Three Open Questions related to the Tick Data Decomposition Problem

Krisztian Buza, Gábor I. Nagy*

Budapest University of Technology and Economics, Hungary
buza@cs.bme.hu, nagy.gabor.i@gmail.com

Alexandros Nanopoulos

University of Eichstätt-Ingolstadt, Germany
alexandros.nanopoulos@ku.de

Keywords: combinatorial optimization, tick data, data storage

The tick data decomposition problem is a combinatorial optimization problem motivated by real-world applications, in particular, by the need for efficient storage structures for discrete-valued multivariate time-series, such as the data describing financial transactions, or bag of words vectors of dynamically changing texts such as blogs or Wikipedia pages. Here, we will describe the tick data decomposition problem and we will point out three open questions related to the tick data decomposition problem.

In many applications, various attributes of an object are measured continuously over time. A *tick data matrix* M is a matrix where columns correspond attributes or features while rows correspond observations of the same features at different time points. 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 one or more attribute(s) change(s). 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 shown in Figure 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 cluster. Then, for each cluster P_i, a matrix M_i is derived from M by selecting the index column and those columns of M that belong to cluster 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, $P_1 = \{\text{Humidity, Pressure}\}$, $P_2 = \{\text{Temperature, Wind (velocity), Wind (direction), Radiation, Outlook}\}$ and the corresponding matrices M_1 and M_2 are shown in the bottom left and bottom right of the Figure 1.

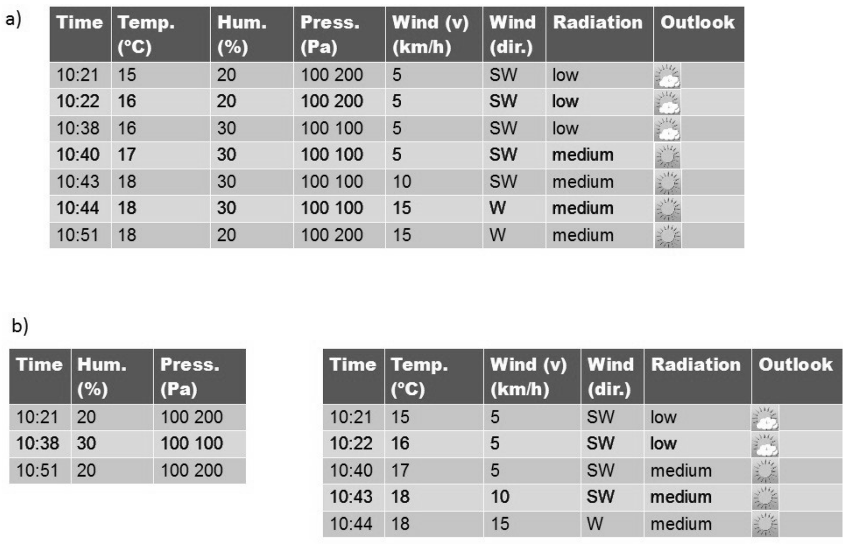
We can easily see that the original matrix can be reconstructed from the decomposition described above, and therefore, instead of the original matrix M, one can use this decomposition to calculate the results of search and analytic queries. Furthermore, as we have shown in our previous works, this decomposition allows to process queries *efficiently*, i.e., without the explicit need for decompressing the data, and simultaneously it leads to substantial improvements in terms of storage space [1, 2].

Consequently, we can state the tick data decomposition problem as follows.

Problem 1 *For a given number of clusters k, we aim at finding a decomposition so that the total number of the cells in all the matrices M_i is minimized.*

The above problem statement directly gives two variants of the tick data decomposition problem: while counting the number of cells in the matrices M_i, we can either count the cells in the index column or not.

*We thank DAAD and Magyar Ösztöndíj Bizottság (MÖB) for supporting the researcher exchange program between the University of Eichstätt-Ingolstadt and the Budapest Univ. of Techn. and Economics (project No. 39859). Research partially performed within the framework of the grant of the Hungarian Scientic Research Fund (grant No. OTKA 108947).



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	☀

Figure 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 (section a). Decomposition of such tables by features (columns) that change their values simultaneously may substantially reduce the required storage space (section b).

Furthermore, the above problem statement implicitly assumes uniform storage costs for all the cells, as it simply targets to minimize the number of cells in the decomposition. Other variants of the tick data decomposition problem may not assume uniform storage cost for each cells.

We note that k is usually relatively small: for example, for the storage of tick data of financial transactions, the user is most interested in the decomposition into $k = 2$ or $k = 3$ clusters. This is because, in case of real data, according to our observations, the decomposition into two or three partitions already leads to substantial gain in terms of storage space, and the decomposition into more partitions do leads to only minor further improvements, whereas the average computational costs of a query may grow with increasing k, see also [1].

In our previous work, we proposed an iterative, greedy algorithm for the tick data decomposition problem [2]. In the first iteration, this algorithm considers each column as a separate partition, then, in each iteration, it merges those two partitions that lead to optimal storage size. In [1], we gave a computationally cheap lower bound for the storage size in order to speed up the algorithm.

Despite its relevance from the point of view of applications, the theoretical foundations of the tick data decomposition problem are largely unclear and the authors are not aware of other combinatorial optimization problems that are equivalent to this problem. Therefore, in order to motivate discussions, we pose the following open questions related to the tick data decomposition problem:

1. Under which assumptions is it *possible* to find a good decomposition of a tick data table, i.e., a decomposition that leads to substantial improvements in terms of storage size?
2. What is the complexity of the tick problem? Depending on the assumptions about the data, are there cases in which the optimal decomposition is "simple" (or "difficult") to find?
3. In which cases do simple greedy algorithms find the optimal, or close to optimal decompositions?

Similar questions were successfully studied in context of various optimization problems resulting in celebrated results such as the theorems related to bin packing or Kruskal's algorithm for searching for the minimal spanning tree in graphs. We hope that the study of the above questions may contribute to establish the theoretical framework of the tick data decomposition problem.

References

[1] Krisztian Buza, Gábor I. Nagy, Alexandros Nanopoulos: Storage-optimizing clustering algorithms for high-dimensional tick data, Expert Systems with Applications, Volume 41, Issue 9, 2014, pp. 4148-4157

[2] Gábor Nagy, Krisztian Buza: SOHAC: Efficient storage of tick data that supports search and analysis, Advances in Data Mining, Applications and Theoretical Aspects, Springer, Berlin Heidelberg, 2012, pp. 38-51

[3] Gábor Nagy, Krisztian Buza: Partitional clustering of tick data to reduce storage space, 16th International Conference on Intelligent Engineering Systems (INES), IEEE, 2012