Curiositas

Journal of undergraduate research at Montana State University **Fall 2021**

Accessibility Note: This PDF contains mathematical notation that may not be compatible with some assistive technologies. For a LaTex version of the manuscript or for other assistance, please contact CuriositasJournal@montana.edu.

Title:
Approximating Expensive Distance Metrics

Authors:
Elliott Pryor, Nathan Stouffer

Author Affiliation:
Gianforte School of Computing, Montana State University

Volume:
Fall 2021

Pages:
22-27

Abstract:
Computing the distance between point *a* and point *b* is typically considered to be very easy. However, there are times when computing a distance can take significant computation time; we call these expensive distance metrics. Suppose we have some expensive distance metric and we need to compute the distances between a bunch of points. This paper explores a method that to reduce the number of queries to the distance metric and the effect on clustering. The authors find that total run time can be reduced while only inducing small inaccuracies in clustering output.

Curiositas is an interdisciplinary research journal dedicated to presenting the breadth and depth of undergraduate research that occurs at Montana State University. The journal places a particular emphasis on showcasing overlooked domains of undergraduate research, such as the humanities and arts, alongside traditional scientific research. *Curiositas* is committed to the belief that research on MSU's campus does not just occur in large laboratories and research groups: it occurs in every discipline and touches every element of scholarship that occurs at MSU. Articles in *Curiositas* are reviewed by a faculty member in the appropriate discipline (where applicable) and by an interdisciplinary undergraduate review committee.

Montana State University
Department of Microbiology & Cell Biology
109 Lewis Hall · PO Box 173520 · Bozeman, MT 59717
406-994-2902 · mcb@montana.edu

Please send questions and comments to
CuriositasJournal@montana.edu

montana.edu/curiositas

Computer Science
http://doi.org/10.15788.f2021.curio5

Approximating Expensive Distance Metrics

Elliott Pryor and Nathan Stouffer[1]
Gianforte School of Computing, Montana State University

Computing the distance between point *a* and point *b* is typically considered to be very easy. However, there are times when computing a distance can take significant computation time; we call these expensive distance metrics. Suppose we have some expensive distance metric and we need to compute the distances between a bunch of points. This paper explores a method that to reduce the number of queries to the distance metric and the effect on clustering. The authors find that total run time can be reduced while only inducing small inaccuracies in clustering output.

1. Introduction

Computing the distance between point *a* and point *b* is typically considered to be very easy. In most cases, we use simple Euclidean distance in two dimensions $d(a,b) = \sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$. These sorts of distances can be evaluated by a computer very quickly.

But what if that is not always the case? Some applications use metrics (subsection 2.1) that are very expensive to compute. What if we said the distance between two points on Earth is the minimum distance making sure that a person could walk the entire thing (i.e. you can't walk off a cliff, over water, up a wall, etc.). Well that makes things harder as the answer is not always the straight line between *a* and *b,* we might have to go around a mountain or wind along a riverbank. There are a lot of different paths that we would have to compare which takes a long time. If we have to compute this distance a hundred times that would take a while, but what about a million, or a few million.

In many cases, we want to consider the distances between many different objects. For example, when clustering points we typically want to find the closest points to another. If we want to find the closest object to another point, we have to consider the distances to all the other objects in the dataset. If we want to do this for all pairs of points in the dataset (as in clustering) this results in $\binom{n}{2}$ distance computations, where n is the total number of points in the dataset.
If n is large this can result in a huge number of distance computations (which we assume to take a long time).

We want a way to approximate these distances so we don't have to do the long distance computation so many times. In this paper, we implement an existing approximation algorithm [5] in order to test the runtime improvements compared to the loss of accuracy from the approximation. Our project is based on work by Kerber and Nigmetov [5], a paper that presents an algorithm that computes an approximate distance metric on a finite metrics space.We implemented the approximation algorithm and tested it on two "expensive" metric spaces: Hausdorff distance [4] on point sets and continuous Frechet distance [8] on paths in $\mathbb{R}$. Then we clustered the points using DBSCAN [3] with the actual and approximate distances. DBSCAN gives a potential practical application where this algorithm could be used, and can help map or quantify distortions caused by the approximation. We evaluated the performance of DBSCAN with homogeneity, completeness, and the V-measure [7].

The rest of the paper is organized as follows. In Section 2 we provide basic information about what a metric is and the distance functions. In Section 3 we provide more details about the approximation algorithm. In Section 4 we describe our experiment setup, and provide the results in Section 5. Finally, we give a discussion and concluding remarks in Section 6.

2. Background

2.1 Metrics

What is a metric? At a high level a *metric* is a function with three properties (like Euclidean distance). If

1. This work was done during a course project in Computational Geometry, under guidance of Dave Millman.

d(a b) is our metric then it has to satisfy these conditions:

$$d(a, b) \in [0,\infty) \quad (2.1)$$
$$d(a, b) = 0 \quad \text{iff } a=b \quad (2.2)$$
$$d(a, b) = d(a, b) \quad (2.3)$$
$$d(a, b) \leq d(a, c) + d(a, b) \quad (2.4)$$

The first one (2.1) just says that we cannot have a negative distance.The second one (2.2) says that if the distance between two points is zero (i.e. they are right on top of each other) then they are the same point. It also says that the distance between a point and itself must be zero. The third condition (2.3) says it is just as far from a to b as it is from b to a. The last condition (2.4) is called the triangle inequality and it is important to this paper. It says that it cannot be shorter (it could be the same) to go to a third `middle' point then to the destination. For instance, it cannot be shorter to drive from Bozeman to Helena, then Helena to Butte than it is to drive directly from Bozeman to Butte.

2.2 Big O Notation

Big O notation is a way to denote the asymptotic complexity of an algorithm. This is often used to denote running time or space usage of algorithms.

Definition 2.5 (Big O Notation)– A function f is Big O of a function g, denoted $f \in O(g)$, if there exists k, c such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n > k$.

One way to think about this, is to think about the dominant term in the function, or the highest order term. For n large enough, the contribution of the other terms is negligible. We can also envision this graphically, if we zoom way out the graph of $f(x)$ looks pretty much the same as the graph of $c \cdot g(x)$.

2.3 Continuous Frechet Distance

Frechet distance is a way to compare the distance between two curves. One common way to think about this is a person walking their dog. The person is on one curve, while the dog is on the other. The Frechet distance is the shortest leash the person could use.

Mathematically, where A, B are curves, and α, β are continuous, non-decreasing reparameterizations of [0,1] where $\alpha(0) = \beta(0) = 0$ and $\alpha(1) = \beta(1) = 1$. [8]

$$F(A, B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A(\alpha(t)), B(\beta(t))\} \quad (2.6)$$

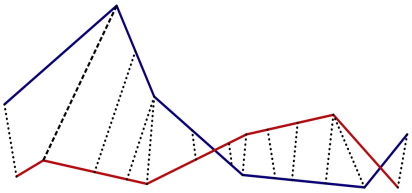


FIGURE 1
Vizualization of Frechet Distance

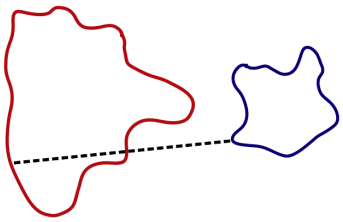


FIGURE 2
Vizualization of Hausdorf Distance.

2.4 Hausdorff Distance

Hausdorff Distance is used to compare the distance between two sets or point clouds. It is the maximum of the minimum distances between the sets.
So take two collections of points, call them A and B. Pick a point in A, then find the closest point in B. We do this for all points in A. Then repeat for B to A, and choose the one that is the biggest.
In our experiments, we just used Euclidean distance (straight line $\sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$) as the distances between individual points within the point clouds.

Mathematically, let d be a distance metric, then

$$H(A, B) = \max\left\{\sup_{a \in A} d(a, B), \sup_{b \in B} d(A, b)\right\} \quad (2.6)$$

We can fairly trivially see that this takes $O(nm)$ time where $m = |A|$, $n = |B|$. So for large sets (n, m big) this can be very expensive.

2.5 Clustering Metrics

These metrics are used to evaluate the quality of the clustering results in our experiments. We use: completeness, homogeneity, and v-score [7]. They are three different ways of considering how accurate a given clustering is on a scale 0-1. Suppose we have a set of points with true labels - classes. The goal of the clustering problem is to produce clusters of these points that exactly match the true labels of the points; i.e. a cluster will contain all members of one class, and no members of any other class.

Homogeneity - measures if all clusters contain only points of a single class. A cluster could be fully homogenous if it contains only a few number of points (of same class). In other words, a cluster can be homogenous if it is 'smaller' than or equal to the class.

Completeness - measures if all points in a class belong to a single cluster. A cluster could be complete if it contains everything (we produce only one cluster). In other words, a cluster can be complete if it is `larger' than or equal to the class.

V-Measure - combines homogeneity and completeness together in one value. It takes the weighted average of homogeneity and completeness. For our experiments we weight both terms equally, so it is just the average of the two values.

We note a perfect clustering corresponds to a score of 1 on all metrics.

3. Algorithm

We implemented an $1 + \varepsilon$ approximation algorithm [5] for use with expensive distance metrics. Our implementation can be found on our GitHub repo.[2]

The overall goal of the algorithm is to reduce the number of explicit distance computations that are needed. The $1 + \varepsilon$ is the error ratio, so the $\frac{\text{approximate distance}}{\text{actual distance}} \leq 1 + \varepsilon$ for any pair of points. We can choose ε to be anything we want, so this can be an arbitrarily good approximation.

In the base case (computes exact distance, $\varepsilon = 0$) we would need to compute all the pairwise distances $\binom{n}{2}$. At a high level, the algorithm computes a pairwise distance between a, b and stores it as an edge in a graph. A graph just gives information about the connection: a is connected to c if there is an path from a to b in this graph. Think of the path as our trip planner, we can go from a to b, then from b to c.

Initially we don't know how far anything is apart, or if they are even connected. So when we compute this pairwise distance we look at all the other pairs of points and see if we can use a more efficient route. For instance, suppose my initial route from Bozeman to Butte was to drive to Los Angeles, then back to Butte. Then I compute the distance from Bozeman to Helena, I can realize that it is much faster for me to drive to Helena then to Butte.

This algorithm uses the triangle inequality (equation 2.4) to approximate distances to neighboring points. After computing a distance between two points, it then considers all other pairs of points and updates upper and lower bounds (with the triangle inequality) through this newly computed edge. Once the ratio $upper/lower \leq 1 + \varepsilon$ for all pairs of points the algorithm terminates. The goal is to minimize the total number of distance calculations (which are assumed to be expensive).

At best, this algorithm runs in $O(n^2)$ time. Since after each distance calculation the algorithm re-computes all other pairwise bounds (which is $\binom{n}{2}$). At worst, the algorithm must compute all pairwise dis-

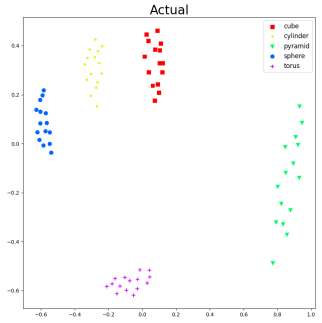
2. https://github.com/nathanstouffer/expensive-distance-metrics

Epsilon	Homogeneity	Complete	V-Score	Time (s)
10	0.929056	0.785739	0.851408	559.1532
5	0.969117	0.805154	0.87956	742.6007
3	0.982907	0.824797	0.896937	1025.282
2	0.982907	0.878377	0.927706	1194.397
1.5	0.982907	0.935371	0.95855	1408.396
1	0.982907	0.935371	0.95855	1764.475
0.5	0.982907	0.976183	0.979533	2837.134
0.35	0.982907	0.976183	0.979533	3747.833
0.1	1	1	1	7007.029

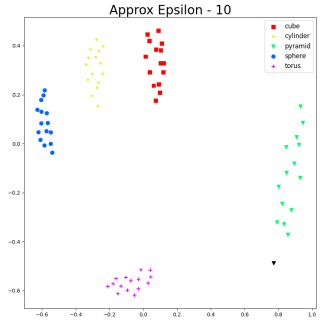
TABLE 1

Clustering metric run times for the Athens small data set

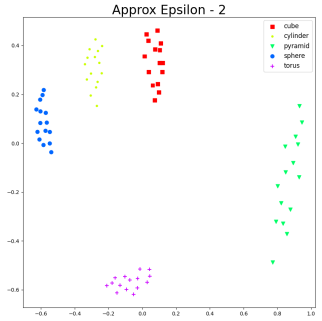
Table displaying clustering metrics (measured with respect to the complete algorithm) and the run times for various $\mathcal{E}$ values ran on the Athens small data set.



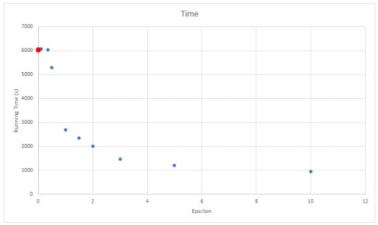
(a)



(b)



(c)

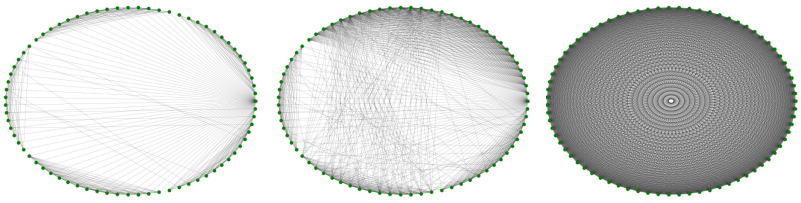


(d) Running time in seconds to compute distances on standard home computer

FIGURE 3

Clustering results on the Shapes data set for various values of ε.

The color denotes which cluster elements are a part of. Points that are measured as closer together are in the same cluster.



(a) $\varepsilon = 10$ (b) $\varepsilon = 2$ (c) $\varepsilon = 0.1$

FIGURE 4

Resulting approximation graph for various epsilon values on the Shapes data set.

This is a visual representation of which pairs of points distances are computed between.

tances, and it then still re-computes all other pairwise bounds, resulting in $O(n^4)$. However, it is assumed that the distance computation is very expensive relative to n. So, this factor of n^2 from updating the bounds is relatively insignificant.

4. Methods

For the continuous Frechet distance, we used the Athens small data set found in [1]. This was a small data set of containing 73 paths. We found an online library that implemented an approximation algorithm of the continuous Frechet distance.[3] We set the Frechet distance epsilon at 0.00001 to get as close as possible to the real distance between paths. Then we found the complete distance matrix and ran the approximation for various epsilon values.

For the Hausdorff distance we made our own artificial data set. We generated 6 different shapes in the cube $[-1,1]^3 \subset \mathbb{R}^3$. We have a sphere, pyramid, torus, cylinder, and cube. Each shape consists of 500 randomly generated points on the boundary of the surface.Then we ran the approximation algorithm while varying ε and compared the results of the clustering to that of the true distance.

After computing the distance matrix, we used the *mds* [2] function in Python's scikit-learn library [6] to embed the points in $\mathbb{R}^2$ for a visualization of what the clustering did.

5. Results

For each of the data sets, we display selected clustering output embeddings in $\mathbb{R}^2$ as well as a table showing runtimes and the metrics used to analyze cluster performance. The information for Athens can be found in Supplemental Figure 1[4] and Table 1. Supplemental Figure 1 shows the clustering output for the complete distance graph, $\varepsilon = 10, 3, 1, 0.35$ in Supplemental Figure 1a-1e respectively. Supplemental Figure 1f plots the run times vs epsilon value.
The red dot marks how long computing the complete distance matrix takes. The information for Shapes can be found in Figure 3 and Supplemental Table 1. The Shapes figure and table show information in the same style as the Athens figure and table.

Figure 4 depicts a visualization of the graphs produced from the approximation algorithm ran on the Shapes data set. Note that with $\varepsilon = 0.1$, the graph is complete and computes every edge.

6. Discussion

For the Shapes data set, the approximation algorithm performed very well. With $\varepsilon = 10$, the approximation performed almost identically to the true distance. Then when $\varepsilon = 2$, it performed exactly the same as the true distance. This can be seen in Figure 3 and the corresponding subfigures. This is possibly due to the dataset being very contrived. We had to use a more simple dataset due to have a more reasonable running time. This dataset could be expanded to include noise or more complicated shapes. But the purpose of this project was to gain a rough idea of applicability of this method, so improving the dataset is not a priority. This dataset has shown that the approximation algorithm can perform very well even at high ε values, drastically reducing running time while still maintaining comparable performance. In Figure 4, we can see the edge density increase as epsilon decreases. This matches our intuition that more direct distances must be computed as we decrease epsilon. It is also interesting to note, in Figure 4a it is easy to see that the clusters (adjacent nodes in circle) are more connected (we know these are the cluster because we read the clusters in groups). So essentially we have 5 highly connected clusters, with an edge or two to give an intuition as to where the other clusters are.

For the Athens dataset, the approximation algorithm still performed very well.This dataset is far more realistic and exemplary of a real-world application. We see in table 1 that the results get increasingly better results (not perfect) as we decrease ε. Although, even at incredibly high $\varepsilon = 10$ we still have a v-score of 0.85. This is still 'pretty good' depending on the application and runs in 10% of the time. So this algorithm could be very practical for real world issues.

One thing to consider when using this algorithm is the use of large data. Due to the extra n^2 term from updating bounds, this algorithm can pay a heavy price

3. The implementation was provided by an open source library Fred found at https://github.com/derohde/Fred
4. Supplemental figures are available in the digital version of this publication at http://doi.org/10.15788.f2021.curio5

on large datasets. So one area for future work is identifying this 'goldilocks' zone where we find the balance between number of points, and the distance computation cost.

In summary, this algorithm could be very applicable to real world problems. It depends on the application and how precise the distance computations need to be. Often, the data points also already have a large degree of uncertainty, so the loss of precision from the approximation isn't relevant. It can also be good to use as a prototype to be able to quickly run trials to gain an idea or test an algorithm, and save the complete execution for a more final product.

References

[1] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and CarolaWenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. GeoInformatica,19(3):601–632, 2015.

[2] Ingwer Borg and Patrick JF Groenen. Modern multidimensional scaling: Theory and applications. Springer Science & Business Media, 2005.

[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd, volume 96, pages 226–231, 1996.

[4] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing images using the hausdorff distance. IEEE Transactions on pattern analysis and machine intelligence, 15(9):850–863, 1993.

[5] Michael Kerber and Arnur Nigmetov. Metric spaces with expensive distances. International Journal of Computational Geometry & Applications, 30(02):141–165, 2020.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[7] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), pages 410–420, 2007.

[8] Timothy Randall Wylie. The discrete Fréchet distance with applications. PhD thesis, Montana State University-Bozeman, College of Engineering, 2013.

Elliott Pryor is a Senior at Montana State University studying Computer Science and Mathematics. He was a recipient of the 2021 Goldwater Scholarship, and he has been involved in undergraduate research since his freshman year. He hopes to develop new machine learning technologies for applications in healthcare and medicine. Beyond work, he enjoys reading as well as hiking, camping, and rock climbing.

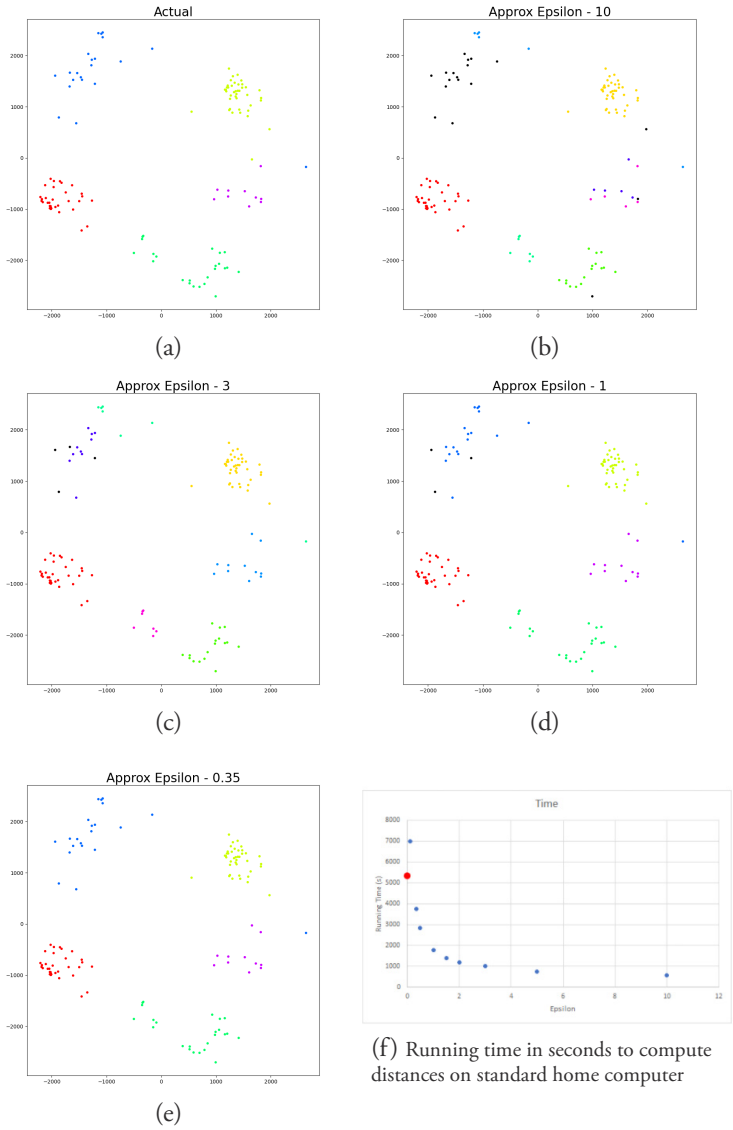
Nathan Stouffer graduated from MSU in Spring 2021 with a degree in Mathematics and Computer Science. Currently, he is a software engineer at onX maps. His favorite game is cribbage and he loves to ski, run, and play ultimate frisbee!

Epsilon	Homogeneity	Complete	V-Score	Time
10	1	0.970462	0.98501	938.984
5	1	0.970462	0.98501	1200.163
3	1	0.970462	0.98501	1471.606
2	1	1	1	2009.892
1.5	1	1	1	2354.257
1	1	1	1	2700.931
0.5	1	1	1	5302.438
0.35	1	1	1	6043.424
0.1	1	1	1	6069.802

SUPPLEMENTAL TABLE 1

Clustering metric run times for the Shapes data set

Table displaying clustering metrics (measured with respect to the complete algorithm) and the run times for various $\mathcal{E}$ values ran on the data set.



SUPPLEMENTAL FIGURE 1

Clustering results on the Athens small data set for various values of ε.

The color denotes which cluster elements are a part of. Points that are measured as closer together are in the same cluster.