Efficient Wind Speed Nowcasting with GPU-Accelerated Nearest Neighbors Algorithm

Arnaud Pannatier, Ricardo Picatoste, François Fleuret

April 28, 2022

Contributions

- Trajectory Nearest Neighbors (TNN) Algorithm.
- An extensive comparison with traditional approaches (linear search, KDTrees [Bentley, 1975].)
- Application : high-altitude wind nowcasting.
- Code and datasets are available at github.com/idiap/tnn

SKYSOFT ATM MALAT Wind Speed Dataset

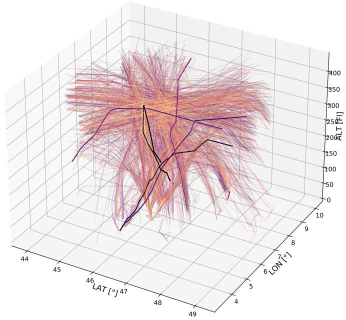


Figure: Measurements

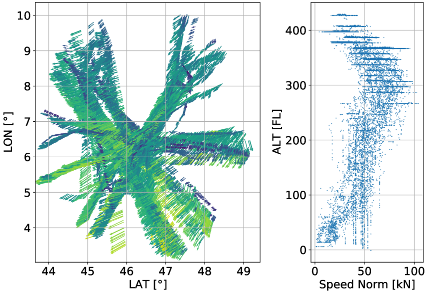
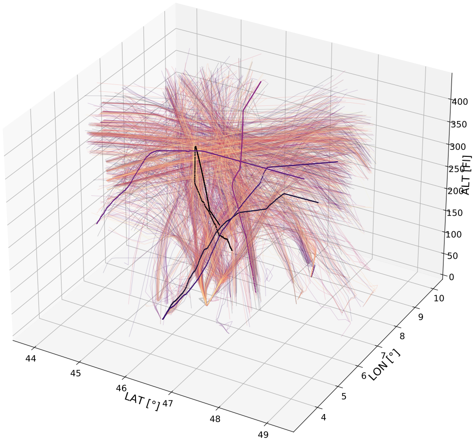


Figure: Wind Speed



SKYSOFT ATM MALAT Wind Speed Dataset

- Measures broadcasted every 4s
- Non-regular structure
- `https://www.idiap.ch/en/dataset/skysoft`

Wind Nowcasting

Last Hour Average

Context prediction

- Forecasts 30min ahead based on a context
- Here the context corresponds to the last hour of measure at our disposal
- E.g. all the measures taken between 9:00 and 10:00 → forecast at 10:30

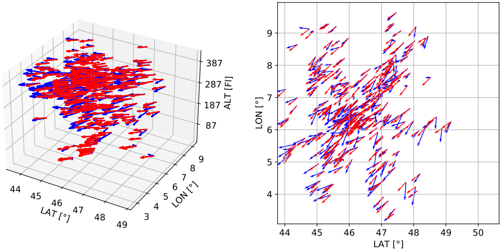


Figure: results

k Nearest-Neighbors (KNN)

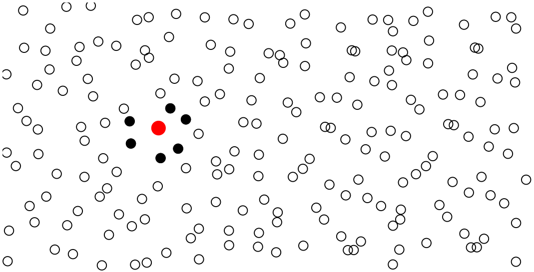


Figure: KNN

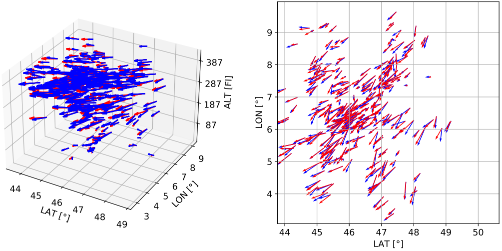


Figure: results

Gaussian Kernel Averaging (GKA)

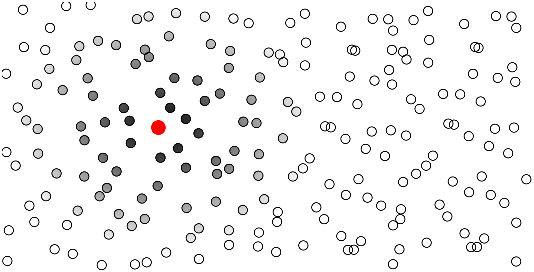


Figure: GKA

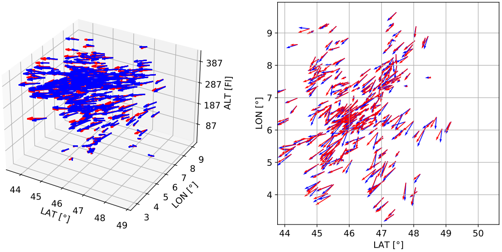


Figure: results

GKA-MLP [Pannatier et al., 2021]

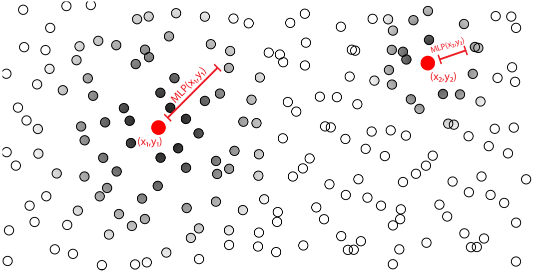


Figure: GKA - MLP

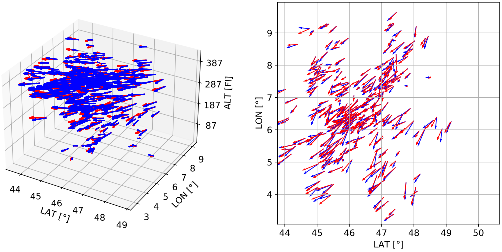


Figure: results

Results

Model	RMSE [kn] Day #1	Day #2	Day #3	Epoch duration 1 day dataset	5 weeks dataset
Mean wind	95 [kn]	49 [kn]	39 [kn]	hh:mm:ss	hh:mm:ss
Day Average	27,87	20,19	13,86	–	–
Hour Average	26,19	17,51	12,67	–	–
Particles [Sun et al., 2017]	9,98	10,07	7,84	–	–
GKA	9,07	9,64	7,66	–	–
k-NN | Persistence	9,02	9,86	7,57	–	–
GKA - TNN	8,71	9,19	7,55	–	–
GKA - MLP - TNN	**8,01**	**8,51**	**6,87**	–	–

Results

	RMSE [kn]			Epoch duration	
Model	**Day #1**	**Day #2**	**Day #3**	**1 day dataset**	**5 weeks dataset**
Mean wind	95 [kn]	49 [kn]	39 [kn]	hh:mm:ss	hh:mm:ss
Day Average	27,87	20,19	13,86	0:03	2:05
Hour Average	26,19	17,51	12,67	0:34	20:00
Particles [Sun et al., 2017]	9,98	10,07	7,84	6:57:15	1121:54:30
GKA	9,07	9,64	7,66	2:39:18	481:47:20
k-NN | Persistence	9,02	9,86	7,57	4:31:47	558:37:05
GKA - TNN	8,71	9,19	7,55	–	–
GKA - MLP - TNN	**8,01**	**8,51**	**6,87**	–	–

Results

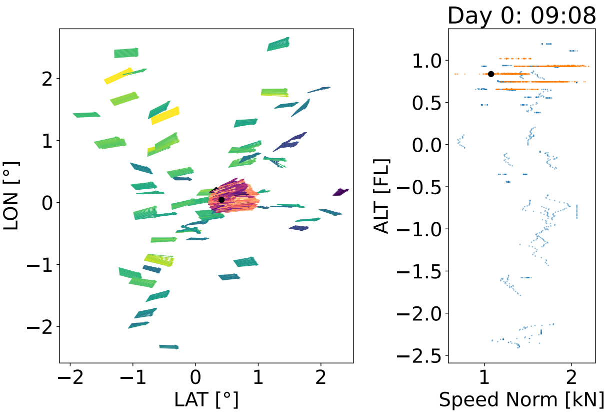
Model	RMSE [kn] Day #1	Day #2	Day #3	Epoch duration 1 day dataset	5 weeks dataset
Mean wind	95 [kn]	49 [kn]	39 [kn]	hh:mm:ss	hh:mm:ss
Day Average	27,87	20,19	13,86	0:03	2:05
Hour Average	26,19	17,51	12,67	0:34	20:00
Particles [Sun et al., 2017]	9,98	10,07	7,84	6:57:15	1121:54:30
GKA	9,07	9,64	7,66	2:39:18	481:47:20
k-NN | Persistence	9,02	9,86	7,57	4:31:47	558:37:05
GKA - TNN	8,71	9,19	7,55	–	–
GKA - MLP - TNN	**8,01**	**8,51**	**6,87**	–	–

We need to speed this up !!

TNN algorithm

How to select relevant context ?

- Restrict the set of measures to be efficient
- Should be *valid*, and *relevant*



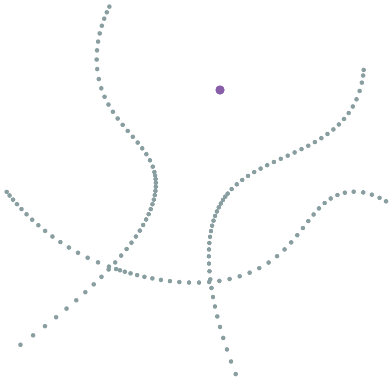
TNN algorithm

Algorithm 1: Trajectory Nearest Neighbors

Data: A batch of point, informations about the segments
Result: k-Nearest Neighbors
Distances = compute distances to segments ;
Distances = sort distances;
Furthest neighbors = ∞;
Next distance = Distances[:, 0];
i = 1;
d = 0;
while *Furthest neighbors* $\geq$ *Next distance or* $d = M$ **do**
  Fetch F segments of K points for the remaining ($M - d$) points in the batch;
  Compute distance from batch points to segments points;
  Current nearest neighbors = sort previous (k) and new points (FK);
  Furthest neighbor = Current nearest neighbors[:, k];
  d = nb of completed lines;
  Put completed lines (d) at the end of the batch ;
  Next distances = Distances[:, $i * F$];
  i += 1;
end
return *k-Nearest Neighbors*

Trajectory Nearest Neighbors (TNN)

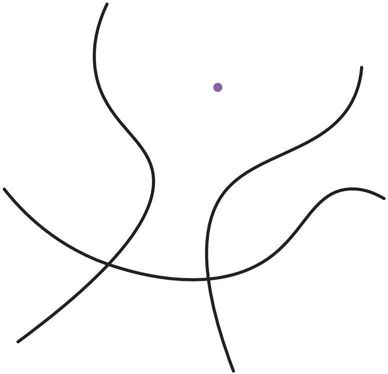
Simplified version – 3 neighbors, no batch



Starting with a query point and all the measurements

Trajectory Nearest Neighbors (TNN)

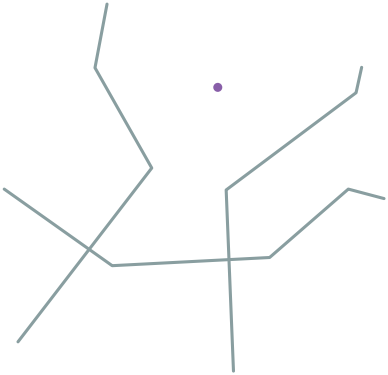
Simplified version – 3 neighbors, no batch



Consider trajectories as lines

Trajectory Nearest Neighbors (TNN)

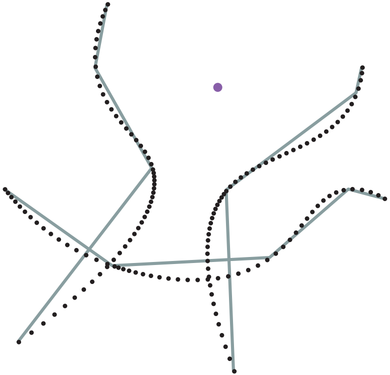
Simplified version – 3 neighbors, no batch



Approximate lines as segments

Trajectory Nearest Neighbors (TNN)

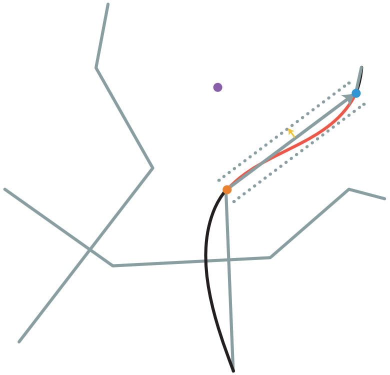
Simplified version – 3 neighbors, no batch



Measure the error made by the approximation

Trajectory Nearest Neighbors (TNN)

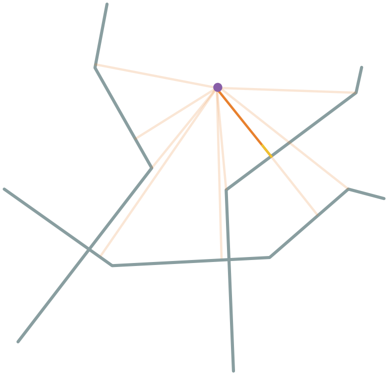
Simplified version – 3 neighbors, no batch



Measure the error made by the approximation

Trajectory Nearest Neighbors (TNN)

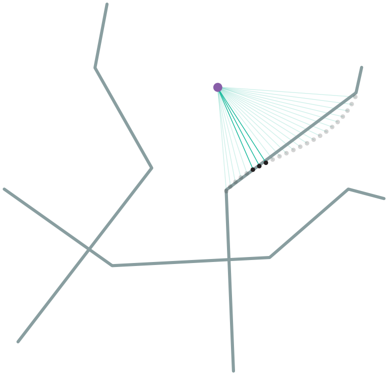
Simplified version – 3 neighbors, no batch



Select the closest segment

Trajectory Nearest Neighbors (TNN)

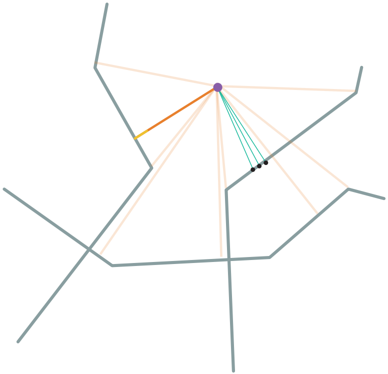
Simplified version – 3 neighbors, no batch



In the considered segment – take the k-nearest points

Trajectory Nearest Neighbors (TNN)

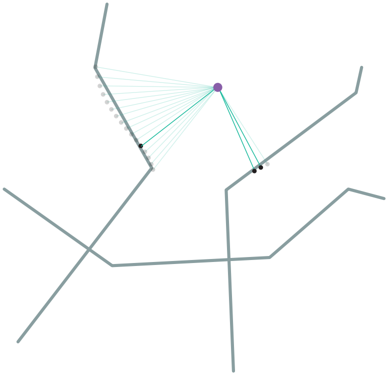
Simplified version – 3 neighbors, no batch



Select segments that are still closer than the farthest neighbor that we have seen

Trajectory Nearest Neighbors (TNN)

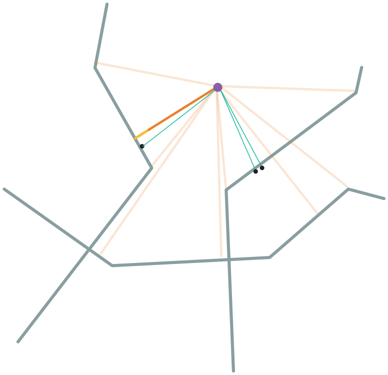
Simplified version – 3 neighbors, no batch



Search for neighbors

Trajectory Nearest Neighbors (TNN)

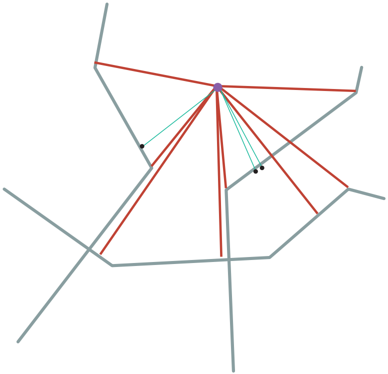
Simplified version – 3 neighbors, no batch



Continue

Trajectory Nearest Neighbors (TNN)

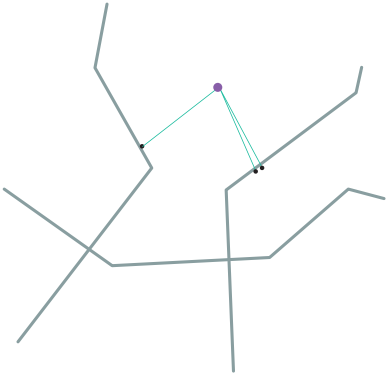
Simplified version – 3 neighbors, no batch



Until no segments can contain neighbors

Trajectory Nearest Neighbors (TNN)

Simplified version – 3 neighbors, no batch

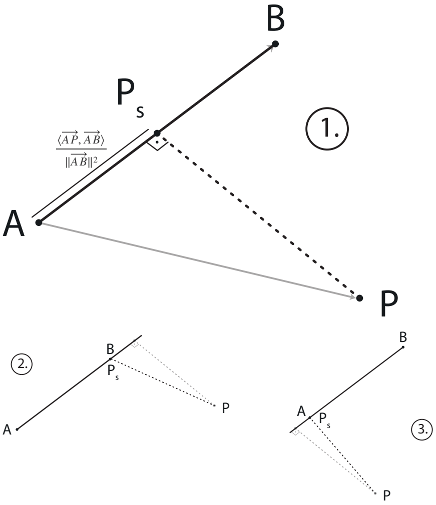


If all remaining segments are too far – we are over

Results

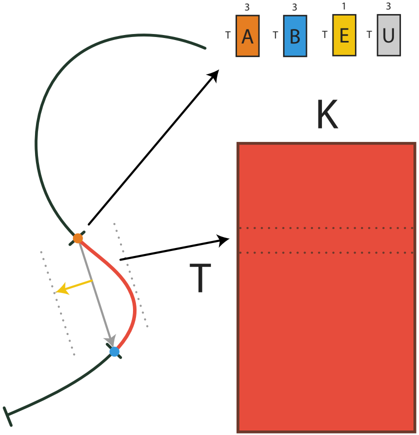
Model	RMSE [kn] Day #1	Day #2	Day #3	Epoch duration 1 day dataset	5 weeks dataset
Mean wind	95 [kn]	49 [kn]	39 [kn]	hh:mm:ss	hh:mm:ss
Day Average	27,87	20,19	13,86	0:03	2:05
Hour Average	26,19	17,51	12,67	0:34	20:00
Particles [Sun et al., 2017]	9,98	10,07	7,84	6:57:15	1121:54:30
GKA	9,07	9,64	7,66	2:39:18	481:47:20
k-NN | Persistence	9,02	9,86	7,57	4:31:47	558:37:05
GKA - TNN	8,71	9,19	7,55	4:13	1:35:30
GKA - MLP - TNN	**8,01**	**8,51**	**6,87**	4:21	1:37:39

Details



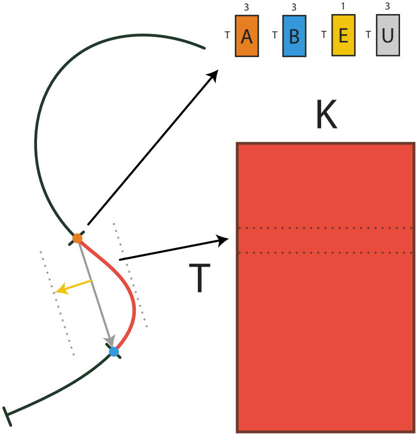
How to vectorize it ?

$$P_s = A + \max\left(0, \min\left(\frac{\langle \overrightarrow{AP}, \overrightarrow{AB} \rangle}{\|\overrightarrow{AB}\|_2^2}, 1\right)\right) \overrightarrow{AB}$$



How to vectorize it ?

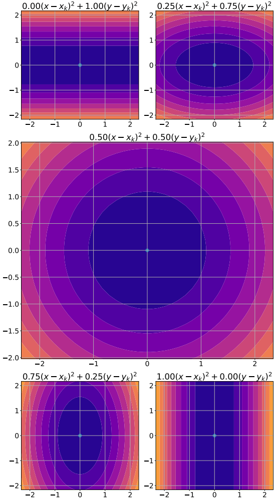
Data Structure



Algorithm 2: Distance from point P to a segment

Data: P, t,$\vec{\sigma}$, A, t_A, B,t_B,U, E, t_w
Result: Distance from P to segment with error
if $t_A > t - t_w$ **then return** ∞ ;
$\delta = \text{clamp}((P - A) \cdot U, 0, 1)$;
$P_s = A + \delta * (B - A)$;
$D = \|P_s - P\|^2_{\sigma_{xyz}} + \sigma_t \max(t_B - t, 0)^2$;
$D = D - E \max(\sigma_{xy}, \sigma_z)$;
return $clamp(D, 0)$

Allow to filter based on a coordinate (here time)



Details: Scalable metric

$$
\begin{aligned}
\|\vec{x}_1-\vec{x}_2\|^2_{\vec{\sigma}} =& \sigma_{xy}[(x_1-x_2)^2+(y_1-y_2)^2] \\
&+\sigma_z(z_1-z_2)^2+\sigma_t(t_1-t_2)^2
\end{aligned}
$$

Bibliography

Bentley, J. L. (1975).
Multidimensional binary search trees used for associative searching.
Communications of the ACM, 18(9):509–517.

Pannatier, A., Picatoste, R., and Fleuret, F. (2021).
Efficient wind speed nowcasting with gpu-accelerated nearest neighbors algorithm.

Sun, J., Vû, H., Ellerbroek, J., and Hoekstra, J. (2017).
Ground-based wind field construction from mode-s and ads-b data with a novel gas particle model.
In *Proceedings of the Seventh SESAR Innovation Days*.
7th SESAR Innovation Days, SIDs.

Math

$$\|\vec{x}_1 - \vec{x}_2\|^2_{\vec{\sigma}} = \sigma_{xy}[(x_1 - x_2)^2 + (y_1 - y_2)^2] + \sigma_z(z_1 - z_2)^2 + \sigma_t(t_1 - t_2)^2 \quad (1)$$

$$\|P_1 - P_2\|^2_{\sigma_{xyz}} = \sigma_{xy}[(x_1 - x_2)^2 + (y_1 - y_2)^2] + \sigma_z(z_1 - z_2)^2 \quad (2)$$

$$\|t_1 - t_2\|^2_{\sigma_t} = \sigma_t(t_1 - t_2)^2 \quad (3)$$

with $\vec{\sigma} = \left(\sigma_{xy}, \sigma_z, \sigma_t\right)$ sets: $T_j = \{\vec{x}_{j,1}, \ldots, \vec{x}_{j,K}\}$, $j \in \{1, \ldots, \frac{N}{K}\}$

$$d = \text{dist}((P, t), s) = \|P - P_s\|^2_{\sigma_{xyz}} + \|t - t_s\|^2_{\sigma_t} \quad (4)$$

$$P_s = A + \max\left(0, \min\left(\frac{\langle\overrightarrow{AP}, \overrightarrow{AB}\rangle}{\|\overrightarrow{AB}\|^2_2}, 1\right)\right)\overrightarrow{AB} \quad (5)$$

$$t_s = \max(t_A, \min(t, t_B)) \quad (6)$$

$$E_{\text{app}} = \max_{i \in 1,\ldots,K} \text{dist}(\vec{x}_i, s) = \max_i \|P_i - P_{s_i}\|^2_{\sigma_{xyz}} + \underbrace{\|t_i - t_{s_i}\|^2_{\sigma_t}}_{0} \leq \sigma_{max} \max_i \|P_i - P_{s_i}\|^2_2$$

with $\sigma_{max} = \max(\sigma_{xy}, \sigma_z)$, $t_A < t_i < t_B$ by construction.

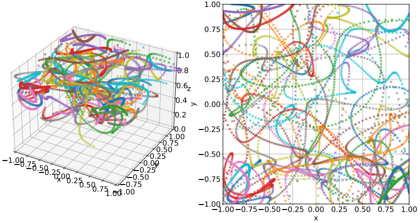
$$t_s^w = \begin{cases} \infty & \text{if } t_A > t - t_w \\ \min(t, t_B) & \text{otherwise} \end{cases} \quad (7)$$

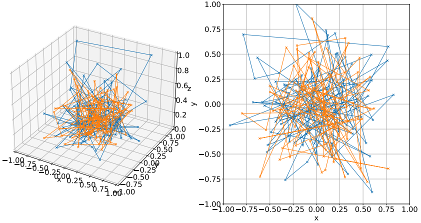
$$d_w = \|P - P_s\|^2_{\sigma_{xyz}} + \|t - t_s^w\|^2_{\sigma_t} \quad (8)$$

Complexity analysis

Method	Steps	Time Complexity	Space Complexity
Linear search	Distance Matrix	$O(N^2 D)$	$M(2N+1)$
	Top-k	$O(N^2)$	$2Mk$
TNN	Distance Segments	$O(\frac{N^2}{K} D)$	$M\frac{N}{K}D$
	Sort	$O(\frac{N^2}{K}\log(\frac{N}{K}))$	$2M\frac{N}{K}$
	Distance to points	$O(Nn_f FKD)$	$M(k+FK)D$
	Top-k	$O(Nn_f(k+FK))$	$2M(k+FK)$
	Total	$O(\frac{N^2}{K}\log(\frac{N}{K}) + Nn_f FKD)$	$M\frac{N}{K}D + M(k+FK)D$

Comparison with linear search





Comparison with linear search

Speed up in time compared to linear search

K \ F	10	20	50	100	200	500	750	1000	1250	1500
50	x0.1	x0.1	x0.4	x0.9	x1.9	x4.4	x4.8	x7.1	x6.4	x6.5
100	x0.1	x0.3	x0.9	x1.9	x3.6	x7.2	x7.2	**x9.3**	x5.9	x5.7
200	x0.2	x0.5	x1.4	x2.8	x4.7	x6.9	x5.0	x6.7	-	-
500	x0.4	x1.0	x2.2	x3.2	x3.9	x3.6	-	-	-	-

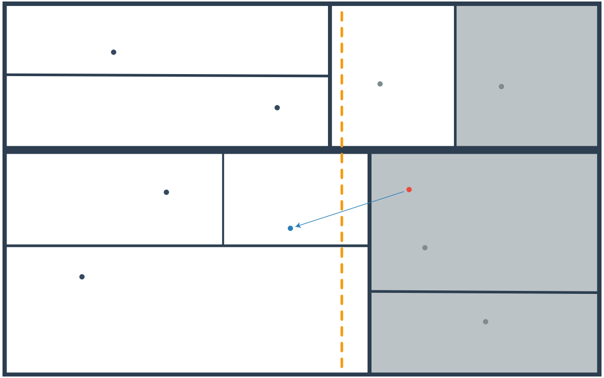
Fraction of comparisons needed compared to linear search

K \ F	10	20	50	100	200	500	750	1000	1250	1500
50	**2.3%**	2.3%	2.4%	2.6%	2.9%	4.2%	5.5%	6.9%	8.3%	9.8%
100	3.0%	3.1%	3.3%	3.6%	4.3%	7.1%	9.9%	12.8%	15.7%	18.8%
200	14.9%	15.1%	15.5%	16.1%	17.4%	21.6%	25.3%	29.6%	-	-
500	42.4%	42.7%	43.7%	45.2%	48.4%	58.4%	-	-	-	-

Comparison with linear search

Data set	Device	Algorithm	Comparisons	Query [ms]	Total duration
Original Data set	CPU	Lin. Search	811'372	9.03	2:30:32
		TNN	**28'579**	**1.03**	**17:08**
	GPU	Lin. Search	811'372	2.55	42:28
		TNN	81'611	**0.16**	**2:43**
SRW Data set	CPU	Lin. Search	1'000'000	11.95	3:19:09
		TNN	**58'408**	**1.60**	**26:35**
	GPU	Lin. Search	1'000'000	2.51	41:48
		TNN	93'555	**0.39**	**6:28**
Random points	CPU	Linear Search	1'000'000	**7.43**	**2:03:51**
		TNN	**999'940**	15.51	4:18:34
	GPU	Linear Search	1'000'000	1.72	28:42
		TNN	**998'588**	**1.36**	**22:40**

Comparison with KDTrees



Comparison with KDTrees

Method	Creation [s]	Query [ms]	Total duration
Linear search CPU	-	9.03	2:30:32
TNN CPU	1.00	1.03	17:08
Scaled masked KDTree	0.11	9.25	2:35:06
Scaled masked cKDTree	0.03	0.70	11:35
TNN GPU	7.00	0.16	2:43
Linear search GPU	-	2.55	42:28

Acknowledgement

Arnaud Pannatier was supported by the Swiss Innovation Agency Innosuisse under grant number 32432.1 IP-ICT – “MALAT: Machine Learning for Air Traffic.”