On Complexity of Effective Data Granulation in Databases

Jakub Wróblewski* and Marcin Kowalski†
*Infobright Inc. ul. Krzywickiego 34/219, 02-078 Warsaw, Poland
†Institute of Mathematics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
jakub.wroblewski@infobright.com, mkowal@mimuw.edu.pl

***Abstract*—We present the problem of splitting objects from a finite set into heterogenous groups of equal or almost equal cardinalities. The problem resembles the classic problem of data clustering, but additional constraints on the groups' size and global measure of clustering quality make well known clustering algorithms inapplicable. We believe that such a problem occurs in many important aspects of computer science. We will consider its context in the area of databases and reference a "body of research": the Infobright RDBMS. We've noticed that the problem emerges naturally in the use and design of Infobright's technology. In this paper we define the problem and present some specific applications. The main contribution of the article is the proof that the problem is NP-hard.**

***Index Terms*—Analytic Databases, NP-hard Problem, Granulating, Outliers.**

I. Introduction

We consider a problem similar to classic task of clustering. However, we make additional assumptions about the equal cardinality of clusters. This means in particular that – in contrast to classic approach – identical objects might not be grouped in the same cluster. This extra assumption makes state-of-the-art heuristics and complexity results inapplicable (see e.g. [3]). Moreover, our clustering quality is measured by some special function, which takes into account some global features of the clustering. We call this task *granulation* – which refers to the granular computing paradigm [8] – as we identify the resulting groups as the data granules. We find this problem very interesting and widely applicable. In particular, we have discovered that a similar problem is the crucial issue for the Infobright [10] database system.

In order to make readers more familiar with the subject and motivation of our research, we first introduce some primitives of Infobright RDBMS, as an example of a granular database system. In Section III we present use cases in which the discussed task is vital. In two next sections we formally define two versions of granulation problems and focus on proofs of their NP-hardness.

II. Granular Database System

The Infobright engine is a column oriented RDBMS based on the concept of *data packs* (DPs) which are disjoint subsets of consecutive values of one column from a given table (see Figure 1). All DPs – perhaps without the last one – are of equal cardinalities. During data loading they are parsed, independently compressed and stored on disk. All data packs

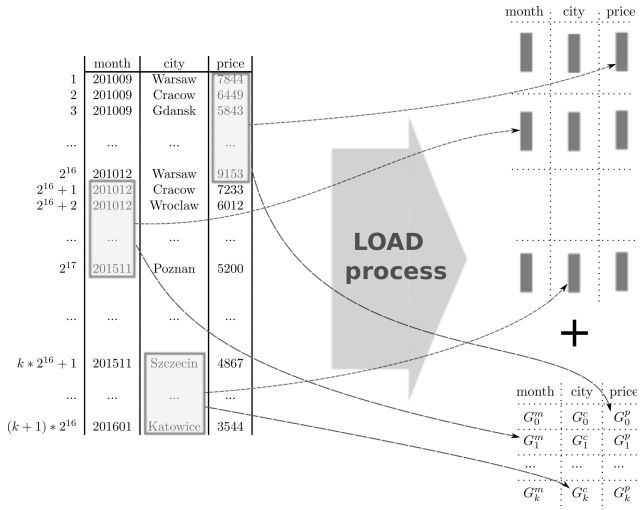


Fig. 1. Schema of loading data into granular engine – column values from input table are grouped into data packs of size 2^{16}, compressed and stored on disk; additionally some composite information about content of data packs is calculated (G_i) and stored on disk.

for respective rows in the table comprise a *row pack* (RP), i.e. a subset of consecutive rows of the same cardinalities (however, such a structure is only logical, as Infobright is a columnar database). Additionally, during the loading of data, some metainformation is calculated for each data pack, describing their contents. Such metainformations of various types, called *rough values* or *knowledge nodes* (KN), are of significantly smaller size than the original DP, containing column values. Each SQL query execution consists of calculations on two levels: the one performed only with KNs (we say that such calculations are done on the *rough level*), and the one using exact column values for some data packs. The latter means direct access to exact data, and is associated with reading DPs from disk, decompressing, and evaluating them. It should be clear that restraining the number of DPs accessed when performing exact level operations will often be crucial for good query evaluation performance.

A very simple (but still practically useful) example of a KN is min/max statistics, which define minimal and maximal values contained in the corresponding DP.

During query execution there are many optimizations performed based on information contained in KNs. For example,

978-1-4799-4143-8/14 $31.00 © 2014 IEEE
DOI 10.1109/WI-IAT.2014.119

analyzing only min/max values from DP, we can evaluate if there are some values in the DP which may satisfy some condition or not (e.g., the one from WHERE clause). Due to such analysis one may effectively reduce the number of costly I/O operations by omitting DPs which do not contain any relevant information wrt. executed query. There can be situations where the entire query result can be calculated without reading any DP from disk.

One may consider DPs and KNs as the "data granule" and "information granule" respectively. Therefore, support of KNs' information in the query evaluation process or database organization may be considered in terms of granular computing. That is why we might call Infobright the *granular RDBMS*.

A detailed example of using such granular calculations for query evaluation optimization (as presented in earlier works – see [10], [11]) is shown in Fig. 2.

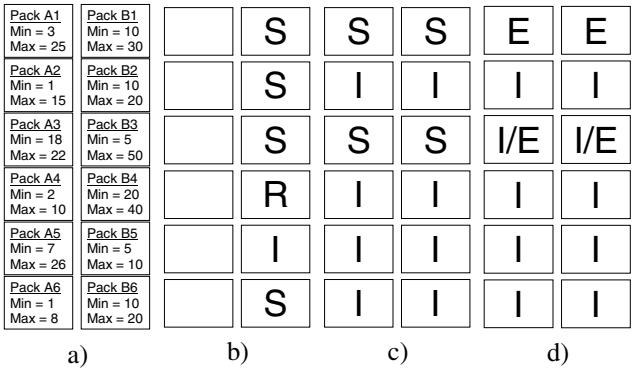


Fig. 2. (a) Simple KNs for columns A and B; (b,c,d) Query execution stages. RSI denote Relevant, Suspect and Irrelevant Data Packs. E denotes processing at the exact level.

Let's consider table `t` which does not contain NULL values, and the query:

```
SELECT max(A) FROM t WHERE B > 17;
```

In the first step, recognition of which DPs may contain useful information for query evaluation is done. During this stage, we distinguish 3 states of DPs: R - stands for $Relevant$, S - stands for $Suspect$ and I stands for $Irrelevant$. The first status means that every value in the DP satisfies the condition; the second one – some of the values may satisfy condition; the third one – none of the value satisfies condition. Simple checking min/max statistics for DPs of column B (presented in Figure 2a.) against predicate $B > 17$ lets us automatically infer that all values from B4 satisfy the predicate and no values from B5 do. For the remaining DPs, we cannot say for certain if they should be labelled with I or R, so they are classified as S. This stage is presented on Figure 2b. In next step, we can use additional information (both from KNs and gained in the first stage). For example, notice that the queried $max(A)$ has to be at least 18 while in third RP – consisting of $A3$ and $B3$ – there exist values satysfying the condition (maximal B in this RP is 50, and we assumed no NULL values in any DPs) and the range of column A in this RP is [18; 22]. This information allows to exclude form our search 2nd, 4th and 6th RP (they contain values of A less than 18, so they are guaranteed to not contain the maximum value). This situation is illustrated on Figure 2c. The last example of utilizing information from a KN is presented in Figure 2d. Having two RPs suspected of containing the queried maximum we can still utilize information from KNs to determine sequence of decompressing DPs (for exact evaluation) in order to increase chance of finding the result more quickly . Note that in case c) if we checked first $A3$, we will have to always decompress also $A1$ (because there can still exist value of A which is greater than maximal value from $A3$), while oppositely: if we first run exact calculations (marked as E on Figure 2d) on the 1st RP and if the maximal value among rows satisfying condition is greater than 22, exact calculations on 3rd RP are superfluous (i.e. no value greater than 22 can be found there). That is why in an optimistic scenario in order to achieve query result, it suffices to open only one RP (two DPs), and in a pessimistic scenario – just two RPs.

III. Database Use Cases

A. Use Case 1: Granulating On Load

The analysis presented in Figure 2 indicates that strength of leverage of rough level (or granular) computing on filtering task depends directly on quality of KNs identified with their informativeness. Indeed – checking predicates against min/max-KNs of DPs containing values from whole range of column will always be futile (apart the trivial case when one ask about value from outside the column range). Effectiveness of any other optimization based on sets of KNs will also suffer from their poor quality.

The definition of KN quality can be considered in a few ways. The simplest case of KN – min/max statistic from data pack looks quite clear here. It is believed that its quality should be connected with span of data packs values. It is however not so clear in real world, as we consider large volume of loaded data (e.g., terabytes stored in data warehouses), limited resources available and limited knowledge of data themselves. Usually, one cannot afford reading input data even twice. That is why task of data granulation – analogous to task of data clustering – will require some methods from the world of data streams [2]. We cannot say in advance what is the "compact" span of data pack while we cannot say much about range of values appearing in data stream. It is clear that problem of elaborating proper KN's quality formula may be interesting itself. In particular, when one consider more compound types of KNs such as histograms (see [11], [13] for details) or KNs for non-numeric columns. In the paper we simplify these aspects and we focus on case when all data are known.

After determining KN quality formula, we face the question of influence on the quality of KN or set of KNs for loaded data. Having in mind methods of constructing KNs and loading data in specified database system, the natural way of improving the quality of KNs is reorganization of input data during load.

The first guess in the situation may be sorting data by subset of columns. If doable, that might be partly effective and sometimes sufficient enough. In the case we get usually the most informative KNs on column which was the sorting

key, usually at the expense of losing quality of KNs for the rest of columns. Some database engines base their performance on maintaining a few copies of the same data sorted by different subsets of columns [1].

If one cannot afford data sorting, or wants to sustain the quality of KNs for all columns, or just this is unfeasible because of data volume or characteristics, then some methods of grouping (*granulating*) input data onto groups of equal size should be applied. Results of some exemplary grouping is presented in the Figure 3.

AMOUNT	PRICE
2	200
10	20
20	5
30	250
100	10
150	100
200	50
1000	210

AMOUNT	PRICE
20	5
100	10
10	20
200	50
150	100
2	200
1000	210
30	250

AMOUNT	PRICE
20	5
10	20
150	100
1000	210
100	10
200	50
30	250
2	200

Fig. 3. Three different organizations of data in the table (sorting by some chosen columns and exemplary granulation) and their influence on ranges of values in DPs

In the example of RPs consisting 2 rows each, we can see that sorting wrt. arbitrary column worsen compactness of DPs of the other one. In granulation case, we may perform better regarding to a general KNs quality.

As mentioned above, the main target of data granulation task may be such reorganization of data that ranges within DPs of important columns are as compact as possible. One of the crucial issues in any kind of granulation algorithm of this kind will be the recognition and handling *outliers* in data during loading process. Outliers are the observations (rows in table) with extreme values on a given attribute (column in table). Such extreme values may be expressed in terms of numerical data (e.g., values outside a "typical" domain values), as well as exceptional date/time values, rare texts, invalid data, or even NULL (unknown) values. Their assignment to concrete RP leads to deterioration of the KN quality for the DPs affected by extreme values (we would call them "broken" DPs to distinguish from the rest – "good" DPs). Exemplary target of data granulating algorithm may thus be minimizing number of "broken" DPs, i.e. grouping all outliers in RPs in such way that minimize the number of "broken" DPs.

In general however, we will say that the main task will remain reorganization of rows in tables that maximizes aggregated KNs' quality of set of DPs determined by the partitioning. Eventually, the quality of data granulation should reflect general performance of database system, where ad-hoc SQL queries are processed and virtually any column may be used for data filtering. In such case a number of "good" data packs for all columns should be maximized, as it reflects a number of data packs typically resolved on rough level (i.e. based on statistics only, without data access).

B. Use Case 2: Splitting task into subtasks

Another example of application of algorithms that split object in equally sized internally homogenous and externally heterogenous groups emerges when one consider multimachine query execution.

While we encounter that basic entity in granular engine is DP, we may like to split the task into subtasks solved by separate "workers" (threads, machines) designating them a set of DPs for evaluation. If we want to minimize the time spent on query evaluation and the "workers" are of similar computational power, we would like to split DPs onto approximately equal subgroups (each subgroup will be then designated to and evaluated by a selected worker).

For some types of queries (e.g., simple table scans) such division may be the simpliest possible (e.g., just calculate the number of RPs per worker and designate the RPs sequentally). However for some types of queries one would like to make additional assumptions on the partitioning.

When considering aggregation query, when some hashing mechanisms are involved and partial results are sent via network, there are situations that it is advisable to limit a size of data sent. On the one hand we would like workers to work with the same number of DPs, on the other hand we want partial results (in the form of groups determined by the grouping attributes/columns) to be as much homogenous as possible. That allows us to apply effective encoding of group values and make data sent via network smaller; moreover, if the groups are nearly disjoint, then effective merging of partial results is possible.

If we assume that from the range of values contained in DP we are able to estimate distribution of groups in partal results, we face the problem of splitting intervals (DPs' ranges) into homegenous subgroups. This use case differs from the first one as we are considering grouping of whole DPs (intervals), not particular rows of table, but the underlying idea remains unchanged.

C. Use Case 3: Data Compacting

During database life there are many operations like inserts/deletes/updates on selected table performed. They usually influence the quality of KNs assigned to the table. Let's consider small batch of new rows inserted into table. Without going into implementation details, if the number of newly inserted rows is smaller than row pack size, applying granulating on load (use case 1) may not be possible. If we consider series of such small batches one may face the problem of poor quality of KNs regarding newly created RPs. In such case we may want to leverage some techniques of rows reorganization analogous to granulating on load, but applied only to a limited number of already loaded RPs.

We presented background of data granulation problem within a world of databases. In next sections we will formalize such notions as "good pack" or "narrow data span" in order to analyze complexity aspects of the problem in its two variants: outliers sieving and maximizing number of compact DPs.

IV. Outliers Sieving

As outliers we understand observations/rows dissimilar to the others in some way. As pointed out in previous section it may not be strictly connected with some numeric values, but it may be related e.g. to NULL value containment (see e.g. [12]). Assume that we can assign each value of the given attribute/column to either 1 if it is an outlier, or 0 otherwise. In other words we will consider observations/rows as binary vectors consistently with the following:

Definition 1: Let $U \subset \{0,1\}^m$, $|U| = n$, where n is dividable by k. Let $P_1 \cup ... \cup P_l = U$ be a partition of U into disjoint subsets of size k, i.e.: $|P_1| = ... = |P_l| = k$, $\forall_{a \neq b} P_a \cap P_b = \emptyset$.

By **optimal division into packs** we call a partition $P_1 \cup ... \cup P_l = U$ such that:

$$\sum_{\substack{(u_1,u_2,...u_k) \in P_i \\ i \in \{1,...l\} \\ j \in \{1,...m\}}} (u_1(j) \vee u_2(j) \vee ... \vee u_k(j)) \longrightarrow min$$

i.e. minimizing a number of k-tuples containing values from a partition i for a dimension j such as at least one value is 1.

For $k = 2$ the problem is solvable in polynomial time, which is a corollary of the following observation:

Lemma 1: For a given instance of optimal division into packs for $k = 2$, let $G = (V, E)$ be a weighted (by a function $w : V \to \mathbf{R}^+$) full graph defined as follows:

$$\begin{aligned} |V| &= n \\ E &= V \times V \\ w(v_i, v_j) &= \textstyle\sum_{s=1}^{m} (1 - u_i(s))(1 - u_j(s)) \end{aligned}$$

i.e. the weight of an edge corresponding to objects u_i, u_j is a number of dimensions for which both objects have value 0.

Then a maximum weighted matching of graph G corresponds to optimal division into packs.

The maximum weighted matching for arbitrary graph is a polynomial task (e.g., Edmonds algorithm in $O(n^3)$, see [4]), although nontrivial. Thus, for $k = 2$ the corresponding optimal division into packs is also polynomial.

It is worth noting that an algorithm (or approximate heuristic) for $k = 2$ would be used to generate good division also for $k = 2^x$, although not necessarily an optimal one. Indeed, any pair of vectors $\{u_1, u_2\} = P_i$ from a good (or optimal) solution may be merged into $u^* = (u_1(1) \vee u_2(1), ..., u_1(m) \vee u_2(m))$ and the same algorithm may be applied iteratively.

However, we will show that for $k = 3$ the optimal division problem is NP-hard.

Definition 2: Let X, Y, Z be finite, disjoint sets. Let $A \subset X \times Y \times Z$ be a set of triples (without repetitions).

A subset $B \subset A$ is a **3-dimensional matching** if it contains triples which does not contain repeating elements, i.e. for any $(x_1, y_1, z_1) \in B, (x_2, y_2, z_2) \in B, (x_1, y_1, z_1) \neq (x_2, y_2, z_2)$ we have $x_1 \neq x_2 \wedge y_1 \neq y_2 \wedge z_1 \neq z_2$.

A problem of finding the largest 3-dimensional matching for a given $B \subset A$ is known to be NP-hard, and a problem of existence of a matching which cover $X \cup Y \cup Z$ is NP-complete [5]. Without loss of generality we may assume that $|X| = |Y| = |Z|$, as it is always possible to add arbitrary number of elements not covered by A, without changing the solution.

Lemma 2: Any maximum 3-dimensional matching problem instance may be encoded as an optimal division into packs where $|U| = n = |X| + |Y| + |Z|$ and $m = |A|$.

Proof: Let every element of U correspond to one element from X, Y, or Z. For convenience, we will number elements of U by double indices: u_{i1} corresponding to x_i, u_{i2} corresponding to y_i, u_{i3} corresponding to z_i. Dimensions of vectors u_{ij} correspond to triples from A. For every $A_k \in A$ the values of u_{ij} are defined as:

$$u_{i1}(k) = \begin{cases} 0 & \Leftrightarrow A_k = (x_i, *, *) \\ 1 & otherwise \end{cases}$$
$$u_{i2}(k) = \begin{cases} 0 & \Leftrightarrow A_k = (*, y_i, *) \\ 1 & otherwise \end{cases}$$
$$u_{i3}(k) = \begin{cases} 0 & \Leftrightarrow A_k = (*, *, z_i) \\ 1 & otherwise \end{cases}$$

Every dimension corresponds to a triple of elements, thus there is always exactly three 0 in a given column. Thus, any pack containing three 0 correspond to a triple of objects (u_{i1}, u_{j2}, u_{l3}) corresponding to a triple from A. On the other hand, in one block of partition P_i there may be only one pack of three 0, otherwise A would contain repetitions. Optimal division into packs will maximize a number of P_i containing three 0, which will also maximize a number of triples selected from A, i.e. a maximum 3-dimensional matching (see Fig. 4 for an example of encoding). Indeed, as every P_i contain distinct elements from U, the triples selected from A are disjoint.

■

As an immediate conclusion the optimal division into packs for $k = 3$ is NP-hard. Moreover:

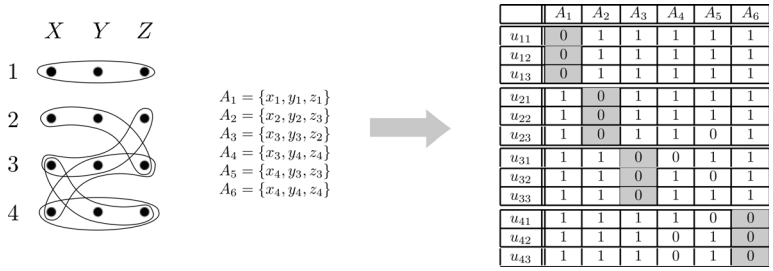
Lemma 3: Optimal division into packs for $k > 3$ is NP-hard.

Proof: We will reduce an instance of optimal division for $k = 3$ for a given U to the problem of optimal division for $k > 3$. Suppose that $|U| = 3n$; let $S = nm$ be an upper bound of a number of packs not containing ones in any possible solution (the bound is never actually met if there is any 1 in U). Let $U^* \subset \{0,1\}^{m+Sn}$, where $|U^*| = kn$, be defined as follows:

- for $i \leq 3n$ let $u_i^*(j) = u_i(j)$ for $j \leq m$ and 0 elsewhere,
- the remaining $(k-3)n$ objects are divided into groups $G_1, ...G_n$, each of these groups contain $(k-3)$ copies of the same object: if $u \in G_i$, then

$$u \in G_i \;\Rightarrow\; u(j) = \begin{cases} 0 & for\ j \leq m, \\ 1 & for\ j \in [m + (i-1)S, m + iS], \\ 0 & otherwise \end{cases}$$

Such construction guarantees that any solution which mix in one pack any pair of objects from different groups G_i, G_j



	A_1	A_2	A_3	A_4	A_5	A_6
u_{11}	0	1	1	1	1	1
u_{12}	0	1	1	1	1	1
u_{13}	0	1	1	1	1	1
u_{21}	1	0	1	1	1	1
u_{22}	1	0	1	1	1	1
u_{23}	1	0	1	1	0	1
u_{31}	1	1	0	0	1	1
u_{32}	1	1	0	1	0	1
u_{33}	1	1	0	1	1	1
u_{41}	1	1	1	1	0	0
u_{42}	1	1	1	0	1	0
u_{43}	1	1	1	0	1	0

Fig. 4. Encoding of 3-dimensional matching into optimal division problem.

is worse than one which does not. Maximal number of good packs from the $j \leq m$ part is less than S, and mixing groups will reduce the number of good packs in $j > m$ part by S. Thus, any optimal solution of this $k > 3$ problem should place all these n groups in separate packs (there are n packs in total). It means that every pack consists of exactly 3 original and $k-3$ additional objects, and the optimal division of the original set is a part of $k > 3$ solution. Solving $k > 3$ problem we will also solve the $k = 3$ subproblem, which proves that $k > 3$ case is also NP-hard.

■

V. Compact Intervals

As we've shown in examples in the previous sections, for numerical data it is crucial to keep data granules as "narrow" (in terms of minimal and maximal value in DPs) as possible. Such data granules have more precise and more unique description on rough (statistics) level and query engine can easily exclude such DPs without accessing actual data. Now we will formalize these notions.

Definition 3: Let $U \subset \mathbf{R}^m$, $|U| = n$. Let $P_1 \cup ... \cup P_l = U$ be a partition of U into disjoint subsets of size k, i.e.: $|P_1| = ... = |P_l| = k$, $\forall_{a \neq b} P_a \cap P_b = \emptyset$. For any subset $P \subset U$ and any $j \in \{1, ...m\}$, let:

$$Span(P, j) = max_{u \in P}(u(j)) - min_{v \in P}(v(j))$$

By **optimal division into packs wrt. interval span** for a given $\varepsilon \in (0, 1)$ we call such partition $P_1 \cup ... \cup P_l = U$, that:

$$\Big|\{(i,j) \in \{1, ...l\} \times \{1, ...m\} : Span(P_i, j) \leq \varepsilon \cdot Span(U, j)\}\Big| \longrightarrow min$$

i.e. minimizing a number of k-tuples for which a difference between maximal and minimal value is less than a given fraction of the whole domain for a dimension (we will call such tuple a **good pack**).

For $k = 2$ the problem is solvable in polynomial time by the Lemma 1, where the weight of an edge corresponding to objects u_i, u_j is defined as a number of dimensions for which the difference between maximal and minimal value is small enough.

To prove that for $k \geq 3$ the problem is NP-hard, we will use a *Maximum H-matching* problem: for a given graph $G = (V, E)$ and some constant graph H of at least three connected vertices, find possibly large set of disjoint subgraphs $H_1, ...H_j$, where H_i is isomorphic with H and $H_i \subset G$. It is an NP-hard problem (GT12 in [5], see also [6]). For a special case of $H = K_3$, the problem is known as *Maximum Triangle Packing* (GT11 in [5]).

Lemma 4: Any maximum H-matching problem instance for $H = K_k$ (i.e. k-clique) may be encoded as a task of optimal division into packs wrt. interval span for a given k.

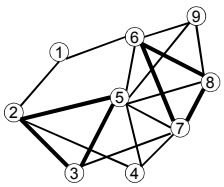
Proof: Suppose that $\varepsilon < 0.5$. We will encode our maximum H-matching problem into optimal division problem for U containing vectors of four possible values: 0, 1, 2, and $t = \lceil 1/\varepsilon \rceil + 1$. Note that in such case $t \geq 4$ and $1 < \varepsilon \cdot Span(U, j) < 2$ for any j, which means that good packs may not contain both 0 and 2, or both t and any other value.

Given an instance of H-matching problem for graph $G = (V, E)$ we will construct the following:

- $|U| = |V| + k$, where any row $u_i \in U$ corresponds to a vertex $v_i \in V$, except k additional rows $u_1^*, u_2^*, ...u_k^*$.
- Let T be a set of all possible k-cliques in G, without repetitions (note that this step is polynomial as k is fixed). Let $m = |T|$ be a number of dimensions (columns) of vectors in U, every dimension j correspond to a clique $T_j \in T$, $T_j = (V_j, E_j)$. Let v_{T_j} be an arbitrarily chosen vertex in clique T_j, e.g., one with the smallest number for some ordering of vertices of G.
- Values of vectors for all u except the three additional rows are defined as:

$$u_i(j) = \begin{cases} 2 & \Leftrightarrow v_i = v_{T_j} \\ 1 & \Leftrightarrow v_i \in V_j, v_i \neq v_{T_j} \\ 0 & otherwise \end{cases}$$

- Values of vectors for additional rows: $\forall_{j=1,...m} u_1^*(j) = ... = u_k^*(j) = t$.



$\mathbf{T_1 = \{2,3,5\}}$
$T_2 = \{2,4,5\}$
$T_3 = \{3,5,7\}$
$T_4 = \{4,5,7\}$
$T_5 = \{5,6,7\}$
$T_6 = \{5,6,8\}$
$T_7 = \{5,6,9\}$
$T_8 = \{5,7,8\}$
$T_9 = \{5,8,9\}$
$\mathbf{T_{10} = \{6,7,8\}}$
$T_{11} = \{6,8,9\}$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
u_1^*	6	6	6	6	6	6	6	6	6	6	6
u_2^*	6	6	6	6	6	6	6	6	6	6	6
u_3^*	6	6	6	6	6	6	6	6	6	6	6
u_2	2	2	0	0	0	0	0	0	0	0	0
u_3	1	0	2	0	0	0	0	0	0	0	0
u_5	1	1	1	1	2	2	2	2	2	0	0
u_6	0	0	0	0	1	1	1	0	0	2	2
u_7	0	0	1	1	1	0	0	1	1	1	0
u_8	0	0	0	0	0	1	0	1	0	1	1
u_9	0	0	0	0	0	0	1	0	1	0	1
u_1	0	0	0	0	0	0	0	0	0	0	0
u_4	0	1	0	2	0	0	0	0	0	0	0

Fig. 5. Encoding of H-matching into optimal division problem. In this example $k = 3$, i.e. T_i are triangles (listed on the left), and $\varepsilon = 0.2$, so $t = 6$. Good packs are shaded on the right, and maximal number of them corresponds to the maximal matching (bolded on the left).

Note that the additional rows $u_1^*, ..., u_k^*$ should be gathered in one partition set P^*: if so, they compose m good packs, and there is no gain if any of these rows is moved to other set P_i (we will loose not only these m packs from P^*, but also all potential good packs from P_i). Note that all other rows contain exactly one value 2 and $k-1$ values 1 in every column (the rest is 0). Thus, there are two possibilities: either there is a pack containing values $(2, 1, ..., 1)$ in any order, which is a good pack, or a pack containing 2 contains also 0. In the first case a column contains all good packs, in the second case there is exactly one pack which is not good. Note also that (as T does not contain repetitions) in one partition set P_i there is only one pack $(2, 1, ..., 1)$ possible.

Thus, optimal division into packs wrt. interval span for such U maximizes a number of packs containing values $(2, 1, ..., 1)$, which correspond to maximization of a set of disjoint k-cliques from T, which is a solution of maximum H-matching problem (see Fig. 5 for an example of encoding).

For a case of $\varepsilon \in [0.5, 1)$ the construction is similar, except there is no need to use additional rows $u_1^*, ..., u_k^*$ and the values of vectors in U may be limited to 0, 1, 2.

■

VI. CONCLUSIONS

We presented formal proofs of complexity of data granulation problem. Such problem may emerge in granular computing applications as well as in areas not connected directly to them. In particular we showed that this is a significant issue in the world of databases, namely granular RDBMS.

On the one hand, theoretical results close (or make questionable) way to search for algorithms finding optimal solution for the problem in reasonable time. However, on the other hand, it is still possible to design new heuristics, approximate algorithms or just modify problem's formulation. For example, if existing of incomplete DPs (i.e. of size less than desired) is permitted, it may be worth considering variant of the problem with maximal, not fixed, size of each group. It should not deteriorate KNs quality in comparison with original statement but may significantly improve it. Of course in this case some additional assumptions of minimal size of groups should be introduced too. We do not claim that such redefinition of the problem will make it easier to solve, but solution of such problem version may be profitable from practical viewpoint.

Also for the original formulation of the optimal division problem one may employ heuristics to tackle the task practically. Similarly to well known strategies of classic clustering, a greedy algorithm may be used to add a row to the best matching RP (in terms of compactness or outliers) until a desired size is achieved. Other strategies may be based on merging similar sub-RPs or exchanging rows between RPs if such operation improves overall quality.

We hope that both the subject matter and obtained theoretical results presented in the paper will increase interest in the problem and lead to developing useful and interesting algorithms to solve it.

REFERENCES

[1] D. Abadi, Query Execution in Column-Oriented Database Systems. MIT PhD Dissertation, February, 2008.
[2] C.C. Aggarwal (ed.): Data Streams: Models and Algorithms. Springer (2007).
[3] D. Aloise, A. Deshpande, P. Hansen, P. Popat (2009). "NP-hardness of Euclidean sum-of-squares clustering". Machine Learning 75: 245?-249.
[4] J. Edmonds (1965). "Paths, trees, and flowers". Canad. J. Math. 17: 449 96-467
[5] M.R. Garey, D.S. Johnson (1979). "Computers and Intractability: A Guide to the Theory of NP-Completeness". Freeman
[6] P. Hell, D. Kirkpatrick (1978). "On the complexity of a generalized matching problem". Proceedings of 10th ACM Symposium on theory of computing, pp. 309–318.
[7] R.M. Karp (1972). "Reducibility Among Combinatorial Problems". In R. E. Miller and J. W. Thatcher (ed.), Complexity of Computer Computations. New York: Plenum. pp. 85 96-103.
[8] W. Pedrycz, A. Skowron, V. Kreinovich, eds.: Handbook of Granular Computing. Wiley-Interscience (2008).
[9] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, S. Zdonik: CStore: A column oriented DBMS. In: Proc. of VLDB (2005) pp. 553–564.
[10] D. Ślęzak, J. Wróblewski, V. Eastwood, P. Synak: Brighthouse: an analytic data warehouse for ad-hoc queries. PVLDB 1(2): 1337–1345 (2008).
[11] D. Ślęzak and M. Kowalski (2009), "Intelligent Data Granulation on Load: Improving Infobright's Knowledge Grid", FGIT, pp. 12–25
[12] D. Ślęzak and M. Kowalski, "Intelligent granulation of machine-generated data", IFSA/NAFIPS 2013: pp. 68-73
[13] D. Ślęzak, M. Kowalski, V. Eastwood and J. Wróblewski, Methods and Systems for Database Organization, US Patent 8,266,147 B2 (2012).