10. APPENDIX

10.1. Rigid stabbing polyline compression

As explained in Section 5, stabbing polyline based on n vertical line segments of length 2τ and n horizontal line segments of length 2η centered at original points in time-distance graph cannot guarantee TSND $\leq \tau$ and NSTD $\leq \eta$. A solution to solve this problem is to guarantee that i) the vertices of the stabbing polyline must be on the vertical line segments; and ii) the vertices of the stabbing polyline must be on the horizontal line segments. We will prove later that requirement i) is to guarantee TSND between the original polyline and the stabbing polyline will be bounded by τ, and requirement ii) is to guarantee NSTD between the original polyline and the stabbing polyline to be bounded by η. If we follow both requirements, it is obvious that the vertices of the stabbing polyline we look for must be located on both the vertical line segments and the horizontal line segments, i.e., the vertices of original polyline.

We plot two rigid stabbing polylines in Figure 23 and Figure 37 respectively. In Figure 23, the original polyline contains four vertices while the rigid stabbing polyline contains three vertices. In this example, we only consider TSND which explains the reason that we only plot vertical line segments centered at original vertices. If we only consider TSND, only requirement i) mentioned above needs to be considered and the vertices of the rigid stabbing polyline can be located anywhere along the vertical line segments, not necessarily being original vertices. In Figure 37, we consider both TSND and NSTD, and the rigid stabbing polyline, represented as a dotted polyline, contains only three vertices, i.e., p_1, p_3 and p_5 of original polyline.

The reason we introduce rigid stabbing polyline is that both TSND and NSTD between a temporal sequence represented as a polyline in time-distance space and a corresponding rigid stabbing polyline based on input τ and η are bounded by τ and η respectively, as stated in Theorem 10.1.

THEOREM 10.1. *Given a temporal sequence $\mathcal{TS}_T$ and a corresponding rigid stabbing polyline, it is guaranteed their TSND and NSTD are bounded by the tolerant errors τ and η respectively.*

PROOF. For convenience, we just prove that TSND between the stabbing polyline and the original polyline is less than the given tolerant error τ. The same procedure could be easily adapted to NSTD. Since the stabbing polyline is a rigid stabbing polyline, all vertices lie on vertical line segments and the polyline stabs all vertical line segments. Consequently, the stabbing polyline is divided into short stabbing line segments by vertical line segments.

Given a stabbing line segment between (d_i, t_i) and (d_{i+1}, t_{i+1}), suppose the intersections of the short line segment and the vertical line segments are (s_i, t_i) and (s_{i+1}, t_{i+1}), where $|d_i - s_i| \leq \tau$ and $|d_{i+1} - s_{i+1}| \leq \tau$. Meanwhile, the original polyline between t_i and t_{i+1} is the line segment that joins (d_i, t_i) and (d_{i+1}, t_{i+1}).

If the stabbing line segment intersects with the original line segment, TSND firstly decreases from $|d_i - s_i|$ to 0 and then increases from 0 to $|d_{i+1} - s_{i+1}|$. Otherwise, the stabbing line and the original one have no intersection, and TSND simply increases or decreases from $|d_i - s_i|$ to $|d_{i+1} - s_{i+1}|$. In both cases, TSND between (d_i, t_i) and (d_{i+1}, t_{i+1}) is bounded by τ. In other words, between any two consecutive time stamps, TSND of stabbing line segment and its original line segment is less than τ, and our proof completes. □

According to Theorem 10.1, we can compress the temporal sequences by constructing minimum rigid stabbing polylines. The rigid stabbing polyline is an extension of original stabbing polyline problem, where a new restriction is proposed, namely *turn in objects*, that each vertex of the polyline must be in the original objects. In the literature, a dynamic programming approach and a greedy algorithm have been proposed as solutions to solve this problem in [Guibas et al. 1991]. In our context, we can simplify both algorithms since the objects to stab here are horizontal or vertical line segments.

First, we introduce the dynamic programming algorithm that obtains a minimum rigid stabbing polyline based on visibility between two vertices. Two points p_1, p_2 of original polyline are visible to each other if the straight line segment from p_1 to p_2, denoted as $\overline{p_1 p_2}$, stabs all horizontal and

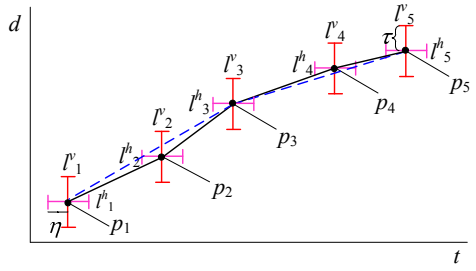
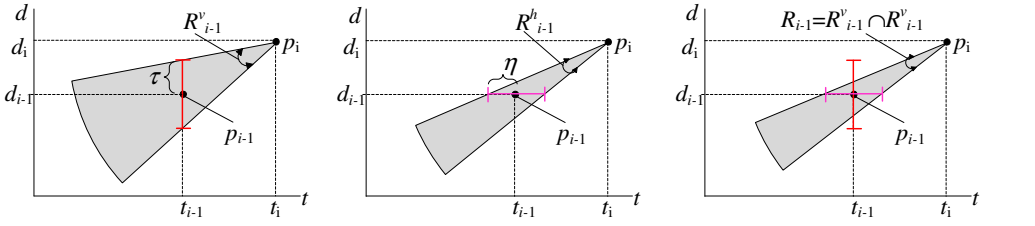


Fig. 37: Rigid stabbing polyline in time-distance space

vertical line segments between them. As shown in Figure 37, point p_1 and point p_3 are visible to each other, as the line segment $\overline{p_1p_3}$ stabs all the line segments that are between p_1 and p_3 (i.e., horizontal line segment l^h_2 and vertical line segment l^v_2). In our implementation, we employ a *visibility table*, denoted as Tab_v, to capture the visibility between two points of original polyline, where $v_{i,j}$ is $true$ if and only if p_i $(= (d_i, t_i))$ and p_j $(= (d_j, t_j))$ are visible to each other. Although the concept of visibility between two points is simple, and table Tab_v could be computed via brute-force approaches, we would like to introduce a novel data structure, namely *visible angular range* (VAR), which can significantly improve the performance of checking whether two points are visible.

In the following, we use an example to illustrate the concept of VAR. Take Figure 38 as an example, where we only plot two vertices (i.e., points p_{i-1} and p_i) of original temporal sequence in time-distance space and we want to decide whether p_j and p_i with $j < i - 1$ are visible. If p_j and p_i are visible, the corresponding line segment $\overline{p_jp_i}$ formed by them definitely stabs both the vertical line segment l^v_{i-1} and horizontal line segment l^h_{i-1} centered at p_{i-1}. In other words, the vertical line segment centered at p_{i-1} and point p_i bound an angular range R^v_{i-1} within which the line segment $\overline{p_jp_i}$ stabs the vertical line segment l^v_{i-1}, as depicted in Figure 38(a). Similarly, the horizontal line segment centered at p_{i-1} and point p_i bound another angular range R^h_{i-1} within which the line segment $\overline{p_jp_i}$ stabs the horizontal line segment l^h_{i-1}, as depicted in Figure 38(b). As our visibility requirement requires the line segment $\overline{p_jp_i}$ to stab both the vertical line segment l^v_{i-1} and horizontal line segment l^h_{i-1}, we have to consider both angular range R^v_{i-1} determined by vertical line segment l^v_{i-1} and the angular range R^h_{i-1} determined by horizontal line segment l^h_{i-1}. That means we need to consider the intersection between R^v_{i-1} and R^h_{i-1} to make sure that the line segment $\overline{p_jp_i}$ stabs both l^v_{i-1} and l^h_{i-1}, i.e., $R_{i-1} = R^v_{i-1} \cap R^h_{i-1}$ as shown in Figure 38(c).



(a) VAR R^v_{i-1} based on TSND (b) VAR R^h_{i-1} based on NSTD (c) VAR R_{i-1} based on TSND and NSTD

Fig. 38: Sample VAR

In the above example, we only consider the requirement that line segment $\overline{p_jp_i}$ must pass the vertical line segment and horizontal line segment centered at p_{i-1}. In the event that there are multiple points between p_j and p_i, we have to consider R_l corresponding to each point p_l with $j < l < i$ to make sure $\overline{p_jp_i}$ actually stabs all the vertical line segments and horizontal line segments between them. In other words, if p_j and p_i are visible, line segment $\overline{p_jp_i}$ must be within the angular range

$\cap_{j<l<i} R_l$. Since we can derive $\cap_{j<l<i} R_l$ for a point p_i in $O(n)$ time by scanning preceding points backward, the visibility table can be constructed in $O(n^2)$.

Assume the visibility table is ready, we now explain how to use dynamic programming to find a rigid stabbing polyline. Given a polyline of n vertices $p_1, p_2, \cdots, p_n$, let F_i be the minimum number of vertices contained by a rigid stabbing polyline joining the first i vertices (i.e., p_1, p_2, $\cdots$, p_i). Then, F_i can be derived based on Equation (18). The main idea is that if p_i and p_j with $j < i$ are visible, then the rigid stabbing polyline linking the first j points and the line segment $\overline{p_j p_i}$ form one candidate rigid stabbing polyline for the first i points. We evaluate all the points p_j located before p_i in order to find a rigid stabbing polyline with minimum number of vertices.

$$F_i = min_{v_{i,j}=true}^{j=1\cdots i-1} (F_j + 1). \tag{18}$$

Our algorithm based on dynamic programming to find a rigid stabbing polyline is listed in Algorithm 2. It first derives the visibility table by checking whether points (d_i, t_i) and (d_j, t_j) are visible, based on whether line segment $\overline{p_j p_i}$ falls within the visible angular range $\cap_{\forall l \in (j,i)} R_l$ (lines 1-10). It next derives the values of F_i (lines 11-16). It finally constructs the rigid stabbing polyline with the help of F_i (lines 17-20). In summary, the time complexity of the dynamic programming approach is $O(n^2)$, because F_i of each point can be computed in $O(n)$ time and there are n points in total.

Algorithm 2: Dynamic Programming based rigid stabbing polyline compression

Input: A temporal sequence $\mathcal{TS}_T = \{(d_1, t_1), (d_2, t_2), \cdots, (d_n, t_n)\}$
Output: a minimum rigid stabbing polyline $\{(d_{k_1}, t_{d_1}), (d_{k_2}, t_{k_2}), (d_{k_m}, t_{k_m})\}$

1: initialize visibility table Tab_v by setting all element $false$;
2: **for** each i from 1 to n **do**
3: $low \leftarrow \frac{\pi}{2}$; $high \leftarrow \frac{-\pi}{2}$; ▷ `form VAR`
4: $O \leftarrow (d_i, t_i)$; $P \leftarrow (d_i, t_i - 1)$;
5: **for** each j from $i-1$ downto 1 **do**
6: $Q \leftarrow (d_j, t_j)$;
7: $A \leftarrow (d_j + \tau, t_j)$; $B \leftarrow (d_j - \tau, t_j)$;
8: $C \leftarrow (d_j, t_j - \eta)$; $D \leftarrow (d_j, t_j + \eta)$;
9: **if** $max(high, \angle POA, \angle POC) \leq \angle POQ \leq min(low, \angle POB, \angle POD)$ **then**
10: $v_{i,j} \leftarrow true$;
11: $F_1 \leftarrow 0$; $prec_1 \leftarrow \emptyset$;
12: **for** each i from 2 to n **do**
13: $F_i \leftarrow \infty$;
14: **for** each j from 1 to $(i-1)$ **do**
15: **if** $v_{i,j} = true$ **and** $F_j + 1 < F_i$ **then**
16: $F_i \leftarrow F_j + 1$; $prec_i \leftarrow j$;
17: $R \leftarrow \emptyset$; $i \leftarrow n$;
18: **while** $i > 0$ **do**
19: $Append\_point(R, (d_i, t_i))$; $i \leftarrow prec_i$;
20: $Reverse\_the\_list(R)$;
21: **return** R;

Although dynamic programming generates a minimum rigid stabbing polyline, its time complexity is high, especially when the input trajectories are long. Unfortunately, there is no efficient algorithm to solve the rigid stabbing polyline problem. Motivated by this, we introduce a greedy algorithm, with its pseudo code listed in Algorithm 3. The main idea is to extend the current line segment as far as possible by scanning the points forward. Take the first point p_1 as an example. Assume $v_{1,2}$, $v_{1,3}$, $v_{1,4}$, and $v_{1,5}$ are true, but not $v_{1,6}$. When we start the scanning, we initialize a line segment from p_1. We then try to extend the current line segment to p_2, p_3, p_4 and p_5. However, when we attempt to extend the current line segment to p_6, it fails as p_1 and p_6 are not visible to each

other. In other words, the first line segment is formed by p_1 and p_5, and the second line segment is issued at p_5. Similarly, we try to extend the second line segment as far as possible to the farthest point that is visible to p_5. This process continues until the last point p_n is reached.

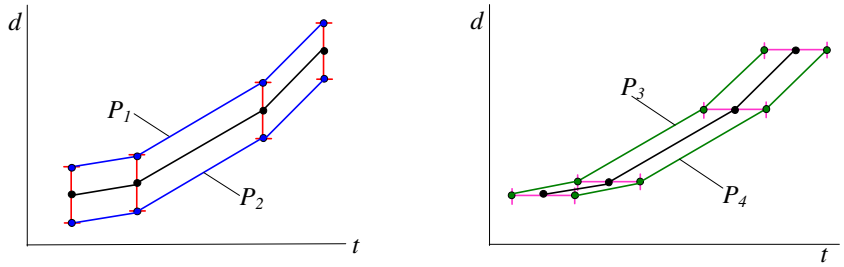
Algorithm 3: Greedy based rigid stabbing polyline compression

Input: A temporal sequence $\{(d_1, t_1), (d_2, t_2), \cdots, (d_n, t_n)\}$, the tolerant error τ and η
Output: A heuristic rigid stabbing polyline $\{(d_{k_1}, t_{k_1}), (d_{k_2}, t_{k_2}), \cdots, (d_{k_m}, t_{k_m})\}$

$R \leftarrow \emptyset$; $i \leftarrow 2$;
while $i \leq n$ **do**
  $Q \leftarrow (d_i, t_i)$;
  if $R \neq \emptyset$ **and** $low \leq \angle POQ \leq high$ **then**
    $A \leftarrow (d_i + \tau, t_i)$; $B \leftarrow (d_i - \tau, t_i)$;
    $C \leftarrow (d_i, t_i - \eta)$; $D \leftarrow (d_i, t_i + \eta)$;
    $high \leftarrow min(high, \angle POA, \angle POC)$; $low \leftarrow max(low, \angle POB, \angle POD)$; $i \leftarrow i + 1$;
  else
    $Append\_point(R, (d_{i-1}, t_{i-1}))$;
    $O \leftarrow (d_{i-1}, t_{i-1})$; $P \leftarrow (d_{i-1}, t_{i-1} + 1)$;
    $low \leftarrow -\frac{\pi}{2}$; $high \leftarrow \frac{\pi}{2}$;
$Append\_point(R, (d_n, t_n))$;
return R;

Obviously, the greedy algorithm scans each point once. If the current point p_i can be reached by the current line segment (i.e., p_i and the starting point of the current line segment are visible to each other), the scanning forwards. Otherwise, a new line segment is issued from p_{i-1}. In other words, it takes $O(n)$ time to construct a heuristic rigid stabbing polyline. Obviously, the greedy algorithm is much more efficient, as compared with the dynamic program algorithm. On the other hand, the greedy algorithm cannot guarantee the generated rigid stabbing polyline contains the minimum number of vertices. However, we want to highlight that the difference in terms of number of vertices between the rigid stabbing polyline generated by greedy algorithm and the rigid stabbing polyline generated by dynamic programming algorithm is small. A comprehensive comparison between these two algorithms will be performed via experiments. The greedy algorithm is an online algorithm that can process the points one by one without having the whole temporal sequence available. The dynamic programming algorithm can be an online algorithm too if we combine the computation of v and F together. To draw a conclusion, RSLC is an efficient algorithm and it is very useful in practice if optimal compression ratio is not a major consideration.

10.2. Proof of Theorem 5.11



(a) upper boundary P_1 and lower boundary P_2 of TSND tube P_d

(b) upper boundary P_3 and lower boundary P_4 of NSTD tube P_t

Fig. 39: Monotone polylines

PROOF. Since distance is a strictly increasing function of time, $P_1 = \langle(d_1 + \tau, t_1), (d_2 + \tau, t_2), \cdots, (d_n + \tau, t_n)\rangle$ (the upper boundary of the TSND tube), $P_2 = \langle(d_1 - \tau, t_1), (d_2 - \tau, t_2), \cdots, (d_n - \tau, t_n)\rangle$ (the lower boundary of the TSND tube), $P_3 = \langle(d_1, t_1 - \eta), (d_2, t_2 - \eta), \cdots, (d_n, t_n - \eta)\rangle$ (the left boundary of the NSTD tube) and $P_4 = \langle(d_1, t_1 + \eta), (d_2, t_2 + \eta), \cdots, (d_n, t_n + \eta)\rangle$ (the right boundary of the NSTD tube) are all monotone polylines (shown in Figure 39). Their distance functions are represented by $d_T(t) + \tau$, $d_T(t) - \tau$, $d_T(t + \eta)$ and $d_T(t - \eta)$.

More precisely, the upper boundary P_{up} of the compositive tube is given by the lower part of P_1 and P_3, i.e., $d_{up}(t) = min(d_T(t) + \tau, d_T(t + \eta))$. Meanwhile, the lower boundary P_{low} is given by the upper part of P_2 and P_4, i.e., $d_{low}(t) = max(d_T(t) - \tau, d_T(t - \eta))$. Given any t_0, since $(d_T(t_0) + \tau) - (d_T(t_0) - \tau) = 2\tau > 0$ and $d_T(t_0 + \eta) - d_T(t_0 - \eta) > 0$, P_1 and P_2 together with P_3 and P_4 do not intersect. Moreover, $d_T(t_0 + \eta) - (d_T(t_0) - \tau) > d_T(t_0 + \eta) - d_T(t_0) > 0$ and $(d_T(t_0) + \tau) - d_T(t_0 - \eta) > d_T(t_0) - d_T(t_0 - \eta) > 0$, so P_1 and P_4 as well as P_2 and P_3 do not intersect either. Consequently, $min(d_T(t_0) + \tau, d_T(t_0 + \eta)) > max(d_T(t_0) - \tau, d_T(t_0 - \eta))$ for all t. In other words, P_{up} is exactly above P_{low}. Then, we join P_{up} and P_{low} together with two short polylines $\langle(d_1, t_1 + \delta_1), (d_1, t_1), (d_1 + \delta_2, t_1)\rangle$ and $\langle(d_n - \delta_3, t_n), (d_n, t_n), (d_n, t_n - \delta_4)\rangle$ to prove that the compositive tube is a simple polygon.

Given two strictly increasing functions $f(x)$ and $g(x)$, the minimum $min(f(x), g(x))$ and the maximum $max(f(x), g(x))$ are strictly increasing functions too. Since we take polylines as strictly increasing functions, their minimum and maximum are both strictly increasing functions. In other words, both two boundaries of the composition tube are monotone polylines and the compositive tube is a monotone polygon. Our proof completes. □