Using MDST-DBSCAN for creating Transaction Price Submarkets

This section covers how MDST-DBSCAN was applied to the data in order to create transaction price submarkets.

We should also note that we created a python implementation of the MDST-DBSCAN, as this algorithm was only implemented in R. The implementation can be found on GitHub. All the functions could be implemented with Numpy and Pandas. A python class object with functions was created. When using this implementation, firstly, one has to initiate the MDSTDBSCAN object with the threshold-parameters eps, eps2, eps3 and the number of minimum points minpts. After this you can run the algorithm with run, where one has to input the data, with x and y coordinates, the time column in datetime-format and the value column as a pandas series. Another function that was implemented is to plot the clusters.

MDST-DBSCAN highly depends on the parameters used. Hence, it is necessary to try different parameters. That is why grid search was performed to find the best hyperparameters eps, eps2, eps3 and minpts. Therefore, different values for the hyperparameters and combinations of these were applied. For each combination the MDST-DBSCAN algorithm was computed and the best parameters were saved. These are defined by a metric, which in this case was the Dunn-Index and Davies-Bouldin. All hyperparameter combinations were logged to a csv-file, in order to save the results for later times. The grid search was carried out for different ranges of hyperparameter values and the aim was to start with relatively big ranges. Later these were narrowed down to smaller steps and ranges to find the optimum values.

The number of minpts and the Epsilon-values were defined using visual interpretation or in other words domain knowledge. Additionally, a maximum number of clusters and maximum number of noise points were defined. The first component was necessary, as smaller clusters show spatially, pricewise more homogeneous areas. However, the aim is to form clusters that are not too narrow and show similar characteristics within. The maximum number of noises accepted had to be defined, as a small number of clusters and a good index score often showed a high number of noise - which is not purposeful. Therefore, the number of maximum clusters was defined with 30 and the number of noise accepted with 2700 data points.

After performing multiple runs and fine-tuning the values with grid-search, the following hyperparameters were used for clustering.

- eps: 0.0135
- eps2: 850
- eps3: 1450
- minpts: 55

Even though it is not implemented in the algorithm itself, multiple runs were performed to improve the sorting of the dataframe. For every run, the data was shuffled and the spatio-temporal metric was computed for each dataframe. 50 runs, therefore 50 different sortings, were computed and the best clusters were saved.

In a next step the output of the clustering algorithm was post-processed. Construction projects should only be assigned to one cluster. However, as every transaction price in the algorithm is treated as an own object, this requirement could not be met by the algorithm itself. Therefore, for all projects with more than one cluster-mark, the cluster with the highest number of

transaction prices in this project was assigned to all the other dwellings in this project. Hence, every project only was assigned to one cluster. Another benefit that comes along with this is that it strengthens the robustness of the algorithm. This step makes the algorithm less sensible to the first point that is used in the method, as the impact of single data points is reduced. However, this disadvantage cannot be eliminated completely by this step. Furthermore, the count of noise is also reduced through this step. Overlapping clusters are not completely eliminated, without changing the algorithm itself. However, this step reduced the overlapping cluster areas. This is the biggest achievement of this step.

Results of MDST-DBSCAN

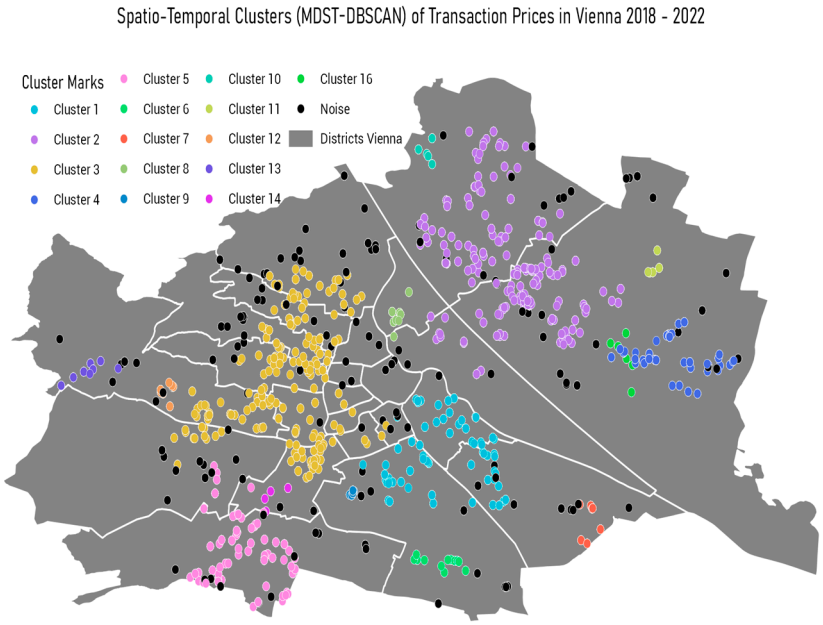
Table S1shows the clusters and the number of data points per cluster, before and after post-processing (grouping by project-id). As you can see the number of noise reduced dramatically through this step from 1958 to 1299. There is a trend visible, that the clusters with lower numbers have higher counts, while those with higher numbers, for instance Cluster 17, show a lower number of data points. One possible reason for this could be the known dependency of this clustering algorithm on the initial point. All clustering runs showed similar trends of many data points being assigned in the clusters with low numbers. The comparison of the counts before and after post-processing indicate, that these small clusters were already there after the clustering algorithm and not produced by processing. These small clusters could be deleted by increasing the minpts-parameter. However, this would increase the number of noise respectively.

Table S1: Cluster values MDST-DBSCAN (count = before processing and count 2= after processing).

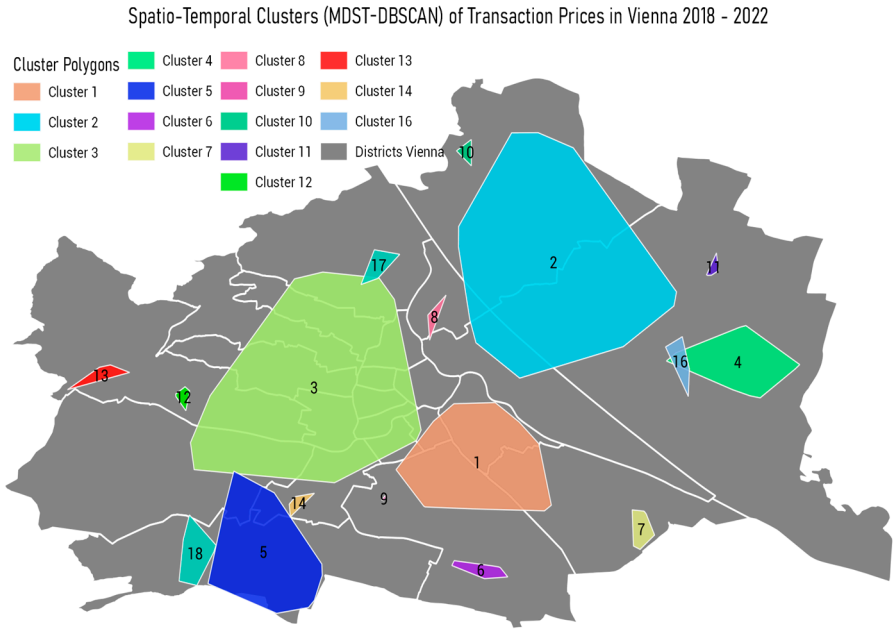
Cluster	Count	Count 2
Noise	1958	1299
Cluster 1	2839	3017
Cluster 2	5094	5304
Cluster 3	4252	4479
Cluster 4	995	1001
Cluster 5	2114	2164
Cluster 6	628	632
Cluster 7	179	192
Cluster 8	221	232
Cluster 9	360	362
Cluster 10	150	150
Cluster 11	123	123
Cluster 12	106	112
Cluster 13	121	127
Cluster 14	76	78
Cluster 16	60	80
Cluster 17	56	35
Cluster 18	84	90

Figure S1 represents the best output of the MDST-DBSCAN algorithm for the epsilon values mentioned above, after executing 50 runs. **Figure S1a** shows all the projects and their corresponding cluster mark, while **Figure S1b** shows the created polygons. Especially the

latter one shows that the size of the polygons differs a lot. There are some very big clusters, such as Cluster 2 and Cluster 3, and on the other hand numerous very small clusters, such as Cluster 9 and Cluster 8.



A) Cluster Marks



B) Cluster Polygons

Figure S1: Spatio-Temporal Clusters identified by MDST-DBSCAN for Transaction Prices in Vienna 2018 – 2022

Results on number of clusters, noise and spatio-temporal metric of MDST_DBSCAN & MDSTC-DBSCAN

Table S2 presents a comparison of the score of the metric, the number of noise points, and the number of clusters for both algorithms when varying the initial point selection (for 9 runs). The number of clusters shows a greater variation for MDST-DBSCAN, ranging from 12 to 21 clusters. On the other hand, MDSTC-DBSCAN exhibits a narrower range of variation in the number of clusters created, with values ranging between 13 and 17 clusters. Also, MDST-DBSCAN tends to receive a much higher number of noise in all the runs (MDST mean =2069.40 and MDSTC mean = 1195.60).

For MDSTC -DBSCAN the third run received the lowest score for the metric, the lowest number of noise points and resulted in 16 clusters. Important to notice is that the number 16 occurred in 5 of the 9 runs. However, it should be noted that there are outliers in the metric's results, as runs 4 and 5 exhibit scores above 4, while the others have significantly lower values. Thus, when comparing the algorithm's performance using our spatio-temporal metric, MDSTC -DBSCAN shows more variable results (MDSTC SD = 0.87 and MDST SD = 0.17).

Table S2*:* Comparison of initial point chosen for MDSTC-DBSCAN and MDST-DBSCAN

Run	MDSTC-DBSCAN			MDST-DBSCAN		
	Score of Metric	Number of Noise	Number of Clusters	Score of Metric	Number of Noise	Number of Clusters
0	2.69	1267	17	2.24	2083	14
1	2.25	1248	15	1.75	2217	13
2	2.32	1268	17	2.01	1985	16
3	**1.72**	**960**	**16**	2.07	1935	13
4	4.16	1194	17	1.88	2175	16
5	4.15	1256	16	1.81	2102	12
6	1.95	1157	16	1.91	2048	21
7	2.23	1238	16	1.96	2165	13
8	2.98	1209	16	2.15	1796	20
9	2.05	1159	13	1.7	2188	13
Mean	2.65	1195.6	15.90	1.95	2069.4	15.10
Median	2.29	1223.5	16.00	1.94	2092.5	13.50
SD	0.87	92.53	1.18	0.17	132.4	3.14

Problems with MDST-DBSCAN for creating Transaction Price Submarkets

MDST-DBSCAN is highly affected by the parameter and initial point chosen for the algorithm. This was also detected in our analysis as the outputs changed when slightly changing the parameters. However, this effect can be reduced by performing more runs and investigating a wide range of parameters.

MDST-DBSCAN method takes thresholds for the spatial, temporal and value dimension. If a point meets the thresholds and minimum number of points (minpts), this point and its neighboring points are assigned to the same cluster. If the time span of the points is greater than eps3 (time), more than one cluster exist for the same spatial area. For instance, if the temporal distance between point i and its nearest neighbor at a later time stamp is further away than eps3, these two points are not connected. Especially with the data used in this thesis, such clusters can occur as the data highly depends on the building year of the houses respectively

the transaction date. For instance, there could be areas where only two projects exist – one with a completion year at the beginning of the time span and one at the end of the time span. However, when forming submarkets, the aim is to form spatially contiguous clusters that show polygons in a 2D area with similar prices/attributes. Hence, there should only be one cluster per spatial location/area. This can be achieved if using a two-clustering method (i.e. space & attribute are the clustering dimensions) but then information on time is completely ignored.

The objective of submarket analysis is to incorporate both spatial and temporal dimensions as valuable components of the data. Thus, regarding the temporal attribute, treating it as a clustering dimension (MDST-DBSCAN) or not including at all (two-clustering like Spatial Kmeans) would potentially lead to loss of important information, as seen above. Therefore, the aim is to build as stable and informative clusters as possible but only one cluster per location. One approach to achieve this with MDST-DBSCAN could be to increase the value of eps3. However, this would mean decreasing the impact of the temporal dimension on clustering. For example, here we used the temporal threshold of 850 days which is rather high. These high thresholds led to some very big clusters with a lot of data points. On the other hand, there are spatially very small clusters for the rest of the data. As a result, the aim of not creating too many overlapping clusters could be achieved by accepting less accurate and meaningful results.

This experimental analysis using the MDST-DBSCAN algorithm, showed us that it is not suitable for delineating submarket prices and an effort to use it by tuning the parameters to fit our scope resulted in not meaningful clusters. Therefore, we concluded that considering change of values in time (using linear regression) would better fit our purpose, which in turn resulted in the development of the MDSTC-DBSCAN algorithm presented in the paper.