The Start-Stop Diagram as a Tool for Continuous Trajectory Simplification

Mees van de Kerkhof`[1], Irina Kostitsyna[2], Marc van Kreveld[1], Maarten Löffler[1]

[1]Utrecht University, Princetonplein 5, 3584 CC, Utrecht
Email: {M.A.vandeKerkhof|M.J.vanKreveld|M.Loffler}@uu.nl

[2]TU Eindhoven, P.O. box 513, MF 4.142, 5600 MB Eindhoven
Email: i.kostitsyna@tue.nl

1. Introduction

The amount of collected trajectory data has increased explosively since the widespread adoption of GPS. Simplifying trajectories helps in storing this amount of information and for performing fast computations. The problem we want to solve in this paper is the Minimum Link Trajectory Simplification Problem: Computing a simplified trajectory with the minimum number of vertices such that the distance between simplification and original is less than a given threshold using an error metric.

We distinguish two versions of this problem: *Discrete*, where the vertices of our simplification are a subset of the original vertices; and *continuous*, where the vertices of the simplification are not restricted to a discrete set of points. Most research so far concentrated on the discrete case. See Zheng (2015: 6-8) for an overview of the field. Here we study the continuous case as it has some advantages: Continuous simplifications are smaller, as there are fewer restrictions on where to place the vertices. Depending on the trajectory, there can be considerable size differences between the optimal continuous and discrete simplifications. We study the continuous case where vertices of the simplification are restricted to lie on the original line segments.

The well-known Imai-Iri (1986) algorithm for polyline simplification considers the allowed *shortcuts*. A shortcut between two points is valid if we can replace the corresponding section of trajectory by a straight-line segment without violating our distance criterion. A graph is constructed with the vertices of the trajectory as nodes and edges between each pair of nodes that has a valid shortcut. A shortest path is then computed on this graph.

In the continuous case we cannot compute such a graph, as there are infinitely many potential shortcuts. But there are still ways to visually represent the set of shortcuts and compute simplifications using them. In the remainder of this paper we will show how the Start-Stop Diagram (SSD), which was first introduced by Aronov et al. (2016) to solve trajectory segmentation, can be used to visualize the shortcuts of a trajectory and compute simplifications. We will also show how to compute SSDs for the simplification of one-dimensional trajectories.

Many applications for simplification research, such as cartographic generalization of linear features (McMaster and Shea 1992), use polyline simplification methods that ignore the timestamps of a trajectory. There is a lack of research into simplification methods that use the timestamps, despite their importance to applications where moving objects are tracked. Therefore we use the *time-synchronous Euclidean distance* as our error metric, which is a distance measure based on De By and Meratnia (2004), where at every time t the Euclidean distance between the original trajectory and the simplification can be at most a given constant ϵ. See Figure 1.

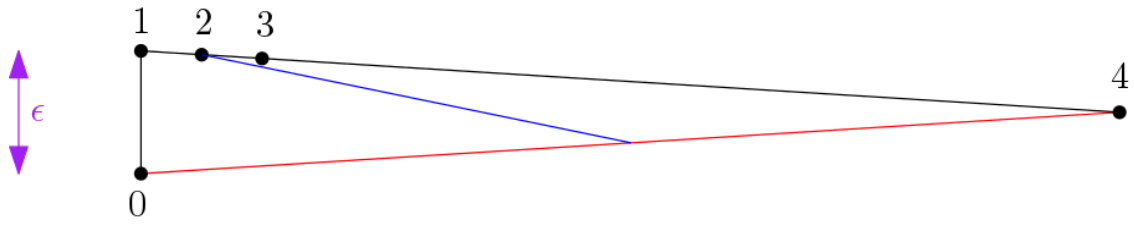

Figure 1. A 2-D trajectory showing the difference between normal curve distance metrics and the time-synchronous Euclidean distance. The timestamps of the vertices are shown. The shortcut from time 0 to time 4 (shown in red) is valid under distance metrics that do not use the timestamps, as they ignore that the trajectory rapidly speeds up after time 3. Under the time-synchronous Euclidean distance, the distance between the trajectories at e.g. time 2 (shown in blue) is greater than ϵ so the shortcut is invalid.

2. The Start-Stop Diagram

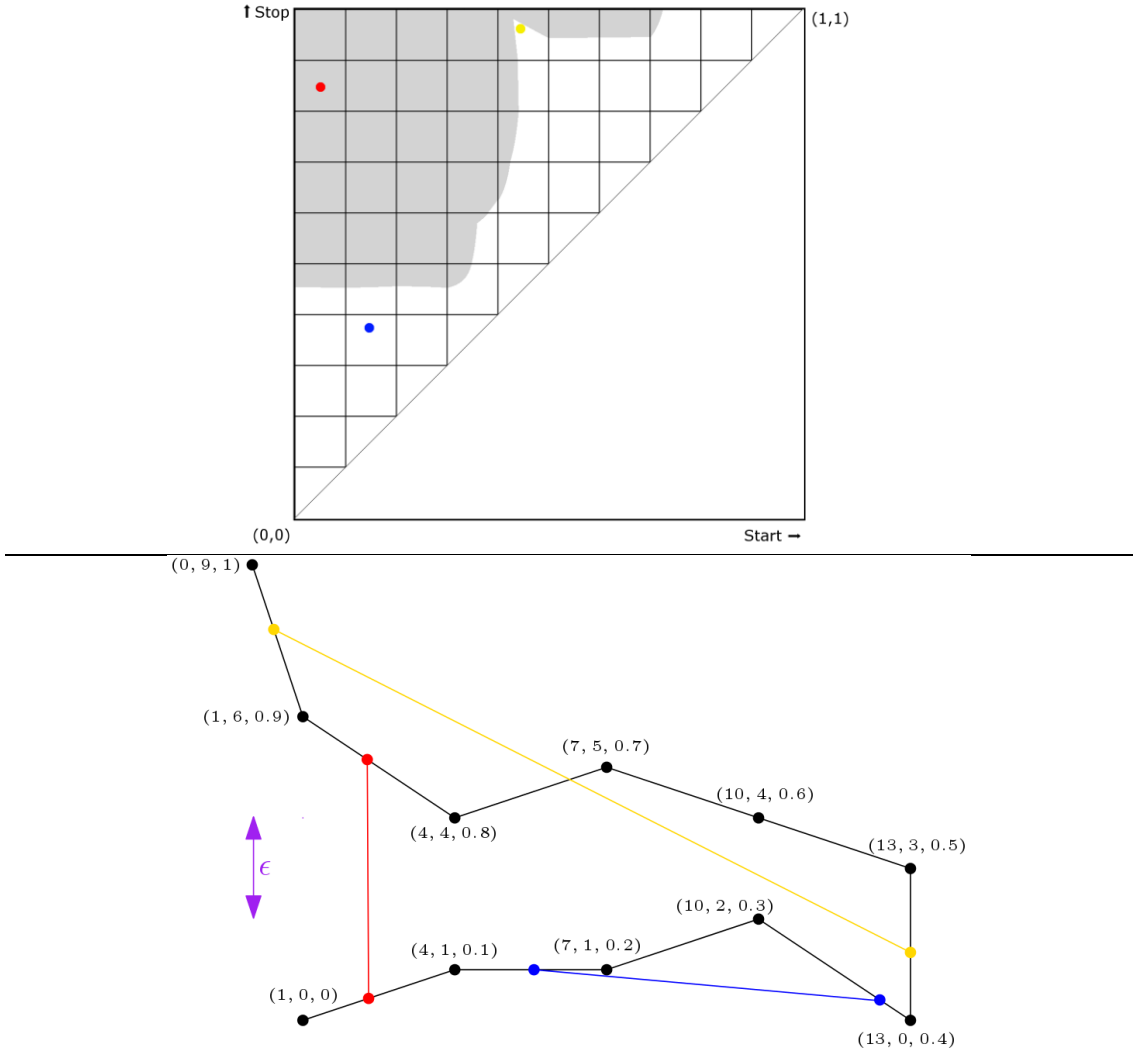

Figure 2. Top: A SSD with free space in white and forbidden space in gray. The grid lines indicate different segments so each cell shows the shortcuts between one pair of segments. Bottom: The trajectory the SSD is based on. Coordinates are given in the form (*x*,*y*,*t*). Three shortcuts are shown in both the SSD and trajectory as an example. If and only if the point associated with a shortcut is in the SSD's free space, the shortcut is valid.

If we normalize the timestamps on our trajectory such that they start at time 0 and end at time 1, we can encode each shortcut as a point on the upper left triangle of the unit square by taking the start and end times as the x- and y-coordinate respectively. A SSD is a coloring of each point based on the validity of the shortcut with respect to our distance metric, see Figure 2. By drawing vertical and horizontal lines at the starting- and stopping times of each segment, the SSD is separated into cells. Each cell contains the shortcuts between a single pair of input segments. E.g. the cell in the second row and fourth column of the SSD contains all shortcuts from points on the second segment of the trajectory to points on the fourth segment of the trajectory.

2.1 Computing simplifications using the SSD

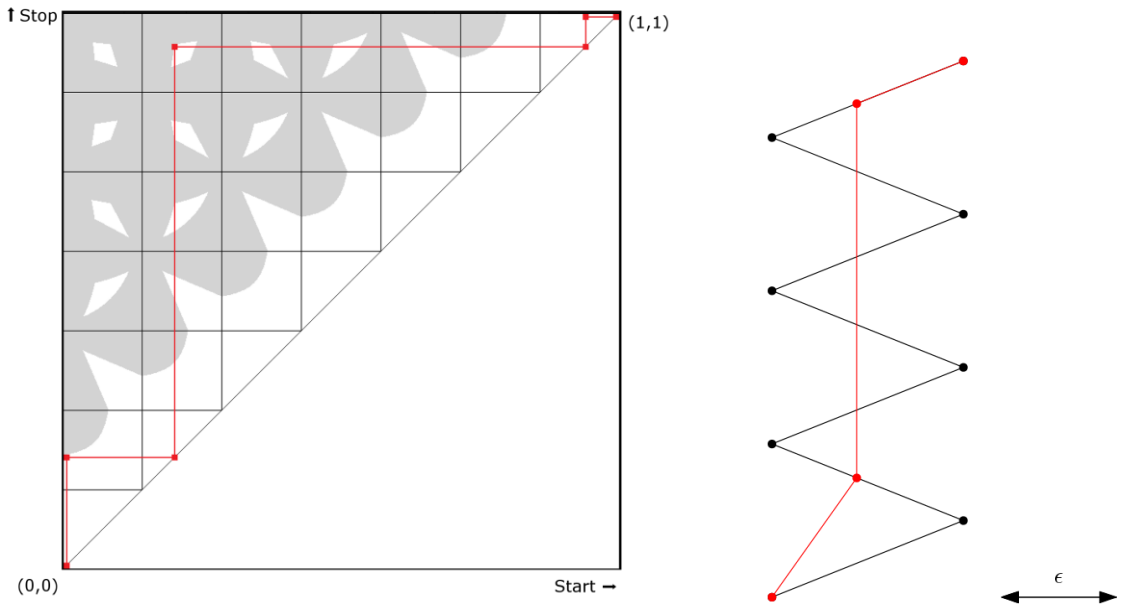

Figure 3. Left: A SSD with a valid staircase. Right: The simplification indicated by the staircase. Note that in the discrete case the curve cannot be simplified.

The set of valid shortcuts is the SSD's *free space*. Conversely, the invalid shortcuts form the *forbidden space.* Then, we can get valid simplifications by computing *staircases* in the SSD. A staircase is a path that goes from point (0,0) to (1,1) that consists of alternating vertical and horizontal segments. The convex vertices must be in the free space and the concave vertices on the main diagonal. The points that are associated with the convex vertices of the staircase then form a valid simplification, see Figure 3.

Using the SSD to test the validity of a simplification is easy as we can check if the points associated with each of the segments of the simplification lie within the free space of the SSD. It is also easy to compute a greedy simplification as we did in Figure 3 by incrementally taking free points with maximum y-coordinate, so we always cut ahead as much as possible. Computing the minimum-link simplification is more complicated, see Section 3.

2.2 Computing the SSD in 1-D.

To compute a simplification we first compute the SSD one cell at a time. Then finding the minimum-step staircase gives us a minimum-link simplification. We will now show how to compute the SSD for one-dimensional trajectory simplification with the time-synchronous Euclidean distance metric in polynomial time. One-dimensional trajectories are also known as time series.

For any shortcut it holds that if it is valid when only considering the distances at the times of each vertex, it must also be valid for all other points. (See the full paper for a proof). So, in the 1-D case if we extend each vertex with a vertical segment of size 2ϵ called an *augment*, shortcuts are valid if and only if they intersect all augments between their endpoints, see Figure 4.

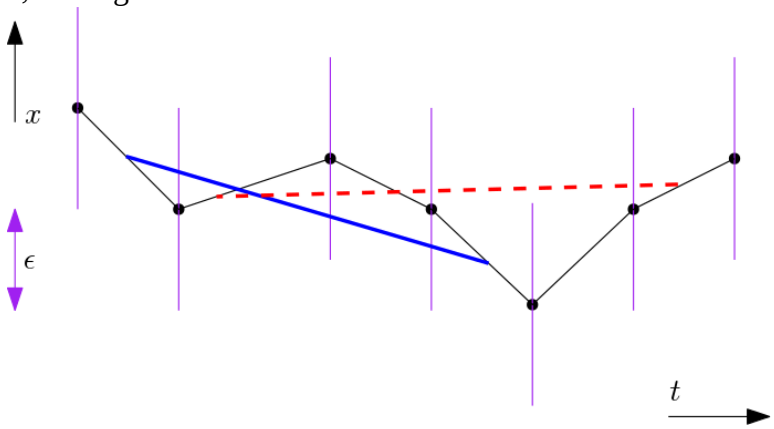

Figure 4. A 1-D trajectory with augmented vertices. The dashed shortcut is invalid because it does not intersect all augments between its start and end points. The other shortcut intersects the augments and is valid.

For each cell we have the start segment *a* (bounded by vertices a_0, a_1 , with a_0 having the earlier timestamp), and stop segment *b* (likewise bounded by b_0, b_1). We can parametrize all points on these segments with equations (1) and (2):

$$a(\lambda) = (1-\lambda)a_0 + \lambda a_1 \qquad \forall \lambda \in [0,1] \qquad (1)$$

$$b(\mu) = (1-\mu)b_0 + \mu b_1 \qquad \forall \mu \in [0,1] \qquad (2)$$

It holds that, given a point on *a*, if two points on *b* are reachable with valid shortcuts, all points in between them on *b* must be reachable as well. (See the full paper for a proof). So, to compute the cell we must specify two functions that give, for each value of λ, the maximum and minimum values for μ whose points we can reach with a shortcut. These min and max functions define the borders of the free space in our cell. To compute them, we construct two convex demi-hulls; one for the endpoints with minimum x-coordinate of all augments of vertices shaping the cell, and one for their endpoints with maximum x-coordinate. Shortcuts are valid if and only if their segments are fully within this convex *hourglass*, see Figure 5.

If we extend the line segments of these demi-hulls, we get a series of intersection points with *a* that indicate the values of λ where different segments of the hourglass become limiting for the minimum or maximum μ that can be reached. These are the breaking values for our piecewise min and max functions. For each piece it holds that either no points on *b* are reachable, all points on *b* are reachable, or the maximum/minimum point is given by shooting a ray from *a(λ)* through the limiting endpoint and finding the intersection between that ray and *b*.

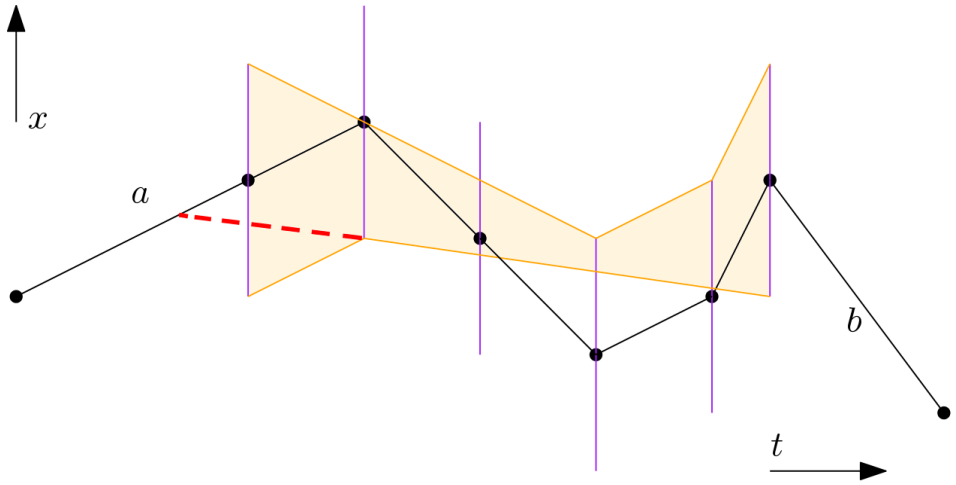

Figure 5. The hourglass consisting of two convex demi-hulls on the two sets of endpoints of the augments. Each demi-hull defines either the min or max function. If extending a segment of the demi-hull creates an intersection point with *a* (as can be seen with the dashed line), the value of λ at that point is a breaking value for that extreme function.

3. Discussion

The SSD seems to be a great tool for visualizing valid shortcuts for simplifying trajectories, but before it can be used in practice there are problems that must still be solved.

Firstly, we are currently investigating how to compute SSDs for two-dimensional trajectories. Because the augments are disks instead of line segments in this case, the hourglass construction does not trivially extend. For a given value of λ we can construct and solve the hourglass on the intersections of the augmenting disks and the plane defined by *a(λ)* and *b*. But we have not yet captured the changes in the hourglass as λ increases as a polynomial number of breakpoints for a piecewise function.

Secondly, the complexity of finding the optimal staircases of SSDs is an interesting open question. Aronov et al. proved that the general version is NP-hard. They do give a polynomial time algorithm for SSDs with certain properties. It is still a topic of study to see if SSDs based on the shortcuts of a trajectory have properties allowing a polynomial time algorithm.

Our current computation of the SSD is based on using the time-synchronized Euclidean distance metric. This is not a requirement for using the SSD however. By using a different error metric, research can also be done in using the SSD for continuous polyline simplification. It is also possible to use different spatio-temporal error metrics, such as a threshold on how much the simplified trajectory differs in direction or speed at every point in time. Of course, the SSD is not equally suited to all simplification problems. The method is based on combining shortcuts, and as Van Kreveld et al. (2018) proved, there are curves for which a shortcut-based algorithm is unable to find the optimal simplification under certain symmetric error metrics. Besides that, there may be many problems where using a SSD is not the fastest way to compute a solution.

Acknowledgements

Supported by NWO under the Commit2Data program.

References

Imai H and Iri M, 1986, Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31-41.

Aronov B, Driemel A, Van Kreveld M, Löffler M and Staals F, 2016, Segmentation of trajectories on nonmonotone criteria. *ACM Transactions on Algorithms*, 12(2):26.

De By R and Meratnia N, 2004, Spatiotemporal Compression Techniques for Moving Point Objects. *International Conference on Extending Database Technology*, Crete, Greece, 765-782.

McMaster RB and Shea KS, 1992, *Generalization in Digital Cartography*, Association of American Geographers, Washington D.C., USA.

Van Kreveld M, Löffler M and Wiratma L, 2018, On Optimal Polyline Simplification Using the Hausdorff and Fréchet Distance, *34th International Symposium on Computational Geometry*, 99:56:1-14

Zheng Y, 2015, Trajectory data mining: an overview, *ACM Transactions on Intelligent Systems and Technology*,6(3):29.