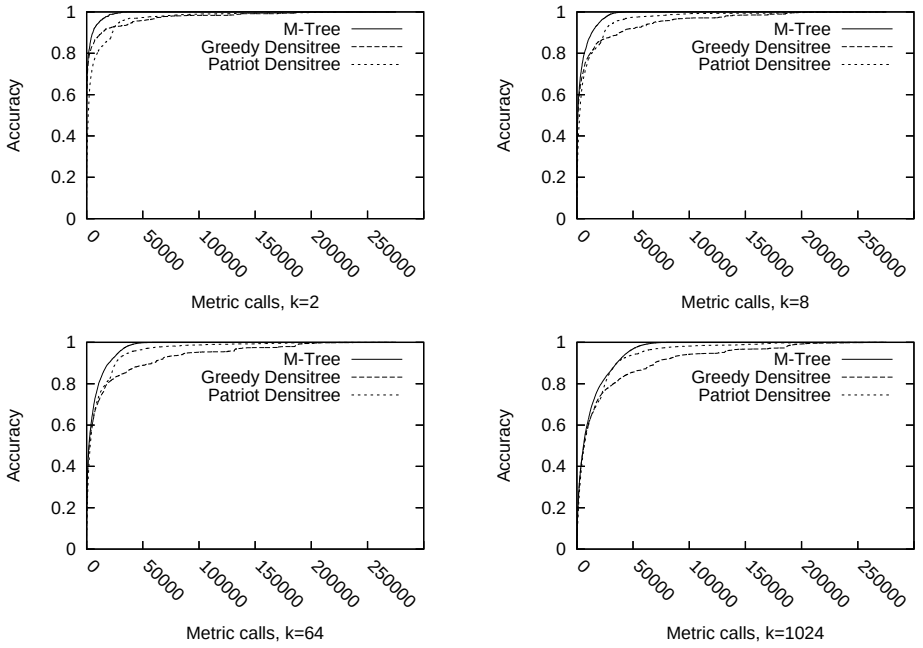


Figure 3: `corel_hist` dataset

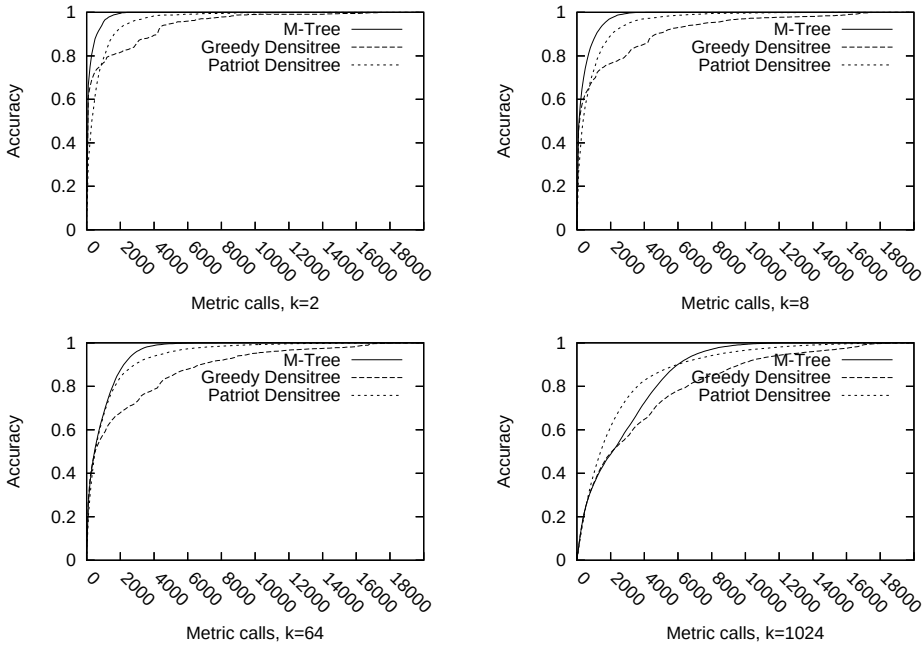


Figure 4: `corel_uci` dataset

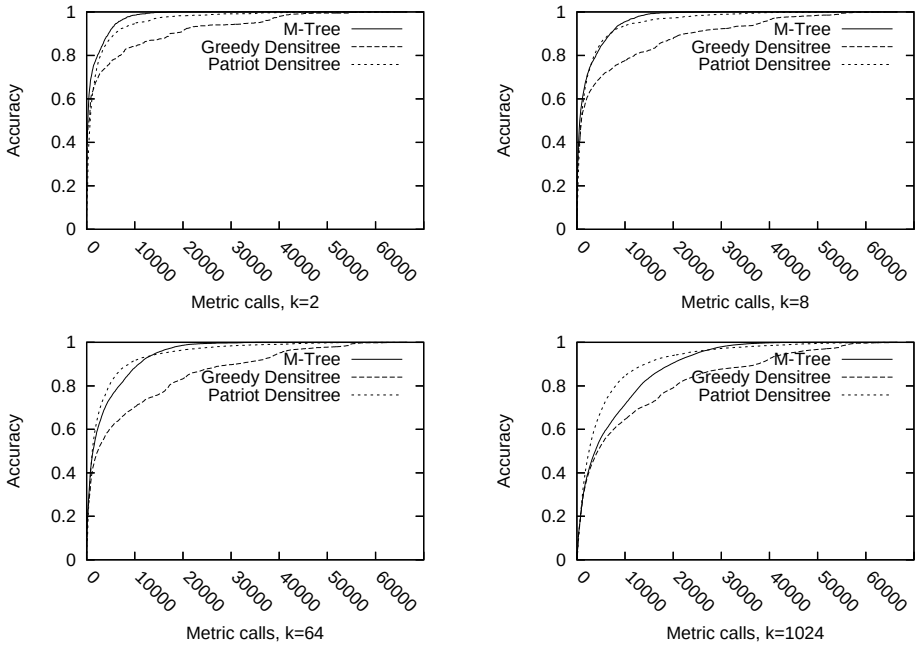


Figure 5: `disk` dataset

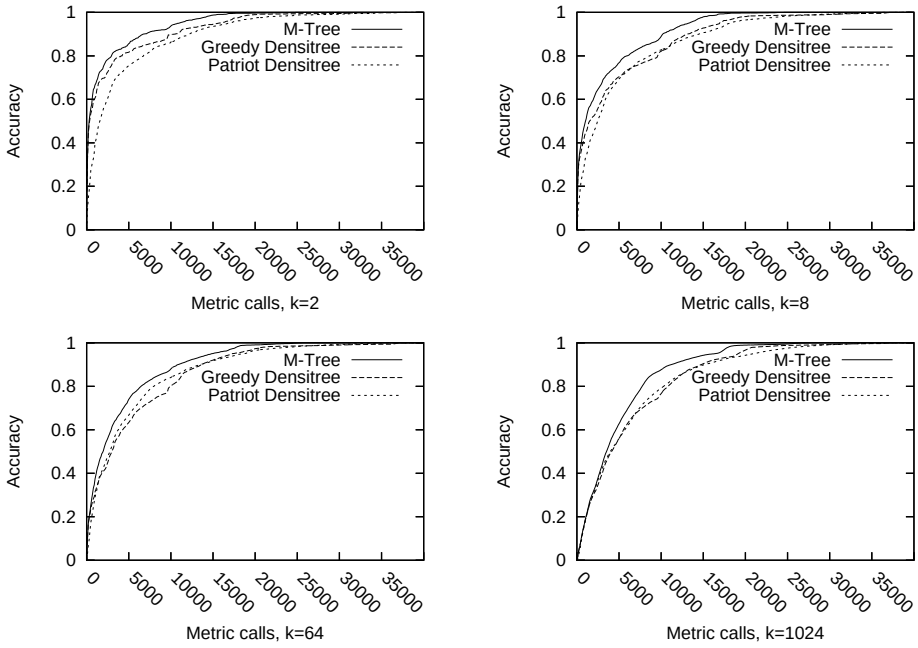
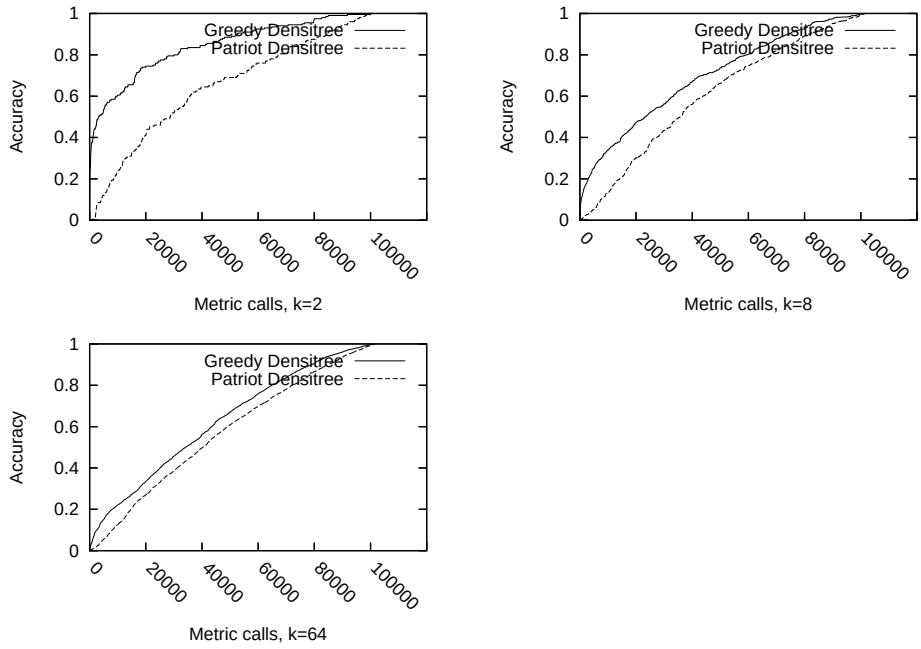


Figure 6: `sprot40` dataset



close to the metric tree performance without relying on the triangular inequality. The PATRIOT$\tilde{\text{d}}$ensitree even tends to slightly outperform[16] the metric tree when the number of requested neighbors is large ($k = 64$ and $k = 1024$).

The results against the `sprot40` dataset are mitigated. On one hand the greedy densitree accuracy profile for $k = 2$ clearly indicates that exhaustive search can be outperformed (75% of accuracy with 20% of exploration). On the other hand, the PATRIOT densitrees perform very poorly. As we have seen in Section 3.4, the PATRIOT estimator relies on many heuristics and the prohibitive CPU computations[17] of the Smith-Waterman distance complicates the "tuning" of appropriate density estimators.

TODO: Smith-Waterman value between very sequence is just noise, any pointer on that ?

We believe the poor performances of the PATRIOT estimator for $k = 2$ is and the poor performance of both density estimator for large neighborhoods ($k = 64$ and $k = 1024$) are related. TODO: complete this.

[16]Nevertheless, it has to be noticed that in practice the CPU costs are not limited to the call the similarity function. Based on a pure CPU benchmark, the PATRIOT densitree would have been outperformed by the metric tree on all the metric datasets where the similarity functions are cheap to compute.

[17]The sole Smith-Waterman computations (excluding all other elements) required to compute a single, yet already reduced to 100 sample queries, `sprot40` query accuracy profile is more than 100h of CPU for a single 2Ghz processor

5 Conclusions

We have introduced, to our knowledge, the first general approach of nonmetric NNS. This approach includes the *query accuracy profile* as a new evaluation criterion of the execution of a nonmetric NNS query, the *density estimation* concept of as a ranking criterion to guide the NNS exploration process and the *densitrees* as a new category of data structures to perform nonmetric NNS.

The densitrees as well as the metric trees have been tested against well established metric and nonmetric datasets. The proposed implementations for densitrees are still very preliminary and we expect large improvements from more accurate density estimators. Our empirical metric NNS results also indicate, maybe a quite counter intuitive result, that the triangular inequality pruning is only second in term of impact on the NNS to the accuracy of the density estimation

This conclusion should not be misinterpreted as "the triangular inequality is an unimportant factor for metric NNS", but it indicates future NNS developments, both empirical and theoretical, must take into the notion of density estimation in order to improve the NNS methods.

References

[1] http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html.

[2] http://www.expasy.org/sprot/.

[3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[4] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[5] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbors. `http://hunch.net/~jl/projects/cover_tree/cover_tree.html`, (*to appear*), 2005.

[6] Edgar Chavez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and Jose L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[7] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric space. In *Proc. Of the 23 rd VLDB Conference*, 1997.

[8] Paolo Ciaccia, Fausto Rabitti, Pavel Zezula, and Marco Patella. M-tree project. http://www-db.deis.unibo.it/Mtree/.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[10] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, New York, NY, USA, 2004. ACM Press.

[11] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pages 518–529, 1999.

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529. Morgan Kaufmann Publishers Inc., 1999.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[14] D. Jacobs, D. Weinshall, and Y. Gdalyahu. Class representation and image retrieval with non-metric distances. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(6):583–600.

[15] Ting Liu and Andrew W. Moore. Datasets repository. http://www.cs.cmu.edu/ tingliu/dataset/dataset.html.

[16] Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

[17] B. S. Manjunath. Airphoto dataset. http://vision.ece.ucsb.edu/download.html.

[18] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(8):837–842, 1996.

[19] M. Omohundro. Bumptrees for efficient function, constraint, and classification learning.

[20] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[21] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, volume 40, pages 175–179, 1991.

[22] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.

[23] Peter N. Yianilos. Excluded middle vantage point forests for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ, July 1998.

[24] Peter N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 361–370, 2000.

[25] B. Zhang and S.N. Srihari. A fast algorithm for finding k-nearest neighbors with non-metric dissimilarity. In *FHR02*, pages 13–18, 2002.