MST Based Ordering of Point Clouds and Curve Fitting

Christoph Dalitz[1][a] and Alexander Jongbloed[1]
[1]*Institute for Pattern Recognition, Niederrhein University of Applied Sciences, Reinarzstr. 49, Krefeld, Germany*
christoph.dalitz@hsnr.de

Keywords: Point Cloud, 3D, Minimum Spanning Tree, Curve Fitting.

Abstract: We present a new method to fit a curve through a set of unordered points in 3D space. The method first computes the distance of each point to an end point in the longest path in the MST. This distance is then used as a predictor in a LOWESS regression where the number of neighbors is locally chosen on basis of a linearity index. Apart from computing a fitted curve, the method can also be used to order the points in the cloud. Compared to other approaches based on thinning, subsampling and spline interpolation, our method has the advantages of simplicity, that it does not depend on a good guess for parameters, and that it automatically yields a natural order on all points by assigning each point a hidden time value.

1 INTRODUCTION

In recent decades, a number of sensor techniques for computer vision have been developed that do not yield frame based pixel arrays, but *point clouds*, i.e., a collection of points with three spatial dimensions. Some of these sensors, for example stereo DVS (Pohle-Fröhlich et al., 2024), also yield a time stamp for each point. For other sensors, for example AT-TPC (Ayyad et al., 2020), the time resolution is too low for measuring the time information, but the points are at least recorded in correct time order. And there is a third group of sensors that do not record time information at all, for example airborne LIDAR (Lohani and Ghosh, 2017), or CR-39 plastic nuclear track detectors (Kodaira et al., 2016). Moreover, point clouds without time information can also occur as the result of numeric procedures, for example the theoretical computation of stagnation points in the magnetically induced current density field of atomic models, which must be ordered for further processing (Dimitrova et al., 2025).

In the present article, we address the problem of ordering the points in such point clouds along an unknown curve $\vec{x}(t)$ and fitting a curve through the points. The points in the point cloud are thus assumed to randomly scatter around some curve $\vec{x}(t)$ where the time t is an unknown hidden variable. Neither the shape of the curve $\vec{x}(t)$ is known, nor its parametric form, but we assume it to have no self-intersections. The point clouds recorded by the above sensors generally contain many more points than can be associated with a single curve. We therefore assume that the point cloud has already been pre-processed with a segmentation algorithm, and that the input points for our algorithm actually all scatter around a single curve. Such a segmentation can be done with, e.g., TriplClust (Dalitz et al., 2019b), or density based clustering methods like DBSCAN (Ester et al., 1996) or MST splitting (Arning et al., 2025).

The lack of time information makes fitting a parametric curve $\vec{x}(t)$ problematic, because local regression methods like LOWESS (Cleveland, 1979) or moving least squares (Amirfakhrian and Mafikandi, 2016) rely on the time information as a predictor variable. A common approach to circumvent this problem is to thin out the point cloud and do a spline interpolation through the remaining few points. The thinning can be done, e.g., by sub-sampling (Sabsch et al., 2017; Zhao et al., 2011), by sampling from a local 2D regression made on projections on a locally fitted plane (Lee, 2000), or by choosing centers of clusters returned by some clustering algorithm (Yan, 2001; Peng et al., 2022). As the resulting curves are quite sensitive to the number of support points, there are further algorithms that improve an initial curve by some optimization scheme (Zhao et al., 2011; Rupniewski, 2014). These algorithms, however, sidestep the problem of ordering the original points and compute the curve only from a thinned out point set.

We propose a new method to order the points according to the edge distance to the endpoint of the longest path in the Euclidean minimum spanning tree

[a] https://orcid.org/0000-0002-7004-5584

This is a self-archived version of a paper that appeared in the Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP 2026), volume 3, pp. 250-257 (2026)

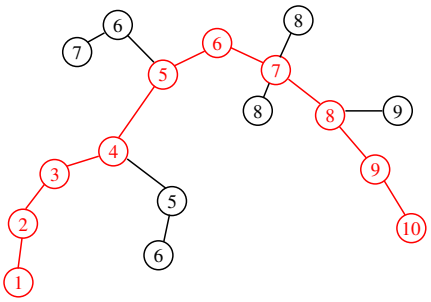


Figure 1: Longest path (red) in the Euclidean MST and the resulting edge distances from one of its end points.

(MST) of the data points. This distance can not only be used for ordering the points, but it can also serve as a predictor in a LOWESS local regression. For choosing the neighborhood size of the local regression, we suggest to base it on a linearity index that is computed from the eigen values of the covariance matrix of the neighborhood. The LOWESS fit can then be further utilized to improve the initial ordering of the points. The new method is tested on a number of point clouds, both from real world sensor data and simulated point clouds. In comparison to the algorithm by Lee (Lee, 2000), it turned out that our algorithm not only has a smaller runtime complexity, but is also more robust with respect to lack of smoothness in the data and still yields results in presence of gaps or sharp bends, which are situations where Lee's algorithm turned out to fail.

Although our algorithm is devised for unordered point clouds, it can also be used for fitting a curve through an already ordered point cloud. In this case, the MST should be replaced with the directed MST (Chu, 1965; Edmonds, 1967) so that the original point order is taken into account.

2 OUR ALGORITHM

Our algorithm consists of three steps: First, the number of edges ("distance") from each point in the MST to an end point of the longest path is computed. This distance is then used as a value for the unknown hidden time stamp of each point, which can be used for ordering the points. Moreover, it can be used as a predictor in a LOWESS local regression which yields a fitted curve. In an optional final step, the integer time stamps are refined by an orthogonal projection on the LOWESS fit, which can be used for tie breaking in the ordering or for recomputing an improved local regression.

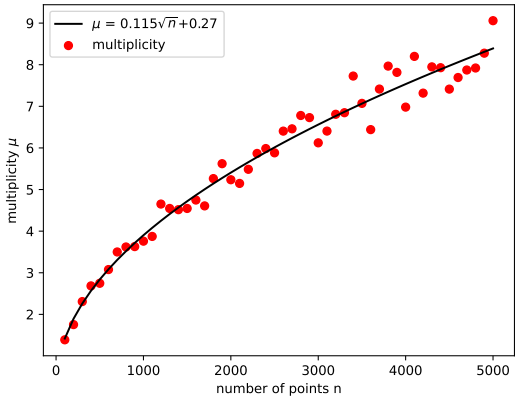


Figure 2: Increase of the multiplicity index μ when the number of points n on a fixed length increases. The black line fits the model $\mu = \alpha\sqrt{n} + \beta$

2.1 Ordering by edge distance

First, the Euclidean minimum spanning tree (MST) of all points is computed. The usual algorithms for computing the MST require the distance matrix for all points as an input and thus have a runtime complexity of $O(n^2)$, where n is the number of points. For the special case of the Euclidean distance, however, this runtime can be reduced to $O(\alpha(n)n\log n)$ (March et al., 2010), where $\alpha(n)$ is the inverse of Ackermann's function which is a very slowly growing function ($\alpha(n) \leq 4$ for $n \leq 10^{80}$). Although the Euclidean distance between the points was used for creating the MST, we subsequently ignore the edge weights after the MST has been built and subsequently measure "path length" only in the number of edges that the path contains.

Then, an endpoint of the longest shortest path in the MST must be found. The length of the longest shortest path in a graph is also known as its "diameter". For trees, the longest shortest path is simply the longest path and it can be found in linear time by two breadth first searches (BFS) (Uehara and Uno, 2007):

1. Start a BFS from an arbitrary point in the graph. The farthest point found is one endpoint of the longest path.
2. The other end point is found with a BFS from the endpoint obtained in the first step.

As we are only interested in one end point, a single BFS is sufficient to find an end point s.

Eventually, we compute for each point p_i its edge distance t_i with respect to s, which is the length of the shortest path between s and p_i. This can be done with a single BFS starting from s: The edge distance is simply the iteration step in the BFS during which the point is found (see Fig. 1).

Note that the same edge distance usually is as-

signed to more than one point. Actually, the average frequency μ among the edge distances (the *multiplicity index*) can be used to draw conclusions about the point density. As can be seen in Fig. 2, the multiplicity index is proportional to the square root of the number of points per curve segment.

2.2 LOWESS fit

For fitting a curve $\vec{x}(t)$ through the points, we apply the classic locally weighted regression known as LOWESS (Cleveland, 1979). We use the previously computed time stamps t_i as a predictor and consider the three spatial coordinates $p_i = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$ as response variables.

This means that, for each prediction time t and each spatial component x_j, we estimate the coefficients $\beta_0, \ldots, \beta_2$ of a second degree polynomial fit by minimizing the weighted sum of squares ($j = 1, 2, 3$)

$$\sum_{i=1}^{k} w_i(t) \left(x_j^{(i)} - \beta_0 - \beta_1 (t - t_i) - \beta_2 (t - t_i)^2 \right)^2 \quad (1)$$

The weights suggested by Cleveland are

$$w_i(t) = \begin{cases} \left(1 - \left|\frac{t_i - t}{h_k(t)}\right|^3\right)^3 & \text{for } \left|\frac{t_k - t}{h_k(t)}\right| < 1 \\ 0 & \text{else} \end{cases} \quad (2)$$

where $h_k(t)$ is the distance $|t - t_k|$ between t and the k-th farthest predictor value t_k. To make the fitting more robust with respect to outliers, there is an additional smoothing iteration, for which the interested reader is referred to (Cleveland, 1979).

The quality of the LOWESS regression depends critically on the number k of neighbors used for the cutoff point in Eq. (2). All library implementations, for example the function *loess* in base R, use a fixed number that is specified as a fraction of the total number of points. This approach does not work in our case, because it would lead to gross oversmoothing in case of long curves.

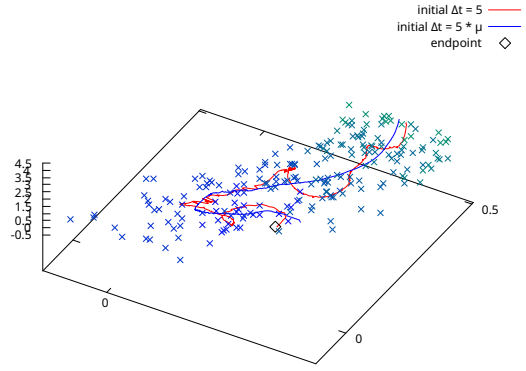
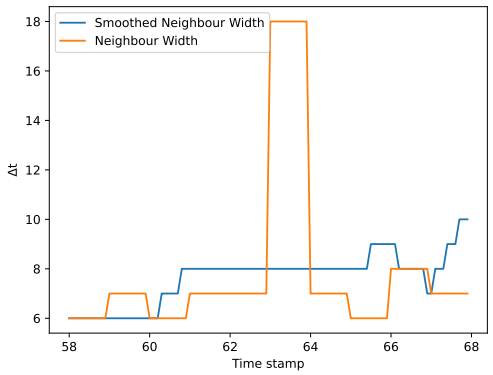
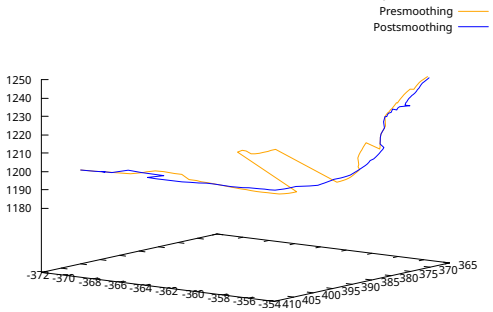


Figure 3: Example for the LOWESS curve with and without scaling the initial Δt with the multiplicity index μ.



(a) Δt with maximum linearity



(b) resulting LOWESS fit

Figure 4: Example for smoothing out outliers in the window width Δt.

We therefore did not use a global number of neighbors, but used a time window $[t - \Delta t, t + \Delta t]$ instead that depends on the position t at which $\vec{x}(t)$ is to be evaluated. This means that $h_k(t) = \Delta t$ in Eq. (2). To compute Δt, we first set it to a starting value Δt_0 that is then optimized with respect to a linearity index of the points in the time window $[t - \Delta t, t + \Delta t]$. For sparse point clouds, we found $\Delta t_0 = 5$ to be a good starting point, but, for denser point clouds, this was too small and resulted in overfitting. We therefore multiply it with the multiplicity index μ, which results in a smoother fits, as can be seen in Fig. 3.

As an optimization criterion for Δt, we used the following linearity index defined in (Hackel et al., 2016), which is based on the two largest eigen values $\lambda_1 > \lambda_2$ of the covariance matrix of the points:

$$\text{linearity} = \frac{\lambda_1 - \lambda_2}{\lambda_1} \quad (3)$$

If we search for the maximum linearity in a straightforward way by increasing Δt with step width one and then stop when the linearity decreases, only a local optimum is found which typically results in too small values for Δt. Therefore we allow the linearity for $\Delta t + 1$ to be lower than the previous maximum

linearity by a value of 0.025 but only up to $\Delta t + 5$. This resulted in greater window widths and smoother curves.

In order to avoid implausible results for Δt at sharp bends in the curve, we imposed an additional constraint in the optimization algorithm: The orthogonal regression line[1] of the points for window width $\Delta t + 1$ must point in a similar direction as the regression line of the previous width Δt. If this constraint is violated for $\Delta t > \Delta t_0 + 10$, the optimization is stopped. As a threshold for the angle between "similar" directions, we used $\pi/4$.

As can be seen in the orange curve in Fig. 4, this algorithm for determining Δt can result in some outliers like the peak at $t = 63$ that can result in kinks in the fitted LOWESS curve. We therefore finally smooth the curve with a moving average with a window width of ± 3.

2.3 Time stamp refinement

In this optional final step we refine the integer time stamps which were calculated from the MST. To this end, we project each point p_i onto the LOWESS fit. As the LOWESS fit is a non-parametric curve, there is no closed form solution for the projection of a point onto the fit. We therefore simply find the closest point on the curve $\vec{x}(t)$ sampled between $t-2$ and $t+2$ with a sampling width of 0.1. The corresponding timestamp then replaces t_i as the time stamp for p_i. Finally, the LOWESS fit is recalculated with the new time stamps.

3 LEE'S ALGORITHM

As a reference algorithm against which to compare our algorithm, we have chosen Lee's algorithm (Lee, 2000) because it is the most cited algorithm for curve fitting from unordered point clouds: In July 2025, Google Scholar listed about 400 citations for (Lee, 2000), versus about 50 citations for (Yan, 2001), and only single digit numbers for the other methods (Sabsch et al., 2017; Zhao et al., 2011; Rupniewski, 2014; Peng et al., 2022). There were already some open source implementations available, but only for 2D data, and one implementation by a github-user "aliadnani" for 3D data[2]. Although its description explains that it is "heavily based on" the paper by Lee, the implementation deviates in considerable ways and no documentation or rationale for the differences is given. We therefore created our own implementation exactly according to the description in (Lee, 2000).

Lee's algorithm consists of three steps. In the first step the point cloud is thinned using the moving least-squares method which moves every point onto a 2D weighted regression curve computed from its neighbors in a radius H projected on the plane that has been fitted through these neighbors. Lee emphasizes that a fixed H has several drawbacks and proposes a variable H which is increased if the correlation of the neighborhood is too small.

In the second step support points for the curve fit are sampled from the moved points. The sampling begins from a random point. Its neighbors in a radius h are split into two groups according to their projection on the direction of the regression line fitted with orthogonal least squares. The splitting point is the projection of the reference point. The farthest point with respect to the reference point is chosen in each group as a support point. Those new support points are then used as new start points to find further support points.

In the last step, an exact spline is fitted through the support points.

4 RESULTS

We have tested our algorithm both on simulated point clouds and on real data that were recorded in third party projects and that were kindly provided to us. The data were AT-TPC data from experiments in nuclear physics provided by Yassid Ayyad and presegmented by TriplClust (Dalitz et al., 2019a), insect trajectories recorded in the Bee Vision project and provided by Regina Pohle-Fröhlich (Pohle-Fröhlich et al., 2024) and presegmented with the algorithm described in (Arning et al., 2025), and numerically computed of magnetically induced current density field stagnation points in molecule structures provided by Maria Dimitrova (Dimitrova et al., 2025) and presegmented by TriplClust.

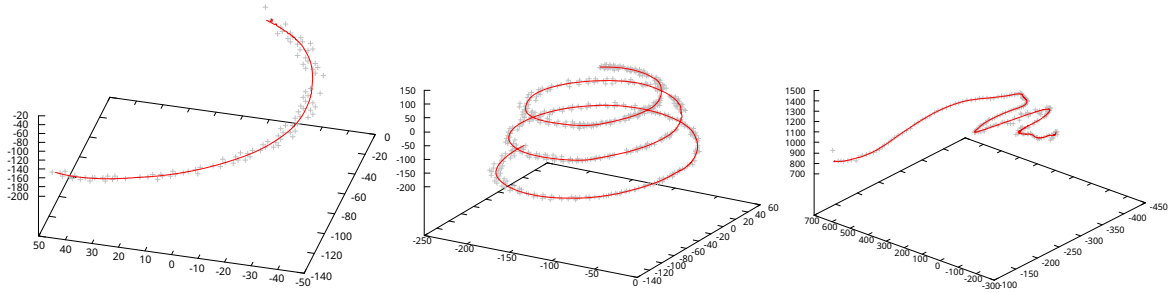
As there were no ground truth curves $\vec{x}(t)$ available for the real data, we could not assess the accuracy of our method in comparison to the method by Lee and could not measure its robustness with respect to a controlled change in the spread of the points around the curve in the data. To this end, we have additionally simulated the following three curves:

$$\vec{a}(t) = \Big(t \cdot \cos(2t), t, 2t\Big) + \vec{\varepsilon} \tag{4}$$

$$\vec{b}(t) = \Big(3 \cdot \cos(5t), 10t, 10t/\pi\Big) + \vec{\varepsilon} \tag{5}$$

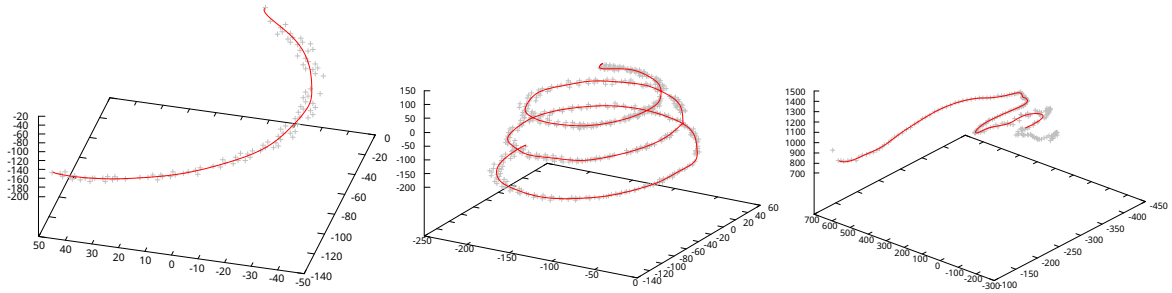
[1] The direction of an orthogonal least squares fit is the eigen vector to the largest eigen value of the covariance matrix.

[2] https://github.com/aliadnani/curves with the last commit in Nov. 2020



(a) AT-TPC ^{4}He+^{4}He experiment (b) AT-TPC ^{46}Ar+p event (c) BeeVision insect flight

Figure 5: Examples of our algorithm applied to real data.



(a) AT-TPC ^{4}He+^{4}He experiment (b) AT-TPC ^{46}Ar+p event (c) BeeVision insect flight

Figure 6: Examples of Lee's algorithm applied to the same real data as in Fig. 5.

$$\vec{c}(t) = \Big(\cos(20t), \sin(20t), 5t\Big) + \vec{\varepsilon} \qquad (6)$$

with $t \in [0,1]$ and $\vec{\varepsilon}$ are iid random numbers in a cube centered at the origin with edge length $2r$.

4.1 Real data

For the magnetic field stagnation points, we have only tested the ordering because fitting a curve is of no interest in this case (Dimitrova et al., 2025). An example with three curves and noise as segmented by TriplClust is shown in Fig. 7. Dimitrova et al. ordered the points by an orthogonal projection on the line fitted via orthogonal least squares. Our algorithm obtained exactly the same orders in all cases. This is

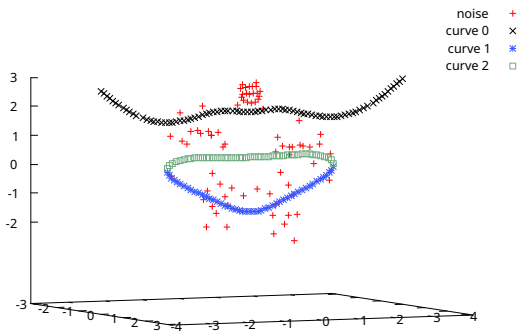


Figure 7: Magnetically induced current density field stagnation points of a Furan molecule split by TriplClust into groups representing single curves.

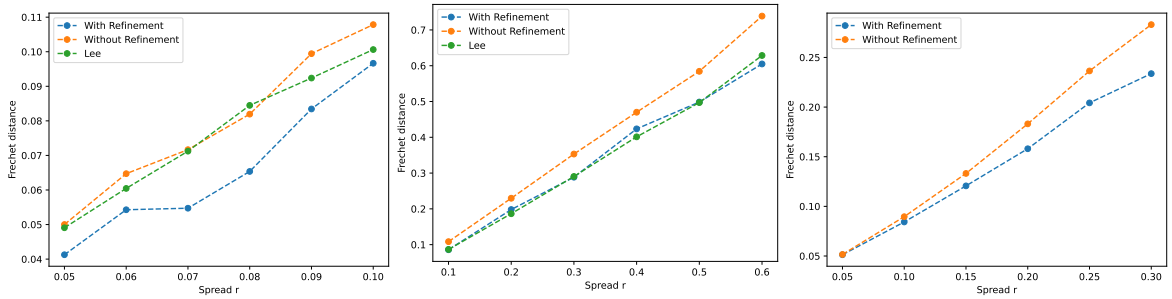
refinement	AT-TPC a	AT-TPC b	BeeVision
without	0.99897	0.99955	0.99982
with	0.99942	0.99957	0.99986

Table 1: Spearman correlation ρ between the true point order and the order estimated by our algorithm.

a very simple example, of course, but our algorithm has the advantage of applicability in more complex situations, too.

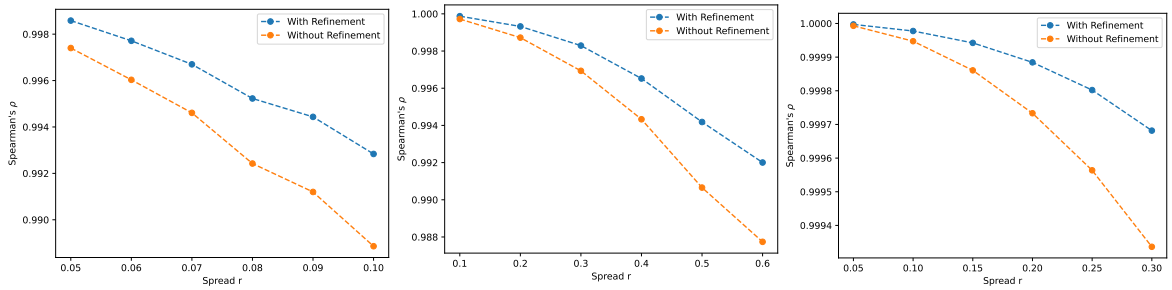
Examples for the application of our algorithm on AT-TPC data and data from the BeeVision project can be seen in Fig. 5. For all three examples, our algorithm fits a smooth curve through all points of the point clouds. Since the order of the points is known in these data sets, we can evaluate the conformity of our point order with the ground truth data. The resulting values for Spearman's ρ are shown in Tbl. 1. The refinement indeed improved the ordering in all cases, albeit only slightly. Note that all values are very close to one, which means that the estimated ordering is very close to the true order.

In Fig. 6, the same examples can be seen after the application of Lee's algorithm. The curves fits are smooth, too, but in Fig. 6c a problem of Lee's algorithm becomes apparent: If the curve has a sharp bend, the sampling step of Lee's algorithm fails to find further support points that are in the direction of the regression line. It therefore terminates without covering the complete point cloud.



(a) curve $\vec{a}(t)$ (b) curve $\vec{b}(t)$ (c) curve $\vec{c}(t)$

Figure 8: Fréchet Distance as a function of the spread r for curves simulated according to Eqs. (4)-(6).



(a) curve $\vec{a}(t)$ (b) curve $\vec{b}(t)$ (c) curve $\vec{c}(t)$

Figure 9: Spearman correlation ρ between the estimated and true point order as function of the spread r for curves simulated according to Eqs. (4)-(6).

4.2 Simulated data

For a quantitative comparison with Lee's algorithm, we additionally tested our algorithm on the simulated curves according to Eqs. (4)-(6). As the true curves are known in these simulations, we can compute the Freéchet distance between the estimated curve and the true curve. The Fréchet distance is robust with respect to reparametrizations $t \to \alpha(t)$ of the curves, because it is the infimum of the maximum distance between the curves over all reparametrizations:

$$F(\vec{x},\vec{y}) = \inf_{\alpha,\beta} \max_{t} \left\| \vec{x}(\alpha(t)) - \vec{y}(\beta(t)) \right\| \qquad (7)$$

For sampled curves, the Fréchet distance can be efficiently computed in $O(nm)$, where n and m are the number of sample points in the curves (Eiter and Mannila, 1994).

The median Fréchet distance of 100 simulations per spread r for each of the three curves is shown in Fig. 8. Note that Lee's algorithm had problems with early termination for the first simulated curve. We removed all values for which this problem occurred to give a fairer comparison. For the third simulated curve, however, the problems of Lee's algorithm were so severe that no real comparison could be made and we omitted the results for Lee's algorithm in Fig. 8c. We can nevertheless still compare the cleaned results for the first simulated curve and the results for the second simulated curve.

The refinement step improved the accuracy of the estimated curve in all three cases. With refinement, our algorithm performs better than Lee's for curve $\vec{a}(t)$, and comparable for curve $\vec{b}(t)$. As our algorithm has the additional advantage that it is more robust and resulted in complete fits in all cases, we conclude that it is preferable to the algorithm by Lee. When the spread r of the data around the true curve increases, the median Fréchet distance of estimate increases, too. This was to be expected, because the estimate follows the random data, not the hypothetical true curve.

For our new algorithm, we also evaluated the quality of the estimated order by means of Spearman's ρ. It should be noted that in this case, the time stamps underlying the simulation not necessarily represent a ground truth order, because, with increasing random spread r, the point order can be increasingly spatially inverted. The decrease of the sorting accordance with increasing r as seen in Fig. 9 was thus expected. It is interesting to note, however, that the refinement step improves the sort accuracy in all cases and that the algorithm with refinement is less sensitive to the random spread than the algorithm without refinement.

A runtime comparison of our algorithm and Lee's Algorithm only makes sense for the simulated curve

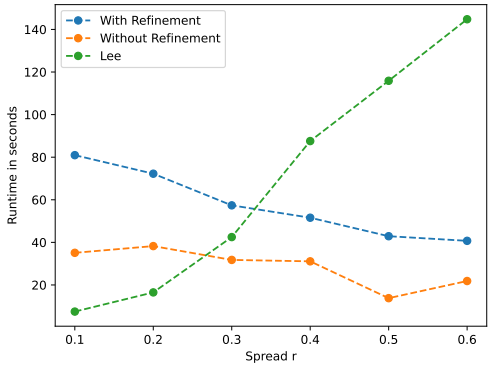


Figure 10: Runtime in seconds averaged over 100 runs as a function of the spread r for the simulated curve $\vec{b}(t)$.

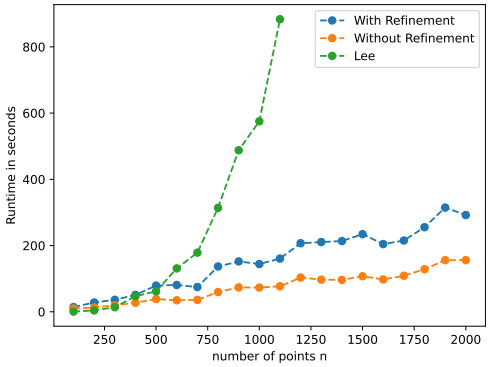


Figure 11: Average runtime in seconds as a function of the number of points.

$\vec{b}(t)$, because, on the other curves, Lee's algorithm often terminated prematurely which resulted in a shorter runtime, but the curves were incomplete. As can be seen in Fig. 10, Lee's algorithm is faster, but becomes slower with increasing spread. The runtime of our algorithm, on the contrary, is unaffected by the spread of the point cloud and becomes even slightly faster with increasing spread. This means that with increasing spread our algorithm outperforms Lee's algorithm.

The runtime difference becomes even more distinct if measured as a function of the number n of input points. We have not changed the length of curve $\vec{b}(t)$ while increasing n, which means that we not only increased the number of points, but also the point density. Note that we implemented both algorithms in Python, which results in a considerably slower runtime than an optimized implementation in, e.g., C++. Nevertheless the order of the runtime complexity as a function of n remains the same. As can be seen in Fig. 11, Lee's algorithm is faster for small n, but increases with a higher power than our algorithm. A closer analysis of the algorithms and the runtime curves showed that Lee's algorithm has an average runtime of order $O(n^3)$. Although our algorithm has a worst case runtime of $O(n^3)$, too, this reduces to an average runtime of only $O(n^{3/2})$ because the multiplicity index μ is of order $O(\sqrt{n})$ as the density increases, which means that the highest index of the MST distances is of order $O(\sqrt{n})$ and the number of chosen neighbors is on average proportional to μ, too.

4.3 Problems

Although our algorithms worked well in the examples presented in this article, it still has problems in two cases. The first, less severe, problem is that the curve fit at the end points is bend towards the endpoints of the longest path in the MST. Depending on the point density, these end points can vary randomly, which can result in suboptimal fits at the end points, especially in denser point clouds. An example can be seen in Fig. 3.

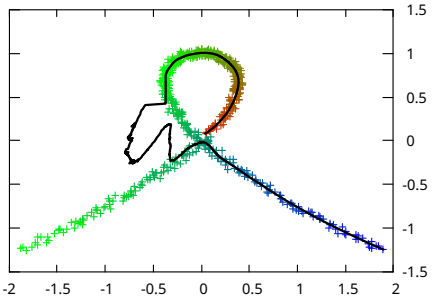


Figure 12: Example of an erroneous result of our algorithm on a self-intersecting curve.

The other, more severe, problem is that our algorithm fails to approximate the curve if the curve intersects or touches itself, because in these cases the MST "splits" at the touch point and the curve fit fails. An example can be seen in Fig. 12. Addressing this problem requires a different approach and is an interesting subject for further research.

5 CONCLUSIONS

Assigning time stamps to points in a point cloud along the longest path (diameter) in the Euclidean minimum spanning tree is a simple idea that has proven useful, both for ordering the points and for fitting a curve. In our experiments with ground truth time stamps, the resulting point order was in good agreement with the true point order. Using the MST edge distances as a predictor for a locally weighted regression is a simple curve fitting method from unordered point clouds. In our experiments it was more robust than Lee's algorithm with respect to sharp bends in the curve bends, and it has a lower runtime complexity.

A drawback of our algorithm is that it does not work in the case of self-intersecting or self-touching curves, because then the longest path does not cover some branches in the MST. This might be solved by a split and merge approach like that described by Arning et al. in the post-processing of MST-based clustering (Arning et al., 2025), but this is a difficult problem that requires a more thorough investigation.

ACKNOWLEDGEMENTS

We are grateful to Yassid Ayyad, Regina Pohle-Fröhlich, and Maria Dimitrova for kindly providing real world data for testing our algorithm.

REFERENCES

Amirfakhrian, M. and Mafikandi, H. (2016). Approximation of parametric curves by moving least squares method. *Applied Mathematics and Computation*, 283:290–298.

Arning, J., Dalitz, C., and Pohle-Fröhlich, R. (2025). Separation of insect trajectories in dynamic vision sensor data. In *International Conference for Computer Vision and Applications (VISAPP)*, volume 69, pages 69–77.

Ayyad, Y., Abgrall, N., Ahn, T., Álvarez-Pol, H., Bazin, D., Beceiro-Novo, S., Carpenter, L., Cooper, R., Cortesi, M., Macchiavelli, A., et al. (2020). Next-generation experiments with the active target time projection chamber (AT-TPC). *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 954:161341.

Chu, Y.-J. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836.

Dalitz, C., Ayyad, Y., Wilberg, J., Aymans, L., Bazin, D., and Mittig, W. (2019a). Automatic trajectory recognition in Active Target Time Projection Chambers data by means of hierarchical clustering. *Computer Physics Communications*, 235:159–168.

Dalitz, C., Wilberg, J., and Aymans, L. (2019b). TriplClust: An algorithm for curve detection in 3D point clouds. *Image Processing On Line*, 8:26–46.

Dimitrova, M., Arning, J., Dalitz, C., and Berger, R. J. F. (2025). A natural scheme for the quantitative analysis of the magnetically induced molecular current density using an oriented flux-weighted stagnation graph. Preprint. https://doi.org/10.26434/chemrxiv-2025-v4qh1.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Eiter, T. and Mannila, H. (1994). Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expertensysteme, Technische Universität Wien.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings KDD '96*, pages 226–231.

Hackel, T., Wegner, J. D., and Schindler, K. (2016). Contour detection in unstructured 3d point clouds. In *Proceedings of the IEEE Conference on Computer Cision and Pattern Recognition*, pages 1610–1618.

Kodaira, S., Morishige, K., Kawashima, H., Kitamura, H., Kurano, M., Hasebe, N., Koguchi, Y., Shinozaki, W., and Ogura, K. (2016). A performance test of a new high-surface-quality and high-sensitivity CR-39 plastic nuclear track detector – TechnoTrak. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 383:129–135.

Lee, I.-K. (2000). Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, 17(2):161–177.

Lohani, B. and Ghosh, S. (2017). Airborne LiDAR technology: A review of data collection and processing systems. *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences*, 87(4):567–579.

March, W. B., Ram, P., and Gray, A. G. (2010). Fast Euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 603–612.

Peng, K., Tan, J., and Zhang, G. (2022). A method of curve reconstruction based on point cloud clustering and PCA. *Symmetry*, 14(4):726.

Pohle-Fröhlich, R., Gebler, C., and Bolten, T. (2024). Stereo-event-camera-technique for insect monitoring. In *International Conference for Computer Vision and Applications (VISAPP)*, pages 375–384.

Rupniewski, M. W. (2014). Curve reconstruction from noisy and unordered samples. In *3rd International Conference on Pattern Recognition Applications and Methods*, pages 183–188.

Sabsch, T., Braune, C., Dockhorn, A., and Kruse, R. (2017). Using a multiobjective genetic algorithm for curve approximation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6.

Uehara, R. and Uno, Y. (2007). On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(05):911–930.

Yan, H. (2001). Fuzzy curve-tracing algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 31(5):768–780.

Zhao, Y.-d., Cao, J.-j., Su, Z.-x., and Li, Z.-y. (2011). Efficient reconstruction of non-simple curves. *Journal of Zhejiang University SCIENCE C*, 12(7):523–532.