SOHAC: Efficient Storage of Tick Data That Supports Search and Analysis

Gabor I. Nagy, Krisztian Buza

Budapest University of Technology and Economics
Magyar tudósok körútja 2, H-1117 Budapest, Hungary
gnagy@tmit.bme.hu,buza@cs.bme.hu
http://www.bme.hu

Abstract. Storage of tick data is a challenging problem because two criteria have to be fulfilled simultaneously: the storage structure should allow fast execution of queries and the data should not occupy too much space on the hard disk or in the main memory. In this paper, we present a clustering-based solution, and we introduce a new clustering algorithm that is designed to support the storage of tick data. We evaluate our algorithm both on publicly available real-world datasets, as well as real-world tick data from the financial domain provided by one of the world-wide most renowned investment bank. In our experiments we compare our approach, SOHAC, against a large collection of conventional hierarchical clustering algorithms from the literature. The experiments show that our algorithm substantially outperforms – both in terms of statistical significance and practical relevance – the examined clustering algorithms for the tick data storage problem.

1 Introduction

In order to describe objects or phenomena in real-world applications, usually, a relatively large set of attributes (or features) are necessary. The values of these attributes often change over time, e.g., prices on the stock market, temperature and humidity of the air, blood pressure or pulse of a person, etc. In most cases, the dynamics of these attributes, i.e., how they change their values, are almost as important as (or sometimes even more important than) the current values of the attributes. Therefore, we need to keep track of the changes of those values which results in very large collections of data.

In particular, the size of data describing financial transactions of stock markets may be several tera or even petabytes. Such data is often called tick data, see e.g. [14]. Tick data can be considered as a matrix that represents trades of financial assets. Columns of this matrix correspond different properties of transactions, such as price, volume of the trade, the symbol of an asset, etc. Every time a transaction is executed or a quote is given for a stock, a row is appended to this matrix. Therefore, this matrix grows rapidly while a technology is necessary that allows efficient storage and quick retrieval of the data. Solutions are often built over database technology such as a KDB database.[1] However, as we

[1] http://kx.com/

will describe in more detail, a straightforward application of such technology leads to suboptimal storage of stock market data.

The major reason for the aforementioned storage to be suboptimal is the redundancy of conventional techniques: in case of a straightforward solution, one would use orders of magnitudes more storage space (either disk space or main memory) than required, therefore, storage, access, search and analysis of the data becomes computationally more expensive than necessary. One way to alleviate this problem is the usage of regular data compression methods (see e.g. [16] for an excellent overview). This approach is well-suited for cheap storage of large, historical archives of the data. However, it does not support quick access to the data: if the data is compressed, in order to be able to execute an analytic or search query, a large archive (or at least some parts of that) must be decompressed which might be computationally expensive and therefore the procedure could become highly inefficient. This problem becomes crucial if *many* queries has to be executed which is usually the case in real-life applications, e.g. when trading on a stock market.

In this paper, we aim at finding a compromise between the both aforementioned cases, i.e., we aim at developing a storage structure for tick data that reduces the storage space required by the straightforward approach while it allows to execute search and analytic queries efficiently. In particular, our approach is based on the decomposition of a large tick data matrix into two (or three) much smaller matrices. We achieve this decomposition by the clustering of the columns of the matrix. Although, conventional clustering algorithms achieve significant improvements, motivated by hierarchical clustering algorithms, we develop a new clustering algorithm that minimizes storage space required for a tick data matrix. Therefore we call our approach SOHAC, Storage-Optimizing Hierarchical Agglomerative Clustering. We evaluate SOHAC both on publicly available real-world datasets, as well as real-world tick data from the financial domain provided by Morgan Stanley, one of the world-wide most renowned investment bank. In our experiments we compare our approach against a large collection of conventional hierarchical clustering algorithms from the literature. The experiments show that our algorithm significantly – both in terms of statistical significance and practical relevance – outperforms the examined clustering algorithms for the tick data storage problem.

This paper is organized as follows: Section 2 reviews related works. Our approach is described in Section 3, followed by our experiments in Section 4. Finally, we conclude in Section 5.

2 Related work

The availability of high-resolution data describing transactions on financial markets, especially *tick data* (also known as *tick-by-tick data*) allows thorough analysis of the markets and their dynamics. Some of the most relevant recent works focused for example on currency exchange rates [24], [15], [18], [17], stock market tick data [14], risk analysis [7] and and the dynamics of stock markets [3]. Based on tick data, Akram et al. empirically studied the law of one price on different financial markets [2], while Ahamad et al. focused on the summarization of tick data time series [1].

Although, storage of tick data is a core component of the systems performing the above analysis tasks, none of the above works focused on how to develop storage structures for tick data. As we will demonstrate it in Section 3.1, the storage of tick data is a non-trivial task: widely-used techniques result in redundant and therefore suboptimal solutions. Conventional compression techniques, such as run-length encoding, may result in massive reduction of the required storage space. We refer to [16] for an excellent overview of conventional compression techniques. As mentioned in Section 1, such techniques are well-suited for cheap storage of large, historical archives of the data. However, they do not support quick access to the data: if the data is compressed, in order to access the data and execute an analytic or search query, a large archive (or at least some parts of that) must be decompressed which might be computationally expensive and therefore the procedure could become highly inefficient. Therefore, in contrast to the previously discussed techniques, we build our approach on clustering which is known to have a high potential to reduce both the volume of data and its redundancy.

In the last decades, very large number of clustering algorithms were developed for various tasks (see e.g. [5], [9], [11], [12], [13] and [21]). We refer to [19] and [23] for excellent surveys of clustering algorithms. Although one can achieve substantial improvements if one uses general-purpose clustering algorithms in our approach, such conventional clustering algorithms were originally not designed for storage optimization of tick data. In contrast, the clustering algorithms we propose in Section 3 directly minimize the storage space required for tick data.

In the context of data compression, Han and Yand [10] used clustering as preprocessing for conventional data compression techniques. As they perform the actual compression by a conventional compressor, 7-zip[2], this approach does not support fast enough execution of search and analytic queries. Instead of using clustering as a preprocessing step for standard compressors, we build our approach on the cluster-based decomposition of tick data matrices.

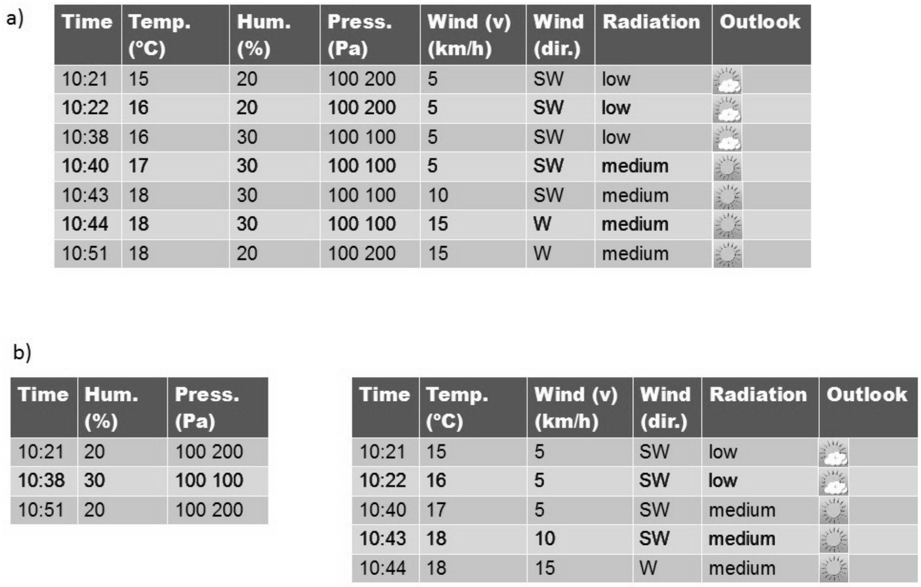
3 Decomposition of Tick Data Matrix based on Clustering

In this section, we describe our approach in more detail. First, we motivate our approach with an illustrative example, then we develop a new clustering algorithm that supports efficient storage of tick data.

3.1 An Illustrative Example

Suppose that a weather station monitors features of weather conditions. In this example, such features are the temperature, humidity and pressure of the air, the

[2] http://www.7-zip.org



a)

Time	Temp. (°C)	Hum. (%)	Press. (Pa)	Wind (v) (km/h)	Wind (dir.)	Radiation	Outlook
10:21	15	20	100 200	5	SW	low	⛅
10:22	16	20	100 200	5	SW	low	⛅
10:38	16	30	100 100	5	SW	low	⛅
10:40	17	30	100 100	5	SW	medium	☀
10:43	18	30	100 100	10	SW	medium	☀
10:44	18	30	100 100	15	W	medium	☀
10:51	18	20	100 200	15	W	medium	☀

b)

Time	Hum. (%)	Press. (Pa)
10:21	20	100 200
10:38	30	100 100
10:51	20	100 200

Time	Temp. (°C)	Wind (v) (km/h)	Wind (dir.)	Radiation	Outlook
10:21	15	5	SW	low	⛅
10:22	16	5	SW	low	⛅
10:40	17	5	SW	medium	☀
10:43	18	10	SW	medium	☀
10:44	18	15	W	medium	☀

Fig. 1. An illustrative example for tick data. Features describing the weather are monitored continuously. Whenever the value of one of the features changes, a new row is inserted into the recordings (see the table in the top). Decomposition of such tables by features (columns) that change their values simultaneously may substantially reduce the required storage space (see the tables in the bottom of the figure).

velocity and direction of the wind, the intensity of the radiation of the sun and the overall outlook (such as sunny, cloudy, raining or snowing). These features are monitored continuously over the time. Whenever the value of one of these features changes, new raw is inserted into the recordings. This new row contains the values of the features as well as time-stamp indicating *when* the observations were made. See the matrix in the top of Figure 1.

This representation, called tick data, is well-suited for queries: for example, if we are interested for the features of the weather at 10:30 o'clock, we only need to find the raw corresponding the most recent observation *before* 10:30, i.e., we have to consider the raw at 10:22. This raw describes the "state of the world", i.e., it contains the values of all the features that are relevant in the current application. Such queries regarding the "state of the world" at a given time can be effectively supported by indexing techniques.

The only disadvantage of the representation shown in the top of Figure 1 is that the total size of the matrix may become much larger than actually required. In order to illustrate this we stored the same information in two smaller matrices in the bottom of Figure 1. In our approach such decompositions are based on the clustering of columns: in the example, we consider two clusters of columns. One of the clusters contains *Humidity* and *Pressure*, while the other cluster contains the other columns, i.e., *Temperature, Velocity of the wind, Direction of the wind, Radiation* and *Outlook*. As shown in the example, due to the decomposition, we can save storage space: the total number of cells required to store the data was reduced from $7 \times 7 = 49$ to $3 \times 2 + 5 \times 5 = 31$ (without counting the cells in the column *Time* which acts like an index column). This corresponds to a *compression ratio* of $31/49 \approx 0,633$.

While the decomposition reduces the required storage space, in the worst case, the computational complexity of a query may increase moderately: if we are interested for *all* the features describing the weather at 10:30, we have to execute two queries instead of one, however, both queries are executed on much *smaller datasets* (and therefore the overall execution time is expected to grow only moderately compared to the previous case). Whereas if we are only interested for the *temperature and radiation* we have to execute just one query on a dataset of reduced size (and therefore the overall execution time is expected to be reduced too).

The example in Figure 1 illustrates the decomposition of a tick data matrix in an intuitive way. Next, we systematically study such decompositions and develop an algorithm that aims at minimizing the storage space required after the decomposition.

3.2 Definitions and Problem Formulation

In general, a *tick data matrix* M is a matrix where columns correspond attributes or features while rows correspond observations of the same features at different moments of time. Rows of the matrix are ordered according to the order of observations, i.e., the values of the i-th row were observed *before* the values of the j-th row if and only if $i < j$. While the observations are made, a new row is added whenever the value of an attribute changes. However, as long as none of the attribute-values changes no new row is added to the matrix, therefore two rows of a tick data matrix differ in the value of at least one attribute. There is an additional column that is used to index the rows of a tick data matrix. This additional *index column* may contain, for example, ascending integer numbers (like the number of the corresponding row) or a time-stamp (see the *Time* column in the example in Section 3.1). We use the term *regular column* for all the columns other than the index column.

With *decomposition* of a tick data matrix M we mean the partitioning of the regular columns of M into k disjoint partitions P_i, $1 \leq i \leq k$, i.e., for each regular column c_j of M:

$$c_j \in P_1 \vee c_j \in P_2 \vee ... \vee c_j \in P_k$$

and for all i, j with $i \neq j$

$$P_i \cap P_j = \emptyset.$$

Note that this partitioning refers to the regular columns only, i.e., in this formulation, the index column does not belong to any partition. Then, for each partition P_i, a matrix M_i is derived from M by selecting the index column and those columns of M that belong to partition P_i. Subsequent rows of a derived matrix M_i may contain the same values in all the regular columns. In such cases we only keep the first row. For example, in Figure 1, $P_1 = \{$Humidity, Pressure$\}$, $P_2 = \{$ Temperature, Wind (velocity), Wind (direction), Radiation, Outlook $\}$ and the corresponding matrices M_1 and M_2 are shown in the bottom left and bottom right of the Figure.

We can easily see that the original matrix can be reconstructed from the decomposition described above, and therefore, instead of the original matrix M, one can use this decomposition to calculate the results of search and analytic queries.

In this paper, we target the problem of finding a decomposition so that the required storage space is minimized. In particular, for a given number of partitions k, we aim at finding a decomposition so that the total number of all the cells in all the matrices M_i (without counting the cells in the index column) is minimized. Our approach can simply be adapted for the case of more advanced storage models, where we do not assume uniform storage cost for each cells and/or the storage costs of the index cell is also taken into account. We plan to access this issue in our future work.

We note that k is usually relatively small: for example, for the storage of tick data of financial transactions, the user is most interested for the decomposition into $k = 2$ or $k = 3$ partitions.

3.3 Clustering of Columns of Tick Data Matrix

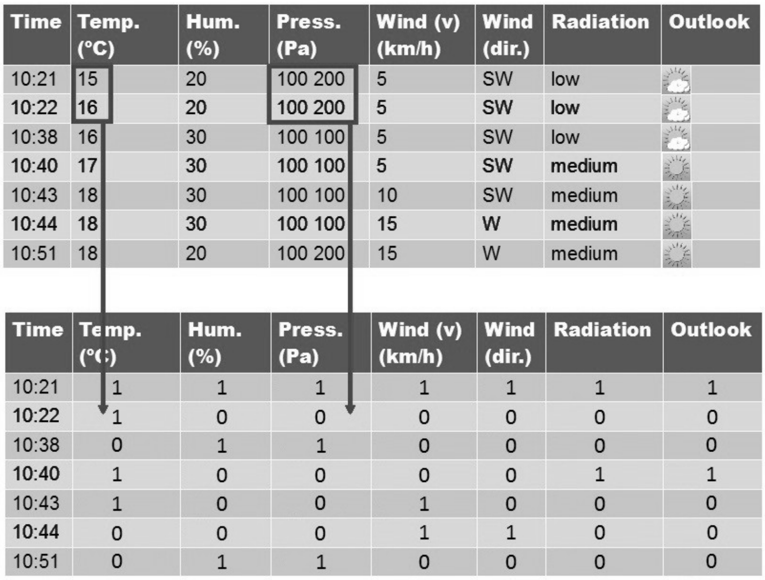
In the literature, there are many clustering algorithms that are able to produce non-overlapping partitions in a way that these partitions together cover all the instances. Therefore, one solution for the problem defined in the previous section is to cluster the columns of a tick data matrix using one of the conventional clustering algorithms.

In the context of our problem, two regular columns are considered to be similar, if they often change values in the same row. In order to be able to reuse proximity measures from the literature, we define a *binary change indicator matrix* I over a tick data matrix M. Except the entries of the index column, all the entries of the binary change indicator matrix I are either 0 or 1 depending on whether or not the value of a cell in the tick data matrix M is equal to the value of the cell in the same column and the *previous* row of M:

$$I(i,j) = \begin{cases} M(i,j) \text{ if the } j\text{-th column is the index column in } M \\ \quad\; 0 \;\text{ if } i > 1 \text{ and } M(i,j) = M(i-1,j) \\ \quad\; 1 \;\text{ otherwise} \end{cases}$$

where $M(i,j)$ and $I(i,j)$ denote the entries in the i-th row and j-th column of the tick data matrix M and binary change indicator matrix I respectively.

As an example, Figure 2 shows how the binary change indicator matrix is derived from a tick data matrix. The index column is the *Time* column in this example.



Time	Temp. (°C)	Hum. (%)	Press. (Pa)	Wind (v) (km/h)	Wind (dir.)	Radiation	Outlook
10:21	15	20	100 200	5	SW	low	⛅
10:22	16	20	100 200	5	SW	low	⛅
10:38	16	30	100 100	5	SW	low	⛅
10:40	17	30	100 100	5	SW	medium	☀
10:43	18	30	100 100	10	SW	medium	☀
10:44	18	30	100 100	15	W	medium	☀
10:51	18	20	100 200	15	W	medium	☀

Time	Temp. (°C)	Hum. (%)	Press. (Pa)	Wind (v) (km/h)	Wind (dir.)	Radiation	Outlook
10:21	1	1	1	1	1	1	1
10:22	1	0	0	0	0	0	0
10:38	0	1	1	0	0	0	0
10:40	1	0	0	0	0	1	1
10:43	1	0	0	1	0	0	0
10:44	0	0	0	1	1	0	0
10:51	0	1	1	0	0	0	0

Fig. 2. Construction of a binary change indicator matrix from a tick data matrix. The tick data matrix is shown in the top of the figure, while the corresponding indicator matrix is shown in the bottom. The index column is the *Time* column in this example.

After constructing the binary change indicator matrix I, we can use its regular columns (i.e., all the columns except the index column) as instances in conventional clustering algorithms. Despite the fact that conventional clustering algorithms are not designed to produce optimal partitions in terms of our problem from Section 3.2, as we will show in the experiments, if we use the partitioning of the columns produced by conventional clustering algorithms we can achieve substantial improvements w.r.t. the required storage space compared to the case of storing the original tick data matrix. In the next section, we develop a clustering algorithm that directly optimize the storage space required to store the decomposed tick data matrix.

3.4 SOHAC: Storage-Optimizing Hierarchical Agglomerative Clustering

In this section we propose our new clustering algorithm, SOHAC, Storage-Optimizing Hierarchical Agglomerative Clustering that is designed for clustering columns of a tick data matrix. The algorithm builds on the hierarchical agglomerative strategy. Therefore, initially, all the objects belong to separate clusters. Then, clusters are iteratively merged together as long as the current number of clusters is more than k, the user-defined number of partitions. Therefore, at the end of this iterative process, k clusters are produced.

Algorithm 1 SOHAC: Storage-Optimizing Hierarchical Agglomerative Clustering for Tick Data

```
Require: Tick data matrix $M$, number of partitions $k$
Ensure: Partitioning of the columns of $M$
Construct the binary indicator matrix $I$ from $M$
$P = \Big\{\{c_1\}, \{c_2\}, ..., \{c_n\}\Big\}$    (Initially, each column $c_j$ of $M$ is a separate cluster)
while $|P| > k$ do
    $s \leftarrow \infty$    (Storage size for the best partitioning found so far)
    for all pairs of clusters $(\mathcal{C}_i, \mathcal{C}_j)$, with $\mathcal{C}_i \in P$, $\mathcal{C}_j \in P$ do
        $\mathcal{C}_i' \leftarrow \mathcal{C}_i \cup \mathcal{C}_j$    (Merge clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ into the new cluster $\mathcal{C}_i'$ )
        $P' \leftarrow P \setminus \{\mathcal{C}_i\} \setminus \{\mathcal{C}_j\} \cup \{\mathcal{C}_i'\}$
        $s'$ = storage size required to store the decomposition corresponding to $P'$
            (This can simply be computed based on $I$.)
        if $s' < s$ then
            $P^* \leftarrow P'$
            $s \leftarrow s'$
        end if
        $P \leftarrow P^*$
    end for
end while
return $P$
```

The key feature of our algorithm is that in each iteration it merges those two clusters that lead to minimal storage size of the decomposed matrix. This storage size can simply be calculated based on the binary change indicator matrix. For each examined partitioning of the columns, we decompose the binary change indicator matrix. Then, we consider the rows that contain only zeros in the regular columns. The cells of such rows can be eliminated in the examined decomposition without loss of information. Therefore, in order to determine the number of cells required for the storage of the examined decomposition, we only need to count the cells in the rows that contain only zeros in their regular columns. The pseudocode of our algorithm is shown in Algorithm 1.

4 Experiments

In this section, we describe the experiments we performed in order to evaluate our approach used and discuss the results.

4.1 Experimental Settings

Datasets — We tested our approach both on a real-world tick data describing financial transactions and several publicly available real-world dataset.

The real-world tick data from the financial domain was provided us by Morgan Stanley, one of the most renowned investment bank of the world. Therefore, in this paper, we call this dataset *MorganStanleyTickData*. MorganStanleyTickData contains 30 regular columns and 4.080.431 rows.

Additionally, we used publicly-available real-world datasets: we used some of the most popular datasets from the UCI machine learning repository [8]. In particular, these datasets were: Adult, Breast Cancer Wisconsin (Diagnostic), Car Evaluation, Forest Fires [6] and Poker Hand. As the datsets in the UCI repository do not contain tick data, in order to be able to perform reasonable experiments, as preprocessing, we removed the id values from the UCI datasets (if present) and sorted the records of the UCI datasets in lexicographical order. After sorting, the values of cells in the same columns and subsequent rows were often equal, this is the key property of tick data that our approach exploits.

Experimental Protocol — In our experiments we compared the decomposition of a tick data matrix resulting from the clusters produced by our approach, SOHAC, with the decomposition of the same tick data matrix using other clustering algorithms from the literature. We measured the quality of a decomposition by the compression ratio (CR), i.e., the ratio of the number of cells in regular columns after the decomposition and the number of cells in regular columns in the original matrix:

$$CR = \frac{\text{number of cells in regular columns after decomposition}}{\text{number of cells in regular columns in the original matrix}}$$

An example for the calculation of compression ratio can be found in Section 3.1.

We used a procedure that is similar to 10-fold-crossvalidation. In particular, we split the entire tick data matrix into 10 disjoint sub-matrices, and we repeated all the experiments 10 times: in each of the 10 rounds of the process, we used a different sub-matrix, and clustered the columns of that sub-matrix. Therefore, we could calculate the average and standard deviation of the compression ratio.

Baselines — As our approach, SOHAC, is built on hierarchical agglomerative clustering, in our experiments we focused on comparing the partitioning produced by SOHAC against the partitioning produced by different variants of hierarchical agglomerative clustering algorithms. Additionally, we compared the partitioning produced by SOHAC against the partitioning produced by k-Means [22].

Regarding the variants of hierarchical agglomerative clustering algorithms, we used Single Linkage, Complete Linkage and Average Linkage with the following proximity measures: Euclidean Distance, Cosine Similarity, Dice Similarity, Jaccard Similarity, Kulczynski Similarity, Nominal Similarity, Rogers-Taminoto Similarity, Russell-Rao Similarity Simple Matching Similarity, Chebychev Distance, Manhattan Distance and Overlap Similarity. Implementations of these proximity measures are available in the RapidMiner software tool[3]. In our experiments, we used this software to calculate the partitioning with the baseline

[3] http://www.rapidminer.com/

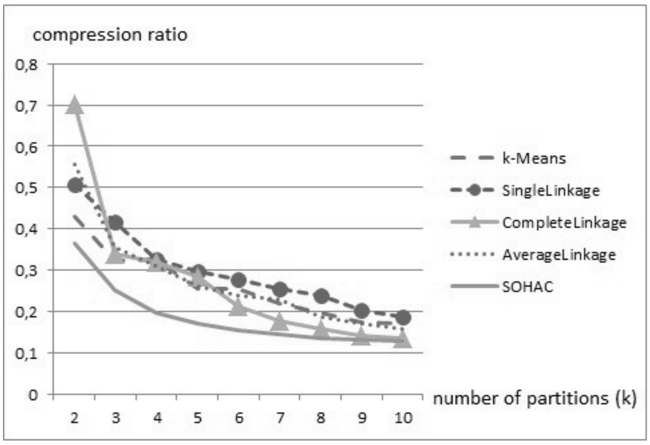


Fig. 3. Performance of our approach, SOHAC, and Complete Linkage with Euclidean distance, Single Linkage and Average Linkage with Cosine Similarity for the case of varying the number of partitions, k, between 2 and 10. The number of partitions are shown on the horizontal axis. The performance is measured in compression ratio (vertical axis) and is averaged for 10 splits.

algorithms, more details about these baseline algorithms can be found in the documentation of RapidMiner and the reference therein.

In total, taking all the examined variants of the baselines into account, we compared our approach against 38 clustering algorithms from the literature.

4.2 Results

In our first experiment, we tested our approach on MorganStanleyTickData. We tried three different values for the number of partitions, k. Table 1 shows the results for k equal to 2, 3 and 4 respectively. Figure 3 shows the results of our approach and some of the baseline algorithms, namely k-Means and Complete Linkage with Euclidean distance, Single Linkage and Average Linkage with Cosine Similarity for the case of varying k between 2 and 10.

As we can see from Table 1, our algorithm substantially outperformed its 38 competitors. In many cases, the difference was significant in terms of average and standard deviation.

We performed our second experiment on datasets from the UCI machine learning repository. For simplicity, in Table 2, we only show the results for our approach and three baselines, Single Linkage, Complete Linkage and Average Linkage with Euclidean distance. We considered these algorithms as representatives of all the examined 38 baselines. The other examined algorithms performed similar to the ones shown in Table 2. As one can see, our approach outperformed the baselines again.

Just like in the first experiment, we additionally tested other values for the number of partitions, k, in the second experiment too. The results were similar

Table 1. Performance of our approach, SOHAC, and the baselines on *MorganStanleyTickData*. The performance is measured by compression ratio, *smaller* values indicate *better* performance. Values are averaged for 10 splits, standard deviation is shown after the ± symbol. For each value of k, the number of partitions, bold font denotes the winner.

Algorithm	Distance Measure	$k = 2$	$k = 3$	$k = 4$
Average- Linkage	Dice	0.9799±0.0016	0.5805±0.0596	0.3094±0.1311
	Jaccard	0.9799±0.0016	0.5805±0.0596	0.3094±0.1311
	Kulczynski	0.9799±0.0016	0.5805±0.0596	0.3094±0.1311
	Nominal	0.7385±0.1072	0.6309±0.0731	0.5612±0.0753
	Rogers-Tanimoto	0.7320±0.1032	0.6339±0.0696	0.5312±0.1184
	RussellRao	0.9799±0.0016	0.5805±0.0596	0.3094±0.1311
	SimpleMatching	0.7320±0.1032	0.6339±0.0696	0.5312±0.1184
	Chebychev	0.9799±0.0016	0.7169±0.0412	0.6836±0.0413
	Cosine	0.5556±0.2659	0.3560±0.0852	0.3084±0.0601
	Euclidean	0.7320±0.1032	0.6339±0.0696	0.5312±0.1184
	Manhattan	0.7320±0.1032	0.6339±0.0696	0.5312±0.1184
	Overlap	0.9605±0.0172	0.8788±0.0129	0.7734±0.0222
Complete- Linkage	Dice	0.7415±0.1020	0.5805±0.0596	0.3094±0.1311
	Jaccard	0.7415±0.1020	0.5805±0.0596	0.3094±0.1311
	Kulczynski	0.7415±0.1020	0.5805±0.0596	0.3094±0.1311
	Nominal	0.7044±0.0462	0.3460±0.1328	0.3254±0.1361
	RogersTanimoto	0.7013±0.0446	0.3388±0.1273	0.3190±0.1299
	RussellRao	0.9799±0.0016	0.5805±0.0596	0.3094±0.1311
	SimpleMatching	0.7013±0.0446	0.3388±0.1273	0.3190±0.1299
	Chebychev	0.9799±0.0016	0.7169±0.0412	0.6836±0.0413
	Cosine	0.8303±0.0762	0.7306±0.1298	0.3075±0.0875
	Euclidean	0.7013±0.0446	0.3388±0.1273	0.3190±0.1299
	Manhattan	0.7013±0.0446	0.3388±0.1273	0.3190±0.1299
	Overlap	0.8696±0.0475	0.7620±0.0408	0.6970±0.0441
Single- Linkage	Dice	0.9799±0.0016	0.7301±0.1952	0.3338±0.1629
	Jaccard	0.9799±0.0016	0.7301±0.1952	0.3338±0.1629
	Kulczynski	0.9799±0.0016	0.7301±0.1952	0.3338±0.1629
	Nominal	0.7607±0.1379	0.7296±0.1511	0.5612±0.0753
	RogersTanimoto	0.7520±0.1329	0.7228±0.1441	0.5587±0.0714
	RussellRao	0.9799±0.0016	0.9055±0.0264	0.4820±0.3088
	SimpleMatching	0.7520±0.1329	0.7228±0.1441	0.5587±0.0714
	Chebychev	0.9799±0.0016	0.7169±0.0412	0.6836±0.0413
	Cosine	0.5072±0.2641	0.4150±0.1893	0.3254±0.0683
	Euclidean	0.7520±0.1329	0.7228±0.1441	0.5587±0.0714
	Manhattan	0.7520±0.1329	0.7228±0.1441	0.5587±0.0714
	Overlap	0.9799±0.0016	0.9466±0.0016	0.9134±0.0018
k-Means	Euclidean	0.4291±0.1821	0.3242±0.1216	0.3244±0.1309
	Manhattan	0.8084±0.1219	0.6029±0.1224	0.4437±0.1274
SOHAC		**0.3649±0.0772**	**0.2526±0.0587**	**0.1960±0.0499**

Table 2. Performance of our approach, SOHAC, and Single Linkage, Average Linkage and Complete Linkage (with Euclidean Distance) on datasets from the UCI repository of machine learning datasets. The performance is measured by compression ratio, *smaller* values indicate *better* performance. Values are averaged for 10 splits, standard deviation is shown after the ± symbol. For each dataset, bold font denotes the winner.

Dataset	SOHAC	Single Linkage	Avg. Linkage	Complete Linkage
k = 2				
Adult	**0.8051±0.0256**	0.8672±0.0473	0.8558±0.0408	0.8558±0.0408
Breast C.W.	**0.5040±0.2420**	0.5708±0.2243	0.5478±0.2181	0.5142±0.2243
Car	**0.5199±0.0291**	0.6347±0.0806	0.6108±0.0733	0.5909±0.0660
ForestFires	**0.7816±0.0208**	0.7887±0.0286	0.7834±0.0288	0.7925±0.0389
Poker Hand	**0.5490±0.0001**	0.7582±0.0572	0.7582±0.0572	0.7871±0.0018
k = 3				
Adult	**0.7101±0.0251**	0.8018±0.0515	0.7884±0.0397	0.7876±0.0388
Breast C.W.	**0.4451±0.2424**	0.5022±0.2167	0.4915±0.2189	0.4628±0.2292
Car	**0.3869±0.0190**	0.4389±0.0235	0.4391±0.0238	0.4391±0.0238
ForestFires	**0.7242±0.0202**	0.7406±0.0212	0.7402±0.0213	0.7387±0.0178
Poker Hand	**0.4477±0.0003**	0.5978±0.0011	0.5978±0.0011	0.5978±0.0011
k = 4				
Adult	**0.6491±0.0222**	0.7402±0.0125	0.7437±0.0215	0.7501±0.0272
Breast C.W.	**0.4068±0.2344**	0.4414±0.2199	0.4394±0.2215	0.4289±0.2183
Car	**0.3141±0.0206**	0.3146±0.0198	0.3146±0.0198	0.3146±0.0198
ForestFires	**0.6857±0.0191**	0.7144±0.0214	0.7113±0.0226	0.7105±0.0177
Poker Hand	**0.4016±0.0004**	0.4272±0.0005	0.4272±0.0005	0.4272±0.0005

to the ones reported in Table 2, i.e., our approach outperformed the baselines for other k values too.

The reason for the good performance of our approach is that it directly optimizes compression ratio by searching for the partitioning that corresponds to minimal storage size, while other, general-purpose clustering algorithms optimize other criteria, e.g., k-Means aims at minimizing the sum of squared distances from the centroids [19].

Additionally, we note that our partners at Morgan Stanley were extraordinarily satisfied with the results of our approach.

5 Conclusion

In this paper we focused on the storage of tick data. Our approach aimed at reducing the disk/memory occupied by the data while it allowed quick access to the data.

In particular, we developed a new clustering algorithm, SOHAC, Storage-Optimizing Hierarchical Agglomerative Clustering that is designed for partitioning the columns of a tick data matrix. This partitioning allows efficient storage of the data by the decomposition of tick data matrices. In our experiments, we compared our approach, SOHAC, against a large number of other clustering algorithms both on real-world tick data provided by Morgan Stanley and on publicly available real-world datasets from the UCI repository. The results showed that our approach, SOHAC, substantially outperforms (in term of statistical significance and practical relevance, respectively) the examined other clustering algorithms. Furthermore, our partners at Morgan Stanley were extraordinarily satisfied with the results.

Future works may include various topics. For example, in this paper, we used a simplified model to calculate the disk/memory space required to store a tick data matrix, as we assumed uniform costs for the storage of each cell of a regular column. Additionally, one could consider other algorithms (local search, genetic algorithms, gradient descent, etc.) for finding the optimal partitioning. As a by-product of our experiments, we observed that our algorithm produced very similar clusterings on different splits of the data. This could motivate to speed-up the algorithm by sampling and the study of its stability, which could be interesting in the light of recent results concerning the theory of clustering [4]. Furthermore, one could examine whether some of the columns of the tick data matrix act as hubs and explore hub-based algorithms, such as *k*-Hubs [21], for the tick data storage problem. Moreover, factorization techniques, see e.g. [20], might also serve as the basis for column clustering algorithms. Last but not least, we mention that our algorithm can be applied in other domains, such as storage of multivariate time series or sensor data.

Acknowledgment. Discussions with Dr. Ferenc Bodon and Zoltan Papp, Morgan Stanley Analytics, Budapest, Hungary are greatly appreciated. The work reported in the paper has been developed in the framework of the project ”Talent care and cultivation in the scientific workshops of BME” project. This project is supported by the grant TÁMOP - 4.2.2.B-10/1–2010-0009

References

1. Ahmad, S., Taskaya-Temizel, T., Ahmad, K.: Summarizing time series: Learning patterns in volatileseries. Intelligent Data Engineering and Automated Learning–IDEAL 2004 pp. 523–532 (2004)
2. Akram, Q., Rime, D., Sarno, L.: Does the law of one price hold in international financial markets? evidence from tick data. Journal of Banking & Finance 33(10), 1741–1754 (2009)
3. Bartiromo, R.: Dynamics of stock prices. Physical Review E 69(6), 067108 (2004)
4. Ben-David, S., Von Luxburg, U., Pál, D.: A sober look at clustering stability. Learning Theory pp. 5–19 (2006)
5. Buza, K., Buza, A., Kis, P.: A distributed genetic algorithm for graph-based clustering. Man-Machine Interactions 2 pp. 323–331 (2011)
6. Cortez, P., Morais, A.: A Data Mining Approach to Predict Forest Fires using Meteorological Data. In: New Trends in Artificial Intelligence, Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence. pp. 512–523 (2007)

7. Dionne, G., Duchesne, P., Pacurar, M.: Intraday value at risk (ivar) using tick-by-tick data with application to the toronto stock exchange. Journal of Empirical Finance 16(5), 777–792 (2009)
8. Frank, A., Asuncion, A.: Uci machine learning repository (2010), http://archive.ics.uci.edu/ml
9. Guha, S., Rastogi, R., Shim, K.: Rock: A robust clustering algorithm for categorical attributes. Information Systems 25(5), 345–366 (2000)
10. Han, B., Yang, Z.: Data matrix compression by using co-clustering. In: Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on. vol. 4, pp. 2600 –2604 (july 2011)
11. Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., Wu, A.: An efficient k-means clustering algorithm: Analysis and implementation. Pattern Analysis and Machine Intelligence, IEEE Transactions on 24(7), 881–892 (2002)
12. Kurucz, M., Benczur, A., Csalogány, K., Lukács, L.: Spectral clustering in telephone call graphs. In: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis. pp. 82–91. ACM (2007)
13. Nanopoulos, A., Gabriel, H., Spiliopoulou, M.: Spectral clustering in social-tagging systems. Web Information Systems Engineering-WISE 2009 pp. 87–100 (2009)
14. Oh, K., Kim, K.: Analyzing stock market tick data using piecewise nonlinear model. Expert Systems with Applications 22(3), 249–255 (2002)
15. Ohnishi, T., Mizuno, T., Aihara, K., Takayasu, M., Takayasu, H.: Statistical properties of the moving average price in dollar–yen exchange rates. Physica A: Statistical Mechanics and its Applications 344(1), 207–210 (2004)
16. Salomon, D.: Data compression: the complete reference. Springer-Verlag New York Inc (2004)
17. Sazuka, N.: Analysis of binarized high frequency financial data. The European Physical Journal B-Condensed Matter and Complex Systems 50(1), 129–131 (2006)
18. Takayasu, M., Takayasu, H., Okazaki, M.: Transaction interval analysis of high resolution foreign exchange data. Empirical Science of Financial Fluctuations-The Advent of Econophysics 18, 25 (2002)
19. Tan, P., Steinbach, M., Kumar, V., et al.: Introduction to data mining. Pearson Addison Wesley Boston (2006)
20. Thai-Nghe, N., Drumond, L., Horváth, T., Schmidt-Thieme, L.: Multi-relational factorization models for predicting student performance. In: KDD 2011 Workshop on Knowledge Discovery in Educational Data (KDDinED 2011) (2011)
21. Tomasev, N., M. Radovanovic, D. Mladenic, M.I.: The Role of Hubness in Clustering High-Dimensional Data. In: 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), LNCS/LNAI, vol. 6634, pp. 183–195. Springer (2011)
22. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann (2011)
23. Xu, R., Wunsch, D., et al.: Survey of clustering algorithms. Neural Networks, IEEE Transactions on 16(3), 645–678 (2005)
24. Zhou, B.: High-frequency data and volatility in foreign-exchange rates. Journal of Business & Economic Statistics 14(1), 45–52 (1996)