Performance Evaluation of Clustering Approaches with Spatial Databases

Simon Braito Gregor Mayr David Weissteiner

October 26, 2023

1. Introduction

Geospatial Clustering is a well known method in Data Analysis to reveal relationships between observed features and the corresponding spatial attributes. Although these clustering methods are well studied and often applied, the majority of Spatial Database (SDB) Systems do not offer ad-hoc clustering methods. In the discipline of Spatial Data Analysis, a common practice is to perform the actual clustering on a separate computational site using additional software. For convenience, often the clustering is conducted using a Geographic Information System (GIS), since it is more user-friendly.

When clustering spatial data on a separate computational site, it is necessary to transfer the entire dataset from the SDB to the computational site and return the resulting clustered data to the database. We identify a huge overhead in performance and memory consumption arising from these data transfers. Such additional overheads result in a heavily increased time and memory consumption that sometimes even causes timeouts or makes it infeasible to perform the clustering due to bandwidth and memory limitations. Especially in big data applications, we do not have the capacity for a complete mirror of the entire dataset.

In this paper, we study the resource requirements of integrated and external clustering approaches with SDB Systems. Furthermore, we evaluate the performance and memory consumption needed for clustering spatial data in three different settings:

1. Integrated Clustering in SDB System
2. External Clustering with a Geographic Information System (GIS)
3. External Clustering with a Machine Learning (ML) System

We test the performance and memory consumption needed for clustering spatial data in the three different settings using a test suite which was conceptualized for this research by us and which is available on Github[1].

Our Contributions are:

1. *Best-Practice Investigation*: We evaluate three commonly applied practices used for clustering spatial data and compare them in terms of performance and memory consumption. The experimental results indicate which of these three practices performs the best.
2. *Limitations*: The additional network transmission in external clustering approaches creates a huge overhead in performance and memory consumption, and hence could make the clustering task infeasible in a big data scenario due to bandwidth and memory limitations. Our experimental evaluations will give a rough indicator about the emerging additional resource consumption and the implicated limitations.
3. *Testing Framework*: We provide an extensive test suite which implements all three mentioned clustering approaches and is able to evaluate their performance and memory consumption for reproducibility.

2. Background

Geospatial clustering is an important technique from Spatial Data Analysis to find groups or clusters with similar characteristics in georeferenced data. In general, there are different clustering algorithms available to solve such a task (see [1] for a comprehensive survey on clustering algorithms). GIS software usually comes with built-in functions

[1] https://github.com/ywcb00/SDB-ClusteringApproachEvaluation

to perform cluster analysis. Mostly, the clustering algorithms k-Means [2] and DBSCAN [3] are implemented in GIS applications, since these algorithms emerge as predominant in practice.

In order to store very large spatial datasets, databases with geographic data support have evolved. Due to the extensive emergence of Spatial Data Analysis in various fields, the leading database systems have either native support for geographic data processing, or can acquire support for spatial data through an extension (e.g., PostGIS [4] in PostgreSQL [5]). This support enables the representation of geospatial data inside the database, and it also provides tools for querying and analysing spatial data.

The advantages of SDB systems as a data management backend in combination with the user-friendly benefits of GIS applications led to built-in interfaces for popular SDB systems in many GIS applications. Vice versa, general analysis functions of GIS applications are often also implemented directly in the SDB systems. Although these methods are integrated in the SDB systems, they are mostly unused because the analyses are conducted with a GIS software, which uses its own analysis methods, even though the equivalent methods exist in the SDB system, as the GIS software is more user-friendly.

Benchmarks to compare the performance of different SDB systems such as [6] are already available, however to the best of our knowledge there do not exist benchmarks for evaluating spatial clustering routines in SDB systems specifically. Furthermore, we were not able to find literature on comparing clustering approaches within an SDB against external approaches.

3. Methodology and System Design

The goal of this paper is to evaluate and to compare common ways of clustering spatial data from a database. Therefore, we set up an SDB system to store input datasets and the resulting clustered data. Using this SDB we then create clusters using the following three spatial clustering approaches:

1. Integrated Clustering in SDB
2. External Clustering with GIS
3. External Clustering with ML System

In order to allow for a good comparison, all three of these approaches deal with the same input data (i.e. spatial data from the database) and produce the same output (i.e. clustered data, stored on the database). To obtain more conclusive results, we use multiple different datasets and two different clustering algorithms[2] (namely k-Means and DBSCAN).

Given that the performance measurements (especially the runtime performance) might vary across different executions of the same experiment, we perform the evaluations on synthetic datasets several times and draw an average of the corresponding measurements to obtain a good representation of the actual resource requirements.

Another important aspect to consider, in order to provide a fair comparison of the approaches, is the modality of how the performance is measured. For example, one could use pgAdmin (a PostgreSQL administration tool) to measure the performance of spatial SQL queries and use a QGIS performance analysis tool to measure the performance of the approach using QGIS. However, this approach would mix different measurement modalities and would likely skew the results of the analysis. Instead, we build a test suite that is able to invoke the different approaches and measure the end-to-end time of each approach.

Note that this analysis does not include methods for validating the quality of the produced clusters, such as the Dunn Index or Silhouette Scores. We assume that each approach provides a correct implementation of the algorithms with equally accurate results.

The remainder of this chapter is structured in the following way: Section 3.1 gives an insight on the underlying database system, which is used to store the input data and output data in all three approaches. In Sections 3.2 - 3.4, we describe the design of the three evaluated approaches and highlight their differences.

3.1. Spatial Database System

We set up a local PostgreSQL [5] database system as the base infrastructure for our evaluations. Furthermore, we add the PostGIS [4] extension to our PostgreSQL instance to support geographic objects. Since the database is on the same node as our test suite, we do not consider the impact of a network connection to our evaluations. However, we emphasize that some of the approaches include an additional bottleneck that arises from large network transmissions with a limited bandwidth in a real-world deployment.

[2]Note that the choice of clustering algorithms was limited to the available clustering algorithm implementations in PostGIS (see Section 3.2)

3.2. Internal Clustering in SDB System

The PostGIS extension from our SDB system offers built-in methods to cluster spatial data using both the DBSCAN [7] and the k-Means [8] algorithm. We use these built-in methods to perform the clustering procedure in the database system itself, and thereby evaluate the performance of an integrated clustering approach inside the SDB system.

3.3. External Clustering w/ GIS

In common practice, the spatial database system might solely be used as a storage management unit while the data analysis is performed using external software. Given that spatial data analysis is frequently performed by analysts who are familiar with GIS software, these applications are often the first choice for a common task such as cluster analysis. Additionally, GIS software offers the advantage of a more user-friendly way of operation and a better visualization of the spatial data. However, to operate on the data, the GIS software has to fetch the whole dataset from the database.

In our evaluations, we use the open source GIS software QGIS [9]. The QGIS project provides a built-in interface to PostgreSQL and the PostGIS extension, allowing us to work seamlessly with our database setup. Additionally, QGIS offers a clustering method with the DBSCAN [10] and the k-Means [11] algorithm. In order to integrate this approach into our test suite, we can utilize the QGIS processing framework [12]. This allows us to invoke the QGIS primitives from a Python script such that we can easily cluster spatial data from our database and evaluate its performance.

3.4. External Clustering w/ ML System

Since clustering techniques originally come from the field of Machine Learning (ML), we investigate the performance of clustering spatial data when using an external ML system. For this reason, we implement a python script to handle the interface between our database system and the ML system. In this python script, we access our PostgreSQL instance using the `psycopg` package [13] to obtain the input data from the database. Then, we perform the clustering task with `scikit-learn` [14], a free python ML library. The contained module `sklearn.cluster` offers well-optimized methods for clustering with k-Means and DBSCAN. Finally, we transfer the resulting clustered data back to the database using the `psycopg` python package.

4. Implementation

To test each clustering approach extensively, a lot of data with different sizes is needed in order to get a good overview. For this, we considered real world datasets, but also generated synthesized data, which we explain in more detail in Section 4.1. In Section 4.2 we provide a better overview of our test suite that evaluates the performance and memory consumption of all 3 mentioned approaches. Lastly, in Section 4.3 we provide information about the system environment in which the tests were conducted. For reasons of conciseness, in this paper we omit more involved implementation details and refer to the corresponding GitHub repository[3].

4.1. Datasets

For our tests, we used 2 types of datasets:

- Real datasets from real world data.
- Synthesized datasets, which are generated procedurally.

The real datasets are as follows:

- Listed Buildings provided by the National Heritage List for England (NHLE) [15],
- Registered Business Locations - San Francisco provided by the City and County of San Francisco [16].

For the synthesized datasets, we procedurally generated both 2D and 4D data, which have the further ability to adjust the size of the dataset. This allows us to test a high range of dataset sizes without the need to search for appropriate real world datasets. Both synthetic datasets are generated by constructing multiple multivariate Gaussian distributions with different variance/co-variances. For the 4D case, the resulting Gaussian blobs are not necessarily isotropic. Further information regarding the datasets used in this paper is available in Table 1.

[3] `https://github.com/ywcb00/SDB-ClusteringApproachEvaluation`

Dataset	Size(s)	Type
Listed Buildings	300368	real
Registered Business	379345	real
2D	[50k-1000k]	synthesized
4D	[25k-1000k]	synthesized

Table 1: The datasets used in our paper. The size(s) of a dataset represents the number of points each dataset contains, if multiple sizes are available, the ranges will be marked in brackets [].

For the 2D dataset, we synthesized several sizes to later test, how the performance relates to an increase of the dataset size. For *KMeans* we chose the following sizes: 200k,400k,600k,800k,1000k. For *DBSCAN* we chose: 50k,100k,150k,200k,250k.

On the other hand, for the 4D dataset, we chose again: 200k,400k,600k,800k,1000k for *KMeans*, and for *DBSCAN*: 25k,50k,75k,100k and 125k. The varying size ranges between algorithms was chosen due to their clustering efficiency, additionally, the 4D dataset size ranges for *DBSCAN* was only half due to the increase of 2 dimensions of the data points themselves.

4.2. Test Suite

To evaluate the effectiveness of all three approaches, we developed a comprehensive test suite. This test suite is written in the Python programming language and allows us to test the performance and memory consumption of all three approaches in an orderly and systematic manner, with as little external influences as possible. Due to the design of this test suite, we also limit errors which could occur due to the use of several different software programs, which would be needed to execute all three approaches otherwise. Additionally, the test suite allows for a quick and easy access to different datasets, which can be switched out for conducting different tests.

In order to create, update and select data from the PostGIS database, this test suit utilizes the packages `psycopg2` [13], `sqlalchemy` [17] as well as `geopandas` [18]. The integrated SDB clustering approach uses the integrated spatial SQL functions *ST_ClusterKMeans* and *ST_ClusterDBSCAN* to directly cluster data on the database using SQL statements, which are provided in Appendix A. For the external ML clustering approach, the classes `KMeans` and `DBSCAN` from the `sklearn.cluster` package were used. Finally, the external approach using QGIS uses the `PyQGIS` [19] interface to access the QGIS API Python interface. Here, the algorithms `native:kmeansclustering` and `native:dbscanclustering` were used to cluster the data. Furthermore, `PyQGIS QgsVectorLayer` object can be configured to internally retrieve and push data from and to the database, hence this approach was utilized.

To measure the runtime performance of the approaches, we measured the time exceeded during three different phases:

1. *Preprocessing*: In this phase data was loaded from the database. For the integrated clustering approach, everything was executed in the database system, hence there was nothing to be done here. For the other two approaches, the data had to be fetched from the database first.
2. *Processing*: This phase consisted of running the actual clustering algorithm.
3. *Post-processing*: In this last phase, the results of the clustering algorithm were stored in the database. Again, for the integrated approach, nothing had to be done here.

In order to evaluate the memory consumption of the different approaches, we measured the maximum memory consumption of the Python and the Postgres processes. As with measuring the runtime performance, here we differentiated between the different phases of the test we enumerated above. For this task, we utilized the `proc` package [20], which, citing the package documentation, exposes process information available in the Linux process information pseudo-file system available at */proc*.

4.3. System

To receive accurate results in our measurements which are unaffected by external influences (*i.e.*, programs which are running in parallel, using up resources), all tests were conducted on the same system with as little programs running at the same time as the tests were executed (to our power of possibility). As this factor plays a major role in the measurements, we tried to exclude external influences to the performance as much as possible and executed the tests with a clean system. Moreover, to average out outliers, we executed and recorded each test case multiple times.

We conducted all of our experiments on a Lenovo ThinkPad Yoga X1 Gen 3 laptop with the following hardware specifications:

- *CPU*: Intel(R) Core(TM) i7-8550U 1.80GHz with 4 physical cores and 2 threads per core
- *RAM*: 2×8GiB LPDDR3 RAM 2133MHz
- *Storage*: 512GB Samsung SSD

Furthermore, we perform our experiments on top of the following software:

- *OS*: Ubuntu 22.04.1 LTS
- *Python*: v3.10.6
- *PostgreSQL*: v14.6
- *PostGIS*: v3.3.2
- *QGIS*: v3.28
- *sklearn*: v1.2.0

It is important to note, that the results can change heavily depending on the hardware used to conduct these tests, and we therefore chose a system configuration similar to a normal workstation, which simulates a normal environment of the most common users. Further research can be made by conducting the tests on more powerful and varying equipment, as the influence of hardware was not tested in our research.

5. Results

This section presents the results obtained from the experiments conducted in this paper.

5.1. Experiments with Synthetic Datasets

In this section, we present the results of our evaluation from the experiments with synthetic datasets. We considered 2-dimensional and 4-dimensional datasets for the clustering tasks with 200k up to 1000k data points for the experiments using the kMeans clustering algorithm and 50k up to 250k points for the experiments with the DBSCAN algorithm. When we put all the configurations together, we obtain the following clustering tasks:

1. Cluster 2-dimensional data with size {200k-1000k} using kMeans
2. Cluster 2-dimensional data with size {50k-250k} using DBSCAN
3. Cluster 4-dimensional data with size {200k-1000k} using kMeans
4. Cluster 4-dimensional data with size {50k-250k} using DBSCAN

We conducted all of the clustering tasks for each of the three approaches under test to obtain the performance measurements. Furthermore, we ran every experiment five times and computed their average to obtain a relatively stable performance indication.

Figure 1 presents the execution times of clustering task #1 for each of the approaches in a bar chart. Note that each of the bars is further divided into "pre", "main", and "post" sections that indicate whether the time is spent on the actual clustering task —"main"—, on preprocessing before getting to the clustering —"pre"— (e.g., getting the data from the database), or on post-processing after the data has been clustered —"post"— (e.g., storing the cluster indices to the database). Figure 2 shows the corresponding maximal memory consumption separated into the memory consumption of the PostgreSQL instance and the memory consumption of the Python program. Note that the memory consumption of the Python program includes the consumption of the QGIS clustering.

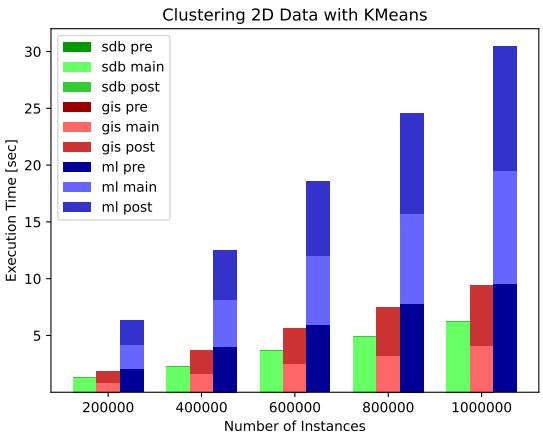

Figure 1: Execution Times - 2-dimensional kMeans

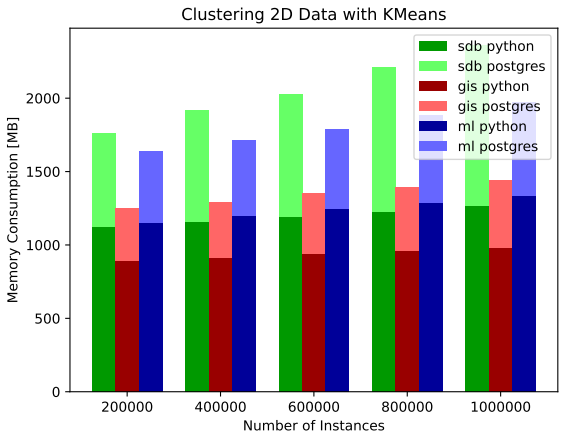

Figure 2: Maximal Memory Consumption - 2-dimensional kMeans

In the next experiment, we tackle the clustering task #2, using the clustering algorithm DBSCAN and a reduced data size compared to the experiment before. Similar as in the experiment of clus-

tering task #1, we show the execution times in Figure 3 and the maximal memory consumption in Figure 4.

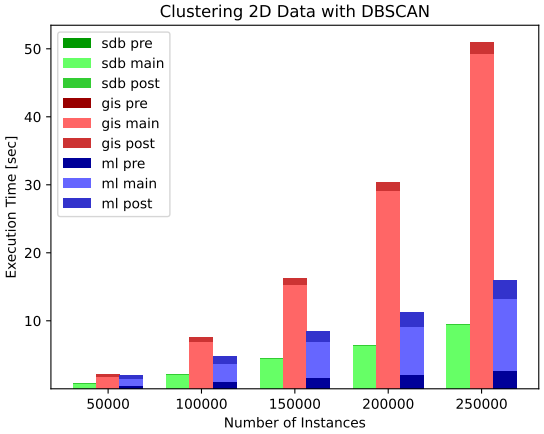

Figure 3: Execution Times - 2-dimensional DBSCAN

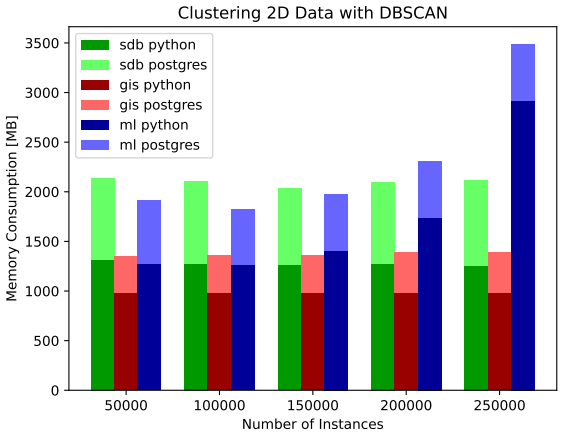

Figure 4: Maximal Memory Consumption - 2-dimensional DBSCAN

In the very same setting as the previous two experiments but with 4-dimensional data instead of two-dimensional, we conduct the following two experiments and evaluate the performance of the three clustering approaches. Like before, we present the execution times and the maximal memory consumption of kMeans in Figure 5 and Figure 6, and the execution times and maximal memory consumption when using DBSCAN in Figure 7 and Figure 8.

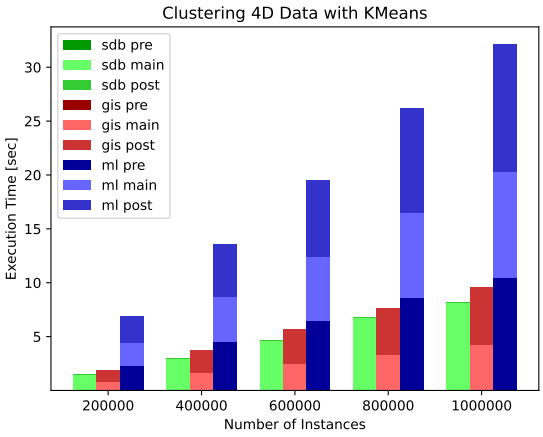

Figure 5: Execution Times - 4-dimensional kMeans

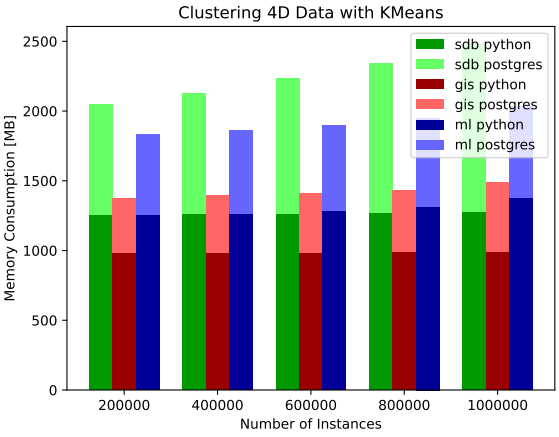

Figure 6: Maximal Memory Consumption - 4-dimensional kMeans

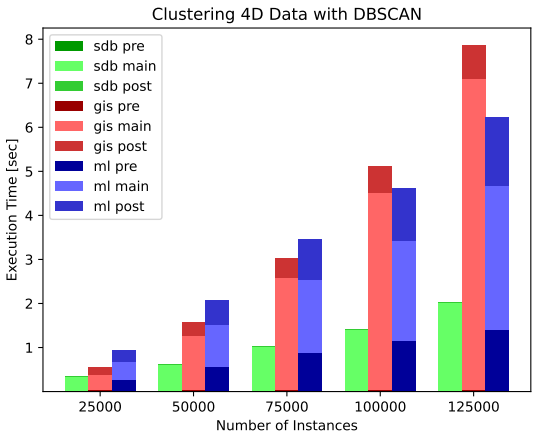

Figure 7: Execution Times - 4-dimensional DBSCAN

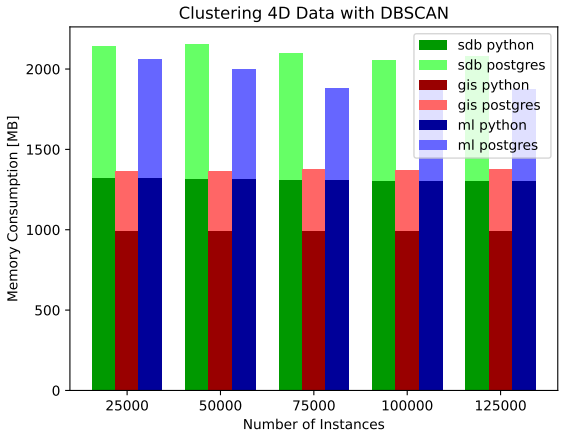

Figure 8: Maximal Memory Consumption - 4-dimensional DBSCAN

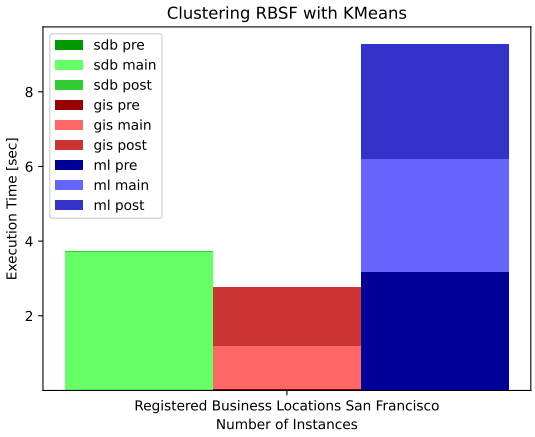

Figure 9: Execution Times - Registered Business kMeans

5.2. Experiments with Real Datasets

For the real datasets, as already mentioned in Section 4.1, we used 2 real world datasets. Again, we clustered with both algorithms, except for the Registered Business dataset, for which we only clustered with kMeans. This leads to the following tests:

1. Cluster Listed Buildings with 300368 points using kMeans
2. Cluster Listed Buildings with 300368 points using DBSCAN
3. Cluster Registered Business with 379345 points using kMeans

Similar to the tests conducted with the synthesized datasets, we conducted the tasks for all three approaches to test the performance and memory consumption. For the real datasets we limited the number of repetitions of the tests to 1, and therefore, no average was taken. Figures 9 and 11 show the performance of all approaches of kMeans based on execution time for both the Registered Business and Listed Buildings datasets, respectively. Figures 10 and 12 instead show the memory consumption of this task for both datasets.

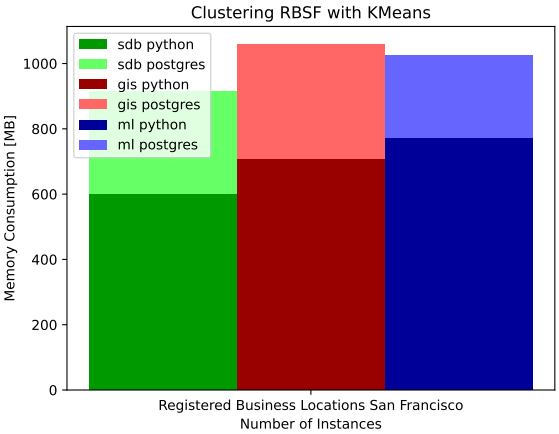

Figure 10: Maximal Memory Consumption - Registered Business kMeans

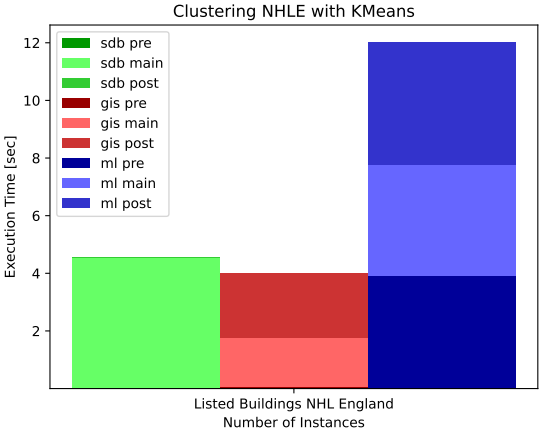

Figure 11: Execution Times - Listed Buildings kMeans

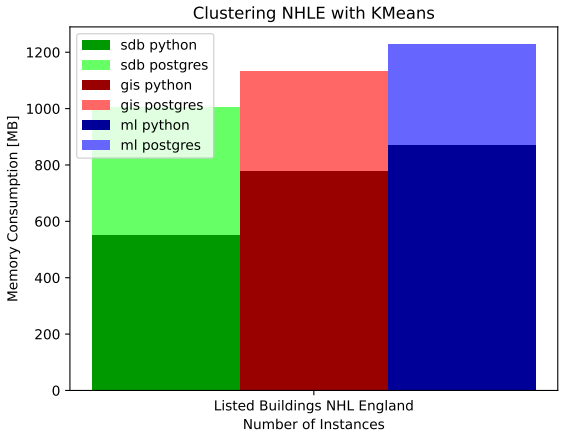

Figure 12: Maximal Memory Consumption - Listed Buildings kMeans

Additionally, Figures 13 and 14 show the performance based on execution time and memory consumption for DBSCAN on the Listed Buildings dataset.

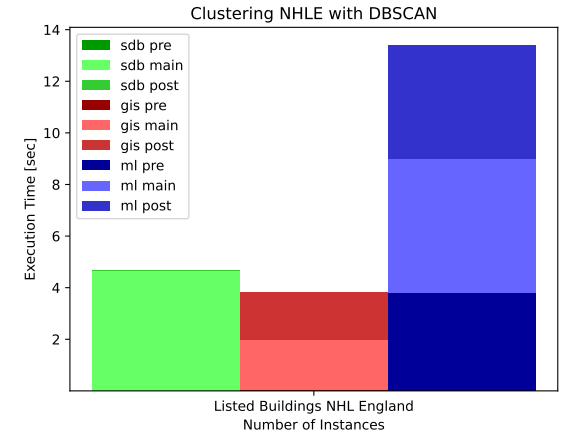

Figure 13: Execution Times - Listed Buildings DBSCAN

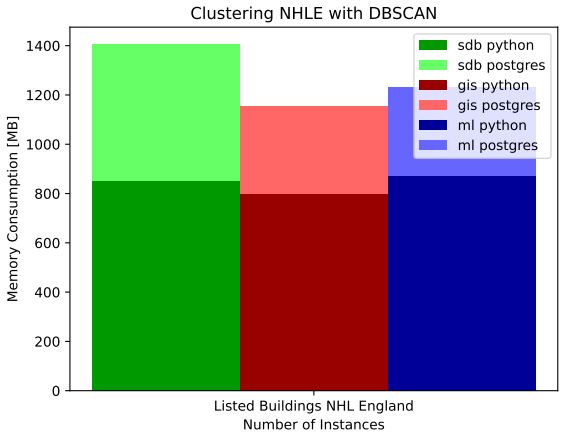

Figure 14: Maximal Memory Consumption - Listed Buildings DBSCAN

6. Discussion

As visible in Figures 1 and 5, we see that the integrated SDB clustering approach performs the best based on performance, followed by the external approach with QGIS and lastly the external ML approach for kMeans on the synthetic datasets. For DBSCAN, which can be observed in Figures 3 and 7, we see again, the integrated SDB clustering approach performing the best, followed by the external ML approach, which performs slightly worse than the external QGIS approach on the 4D dataset on smaller sizes, but overall comes out on top. Based on performance, we observe a linear relation between dataset sizes, *i.e.*, the decrease in performance is linear to the increase in dataset size for both kMeans and DBSCAN, except the external approach with QGIS, which has an exponential decrease in performance. Memory consumption on the other hand stays relatively the same, with the external approach with QGIS showing the best results, next the integrated SDB clustering approach, and the worst performing based on memory consumption again is the external ML approach, see Figures 2, 4, 6 and 8.

For the real datasets on the other hand, we can observe a similar behaviour for both datasets based on the performance of all three approaches if we cluster with kMeans in Figures 9 and 11, which makes sense due to their similar sizes. In comparison to the synthesized datasets, based on performance for kMeans, the external approach with QGIS shows the best results, second-best the integrated SDB clustering approach, and worst, the external ML approach. As for the memory consumption, here the integrated SDB clustering approach performs best, and we see slight differences between the other two approaches between both datasets, visible in Figures 10 and 12. Whereas in the Listed Buildings dataset, the external approach with QGIS is slightly better than external ML, in the Registered Business dataset we see the external ML approach coming out on top between both approaches. Unfortunately, these results have to be taken with a grain of salt, as we performed the tests only once, which makes the results, at least for the memory consumption, imprecise. For DBSCAN, we only conducted tests for the Listed Buildings dataset, see Figures 13 and 14. We found, similarly to the tests conducted with kMeans, that the external approach with QGIS shows the best results, followed by the integrated SDB clustering and external ML approaches. Based on memory consumption, we ob-

serve again, the external approach with QGIS performing the best, but interestingly, followed by the external ML approach, with the integrated SDB clustering approach performing the worst.

7. Conclusion and Outlook

In conclusion, our research showed that the integrated approach performs best when considering runtime performance. This indicates, that it is advantageous to use this approach if the dataset is larger and system resources are limited. In terms of memory consumption, all three exhibit comparable performances. Given that the external GIS approach introduces significant overhead and an exponential increase in execution time the larger the dataset, an important takeaway for users of GIS systems is to familiarize oneself with spatial database systems when facing clustering tasks for large spatial data. Furthermore, if one is already acquainted with Python programming and corresponding machine learning framework, it is wise to investigate if an approach using spatial database queries is a better fit to a given task. Someone already capable of programming such a Python approach will likely find it easy to implement an internal spatial database approach.

Lastly, there are several research directions to extend this work. One could add further datasets of variable sizes to more thoroughly investigate the impact of different approaches to real life scenarios. In addition, one could also extend the synthetic datasets to include more diverse point distributions. Another direction of future work would be to test different spatial database systems, GIS applications and machine learning frameworks. One could also test different system configurations such as having a dedicated server for the spatial database system, using different database indexing strategies or using different operating systems.

8. References

[1] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[2] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[3] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, 1996, pp. 226–231.

[4] *About PostGIS — PostGIS*, `https://postgis.net/`, Accessed: 2022-12-10.

[5] *PostgreSQL: The World's Most Advanced Open Source Relational Database*, `https://www.postgresql.org/`, Accessed: 2022-12-10.

[6] S. Ray, B. Simion, and A. D. Brown, "Jackpine: A benchmark to evaluate spatial database performance," in *2011 IEEE 27th International Conference on Data Engineering*, IEEE, 2011, pp. 1139–1150.

[7] *ST_ClusterDBSCAN*, `https://postgis.net/docs/ST_ClusterDBSCAN.html`, Accessed: 2022-12-10.

[8] *ST_ClusterKMeans*, `https://postgis.net/docs/ST_ClusterKMeans.html`, Accessed: 2022-12-10.

[9] *Welcome to the QGIS project!* `https://qgis.org/en/site/`, Accessed: 2022-12-10.

[10] *25.1.15.4 DBSCAN clustering - QGIS documentation*, `https://docs.qgis.org/3.22/en/docs/user_manual/processing_algs/qgis/vectoranalysis.html#dbscan-clustering`, Accessed: 2022-12-10.

[11] *25.1.15.4 k-means clustering - QGIS documentation*, `https://docs.qgis.org/3.22/en/docs/user_manual/processing_algs/qgis/vectoranalysis.html#k-means-clustering`, Accessed: 2022-12-10.

[12] *QGIS processing framework*, `https://docs.qgis.org/3.22/en/docs/user_manual/processing/index.html`, Accessed: 2022-12-10.

[13] *Psycopg - PostgreSQL database adapter for Python*, `https://www.psycopg.org/docs/`, Accessed: 2022-12-10.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] *Listing data from the national heritage list for england*, `https://historicengland.org.uk/listing/the-list/data-downloads/`, Accessed: 2023-01-18.

[16] *Registered business locations - san francisco*, `https://data.sfgov.org/Economy-and-Community/Registered-Business-Locations-San-Francisco/g8m3-pdis/data`, Accessed: 2022-12-13.

[17] *Sqlalchemy*, https://www.sqlalchemy.org, Accessed: 2023-02-05.

[18] *Geopandas*, https://geopandas.org, Accessed: 2023-02-05.

[19] *Qgis python api*, https://qgis.org/pyqgis/master/, Accessed: 2023-01-18.

[20] *Proc: Linux process information interface*, https://proc.readthedocs.io/en/latest/, Accessed: 2023-02-05.

A. SQL Statements

```sql
CREATE TABLE IF NOT EXISTS synth1(
    id SERIAL PRIMARY KEY NOT NULL,
    geom GEOMETRY(Point, 26918) NOT NULL,
    cid INT
);
```

Listing 1: Create table statement for the synth1 table. The table name can be switched for other dataset names, but the structure remains the same for all datasets.

```sql
SELECT id, ST_ClusterKMeans(geom, {k})
        over ()
    AS cid
FROM {table_name};
```

Listing 2: Select statement, which calls the integrated *kMeans* algorithm to cluster data. the variables in curly brackets will be filled at runtime using the python string formatter.

```sql
SELECT id, ST_ClusterDBSCAN(geom, eps := {eps}, minpoints := {minpoints})
        over ()
    AS cid
FROM {table_name}
```

Listing 3: Select statement, which calls the integrated *DBSCAN* algorithm to cluster data. the variables in curly brackets will be filled at runtime using the python string formatter.

```sql
SELECT json_agg(jsonb_build_object(
        'type',       'Feature',
        'geometry',   ST_AsGeoJSON(ST_MAKEPOINT(ST_X(geom), ST_Y(geom)))::jsonb,
        'properties', to_jsonb( t.* ) - 'id' - 'geom'))
FROM (
    SELECT geom FROM synth1 -- synth2
    WHERE random() <= 0.01
    ORDER BY random()
    LIMIT 500
) as t;
);
```

Listing 4: Statement to create a GeoJSON-String to visualize synthetic data. The random number in combination with the limit is used to sample a subset of the data.