

A new approximate query engine based on intelligent capture and fast transformations of granulated data summaries

Dominik Ślęzak¹  · Rick Glick² · Paweł Betliński³ · Piotr Synak³

Received: 24 March 2017 / Revised: 14 June 2017 / Accepted: 15 June 2017 /
Published online: 5 July 2017
© The Author(s) 2017. This article is an open access publication

Abstract We outline the processes of intelligent creation and utilization of granulated data summaries in the engine aimed at fast approximate execution of analytical SQL statements. We discuss how to use the introduced engine for the purposes of ad-hoc data exploration over large and quickly increasing data collected in a heterogeneous or distributed fashion. We focus on mechanisms that transform input data summaries into result sets representing query outcomes. We also illustrate how our computational principles can be put together with other paradigms of scaling and harnessing data analytics.

Keywords Big data processing · Approximate query processing · Data summarization · Granular computing · Machine intelligence

1 Introduction

There is a growing need to explore big data sets. Most companies address this challenge by scaling out resources. However, this strategy is increasingly cost-prohibitive and inefficient for large and distributed data sources. On the other hand, people are realizing that the tasks of data exploration could be successfully performed in at least partially approximate fashion. This way of thinking opens new opportunities to seek for a balance between the speed, resource consumption and accuracy of computations. In the case of an approach introduced

✉ Dominik Ślęzak
slezak@mimuw.edu.pl

¹ Institute of Informatics, University of Warsaw, Warsaw, Poland

² Independent Consultant, San Diego, USA

³ Independent Consultant, Warsaw, Poland

in this paper – an engine that produces high value approximate answers to SQL statements by utilizing granulated summaries of input data – such a new kind of balance possesses very attractive characteristics.

The presented engine captures knowledge in a form of single- and multi-column data summaries. It collects chunks of newly gathered data and builds summaries for each chunk separately. Unlike in standard databases, the considered query execution mechanisms do not assume any access to the original chunks. Those chunks can be available in a broader application framework but the primary goal of our engine is to work with summaries, as illustrated by Fig. 1. Namely, for a given query received from an external tool, each consecutive data operation (such as filtering, joining, grouping, etc.) scheduled within the query execution plan is performed as a *transformation* of summaries representing its input into summaries representing its output.

The engine allows its users to achieve approximate – yet sufficiently accurate – analytical insights 100-1000 times faster than traditional solutions. From an architectural perspective, it comprises two layers: 1) the knowledge capture layer responsible for software-agent-style acquisition of data summaries and 2) the knowledge transformation layer responsible for utilizing already-stored summaries to produce fast approximate answers to ad-hoc queries. The first layer looks through potentially distributed and heterogeneous data sources, leaving the actual data in-place. The second layer operates on the output of the first layer, removing the need of accessing the original data, which would be far too costly and simply not needed for analytical purposes.

Several organizations have been already working towards embedding our solution into their projects. Figure 2 illustrates its benefits in particular areas, where the emphasis is put on the speed of query-based analytical processes and the ability to integrate broader aspects of the data. Thanks to informative richness of summaries, the approximate results can be

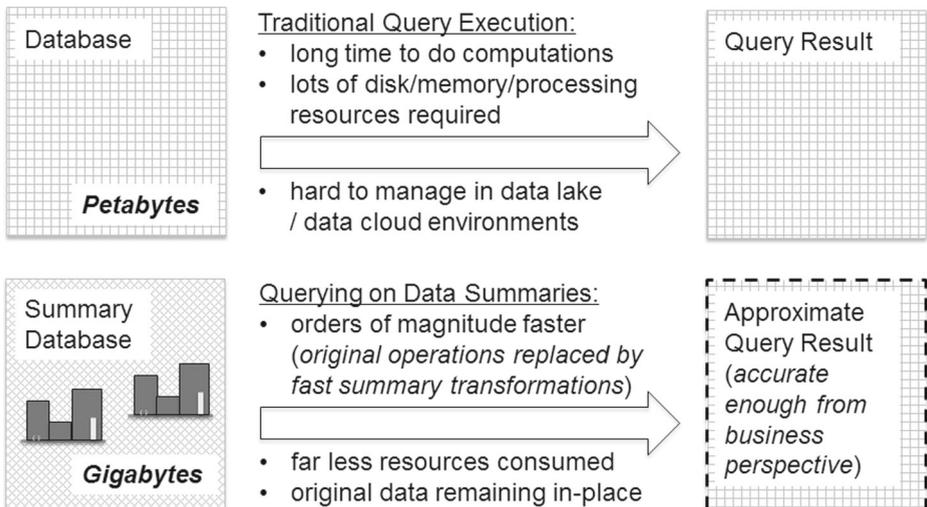


Fig. 1 High-level comparison of traditional approach to relational query processing versus our approach based on computing with data summaries

used for insightful decisions. Moreover, our analytical testing environment allows the users to investigate similarities of approximate answers compared to query outputs that would be extracted in a standard way from the atomic data.

Properties highlighted in Figs. 1 and 2 can be also considered from the perspectives of business intelligence or cognitive computing, where there is an ongoing discussion whether in all use cases query answers must be exact and whether one can afford to wait for them. Furthermore, in many scenarios, the analysis needs to be conducted over the data collected in a lake or cloud environment. Traditional solutions would then require moving / reorganizing large amounts of the data to make them efficiently accessible. The data sets might be queried in their original locations but this makes it hard to jointly analyze diverse data sources in single operations. Thus, the ability to work with easily manageable summaries brings a great advantage.

The paper is organized as follows. In Section 2, we discuss our previous developments and some state-of-the-art approaches in the area of approximate query processing. In Section 3, we discuss the details of the captured summaries and provide some means for their analysis. In Section 4, we describe examples of the knowledge transformation operations that produce new summaries reflecting intermediate stages of query execution. In Section 5, we discuss the final stage of assembling approximate answers, namely, generating query results from the produced summaries. In Section 6, we report our empirical experiences gathered to date. In Section 7, we outline some of other aspects still in development. Section 8 concludes the paper.

Although our main aim is to discuss the engine from a practical viewpoint, let us also emphasize that it relies on some significant scientific novelties. It is particularly visible in Section 4, where we adapt the mechanism of tree-based belief propagation to populate the WHERE-related changes in data summaries, by introducing new methods of dynamic derivation of optimal trees from input summaries, extending the standard propagation model to let it work with more complex filter conditions, using partial knowledge about data-derived joint probabilities to efficiently run recursive calculations and producing output summaries representing the filtered data for the purposes of further transformations reflecting execution of a given SELECT statement.

Use Case	Improvements
Intrusion Detection	faster analytics ⇨ ⇨ improved reaction time ⇨ improved customer retention
Digital Advertising	richer sources of analytics ⇨ ⇨ improved quality of customer profiles ⇨ increased click-thru customer revenue
Sensor-based Monitoring of Industry Processes	faster/deeper machine learning ⇨ ⇨ improved risk prediction efficiency ⇨ lower cost of incorrect predictions

Fig. 2 Examples of use cases related to deployments of analytic databases (Glick 2015)

2 Related works

In this section, we first refer to our own previous experiences and then proceed with a brief discussion related to other methodologies, as well as a more general scope of approximate querying in data analytics.

2.1 Historical background

To some extent, the approach introduced in this paper originated from the architecture of our earlier database engine, which clustered the incoming rows into so-called packrows, further decomposed into data packs gathering values of particular columns (Ślęzak et al. 2013). In that earlier framework, packrows were described by simple summaries accessible independently from the underlying data. It combined the ideas taken from other database technologies (Abadi et al. 2013) and the theory of rough sets (Pawlak and Skowron 2007), by means of using summaries to classify data packs as relevant, irrelevant and partially relevant for particular `SELECT` statements – by analogy to deriving rough set positive, negative and boundary regions of the considered concepts, respectively. Such higher-level classifications were useful to limit the amounts of compressed data packs required to access to finish calculations. Actually, one could think about it as production-ready example of a granular data analytics mechanism switching efficiently between the detailed and summary-based data representations – a mechanism envisioned in a number of studies on the principles of granular computing (Yao 2016).

Our previous engine included *rough query* functionality developed for the purposes of both external usage and internal query execution accelerations (Kowalski et al. 2013). The idea was to quickly deliver some bounds for actual query results. However, it was hard to configure other tools to work with such a new kind of output syntax. External tools would expect the engine to use its rough query capabilities to generate approximate results in standard format. Somewhat in parallel, we encouraged the academic community to design summary-based machine learning and knowledge discovery methods (Ganter and Meschke 2011). However, once the original data access was disallowed, summaries stored within our previous framework could not provide the new versions of machine learning algorithms with sufficient information to make their results truly meaningful.

The above lessons – related to non-standard format of rough query results and limited usefulness of the aforementioned engine's summaries – had a significant impact on the design presented in this paper. We decided to continue our works in this area because of the rapidly growing challenges of standard database system users, not only with analyzing but also simply managing their data. We followed the same strategy of grouping rows into packrows as previously. However, to make it really efficient, we needed to solve two problems: 1) how to build compact summaries that contain enough knowledge about the original packrows and 2) how to accurately perform operations on those summaries thereby removing the need for access to the actual data.

2.2 Literature overview

Approximate query processing is already a popular trend in data analytics. Exact results of database computations are not always a must, e.g., for the purposes of reporting, visualization,

trend analysis, event detection, or decision making in general. By analogy to modern image processing, approximate outcomes of data operations are acceptable, if they enable the users to validly *perceive* and deploy data-derived knowledge. One may look at our algorithms also from the perspective of information granulation and granular computing, wherein the major rationales are that: 1) crisp, fine-grained information is often not available, 2) precise information is costly, 3) fine-grained information is not necessary and 4) coarse-grained information reduces cost (Zadeh 1997). Furthermore, approximate query processing corresponds to the area of approximate computing, whereby the primary goal is to determine what aspects and degrees of approximations are feasible so that the produced results are acceptable (Agrawal et al. 2016). Actually, quite many components of the approach considered in this paper, such as reducing the length of heaviest algorithmic loops, quantizing input information, as well as loosening the assumptions about completeness and consistency of the analyzed data sources, could be compared to some of the most fundamental principles of approximate computing.

There are several ways to develop approximate SQL solutions. In most approaches, the results are estimated by executing queries on collections of intelligently derived data samples (Mozafari and Niu 2015). One of advantages of such approaches is their ability to adapt statistical apparatus to deliver confidence intervals for approximate outcomes. For that reason, we also considered introducing sampling into our earlier developments (Kowalski et al. 2013). However, for truly big data sets, good-quality samples will need to be quite large too. This surely limits query acceleration possibilities, not mentioning about other sampling challenges related to multi-table joins, handling important outlying values, etc.

The second category of approximate query approaches is based on summaries (histograms, sketches, etc.) (Cormode et al. 2012). These two types of data synopses – samples and summaries – are to some extent combinable (Ramnarayan et al. 2016). However, the solutions developed so far build summaries for predefined query configurations or, e.g., OLAP-specific scenarios (Cuzzocrea and Saccà 2013). This limits their usefulness for exploratory analytics, where – by default – it is hard to anticipate queries that will be executed (Nguyen and Nguyen 2005). Because of this, it is important to emphasize that the way we construct and utilize data summaries makes the engine introduced in this paper fully applicable for ad-hoc analytical SQL statements.

Our summaries are expressed by means of histograms, whereby there is a long tradition of their use within standard relational database optimizers (Jagadish et al. 1998). This is a great source of inspiration when developing new engines such as ours. In the literature, a lot of effort has been spent on addressing a need of updating histogram structures while loading new data (Gibbons et al. 2002). However, this is one aspect where our approach is different. This is because we build separate summaries for each subsequently ingested data chunk. Hence, the newly buffered packrows do not interfere with the previously captured knowledge structures.

There is also a significant amount of research related to multi-dimensional summaries, although their derivation and utilization is far more complex than for single columns. Histograms reflecting intensities of co-occurrences of values of different columns are a good starting point in this field (Bruno et al. 2001). One can also refer to some examples of utilizing more complex multi-dimensional summaries, e.g., tree-based structures that can be applied in a progressive / iterative framework for approximate querying (Lazaridis and Mehrotra 2001).

Compared to the aforementioned approaches, we operate with relatively simple multi-column representations that are easy to embed into our knowledge transformation mechanisms yet they contain sufficient information about co-occurrences of values in the original data. Let us re-emphasize that our transformations work in a loop over the captured packrow summaries. This gives us a truly valuable opportunity to model a potential drift of occurrence and co-occurrence characteristics with respect to time and origin of the continually ingested data sources.

It is also important to realize that approximate and traditional styles of querying can be used within the same application frameworks. Let us refer again to the tasks of data exploration, where it is quite common to begin an analytical process with approximations and finish with their thorough validation. With this respect, it is worth referring to methods that support incremental and interactive computations. Herein, let us refer to paradigms and tools related to evolution of complex SQL execution processes (Hu et al. 2007) and interactive visualization / representation of analytical query results (Fisher et al. 2012).

In addition to savings in execution time and resource consumption, there are also other motivations for approximate querying. Some of them are related to the fact that in dynamic environments the data may evolve too fast to care about exact answers or – in other cases – some data fragments may be temporarily unavailable (Narayanan et al. 2008). Yet another challenge – analogous to the one encountered by search engines – is that available data items and query conditions often do not perfectly match each other. In such situations, one may wish to retrieve *approximately* fitting items (Naouali and Missaoui 2005). However, such approaches usually require modifications of query syntax. This can be problematic in practice, like in the case of changing standard format of query outcomes.

3 Knowledge capture settings

In this section, we describe the methods designed to fill the engine with meaningful knowledge about the original chunks of data, i.e., packrows which are collections of rows acquired from the original data tables.

3.1 Histograms, special values and domain representations

The knowledge capture layer needs to assess what aspects of the ingested data will be most useful later for approximate query algorithms. This may turn out to be even more important than the ability to store and process the original data, especially given the fact that perfect data access is questionable in many environments (Narayanan et al. 2008). For the knowledge representation purposes, we utilize enhanced histogram structures (Cormode et al. 2012). For each original data pack, its histogram contains information about dynamically derived range-based bars and *special values* that differ from neighboring values of the given column by means of their frequencies in the given packrow. We also store information about significant *gaps*, i.e., the areas where there are no values occurring. This is further used as a foundation for multi-column summaries capturing packrow-specific co-occurrences of special values or values belonging to histogram ranges of different data columns. The developed methods decide what is worth storing based on a significance ranking of the detected co-occurrences.

Algorithm 1 histogram construction

Input: *list* : sorted list of values *v* of column *c* in packrow *t* in table *T*;
max_no_of_areas : maximum number of equal-length long intervals (8 by default);
max_no_of_bars : maximum number of resulting histogram bars (64 by default);
Output: *cuts* : the list of cut-points between bars describing *c* in packrow *t*;

- 1: **begin**
- 2: **if** the number of distinct values in the list \leq *max_no_of_bars* **then**
- 3: add to *cuts* all elements of *list* except $\max(\text{list})$;
- 4: **else**
- 5: split $[\min(\text{list}), \max(\text{list})]$ onto *max_no_of_areas* intervals of equal length;
- 6: remove intervals, which do not contain any elements of *list*;
- 7: shift intervals' right and left sides to their closest leftmost elements of *list*;
- 8: add all intervals' right sides except $\max(\text{list})$ to *cuts*;
- 9: *no_of_waiting_areas* \leftarrow the number of intervals;
- 10: *no_of_undefined_bars* \leftarrow *max_no_of_bars*;
- 11: **for all** intervals sorted by the number of elements of *list* that they contain **do**
- 12: # \leftarrow the number of elements of *list* contained in the considered interval;
- 13: **if** # \leq $\lfloor \text{no_of_undefined_bars} / \text{no_of_waiting_areas} \rfloor$ **then**
- 14: add to *cuts* all elements of *list* contained in the considered interval;
- 15: **else**
- 16: # \leftarrow $\lfloor \text{no_of_undefined_bars} / \text{no_of_waiting_areas} \rfloor$;
- 17: split the considered interval onto # bars with roughly uniform supports;
- 18: and cut-points between the obtained bars to *cuts*;
- 19: **end if**
- 20: *no_of_waiting_areas* \leftarrow *no_of_waiting_areas* - 1;
- 21: *no_of_undefined_bars* \leftarrow *no_of_undefined_bars* - #;
- 22: **end for**
- 23: **end if**
- 24: **end**

Algorithm 1 outlines how we derive cut-points between consecutive histogram bars. This is a simple combination of two standard domain quantization techniques. However, out of all procedures that we have examined by now, it yields the highest accuracies of approximate queries over data sets with complex column characteristics. First, the domain of a data column within a given packrow is decomposed onto a fixed number of intervals of (almost) equal length (lines 5-7). Then, the algorithm allocates (almost) equal amounts of wanted histogram bars per interval and – for each interval separately – it produces the final bars attempting to equalize frequencies of occurrence of rows with values dropping into particular ranges (lines 16-18).

Figure 3 shows the representation of a data pack corresponding to “intersection” of a packrow and a column. Once cut-points between histogram bars are decided, the knowledge capture algorithms prepare heuristic rankings of candidates for special values and gaps. Only a limited number of the highest-ranked values and gaps per data pack can be stored, to address the need for a reasonable trade-off between summary footprint (translatable to the expected query execution speed, by means of runtime of algorithmic *hot loops* that transform data summaries) and completeness of gathered knowledge (translatable to the expected accuracy of approximate query outputs).

Special values are chosen to minimize an average error related to estimating frequencies of the original values. It resembles classical methods of histogram variance minimization (Jagadish et al. 1998), although some modifications were required for columns with

Examples of captured knowledge:

- Value 300 occurred 1120 times
- There were 4570 occurrences of values between 200 and 350 (including value 300)
- There were no occurrences of values between 40 and 60
- Values 0, 40, 60, 100, 200, 350 occurred at least once

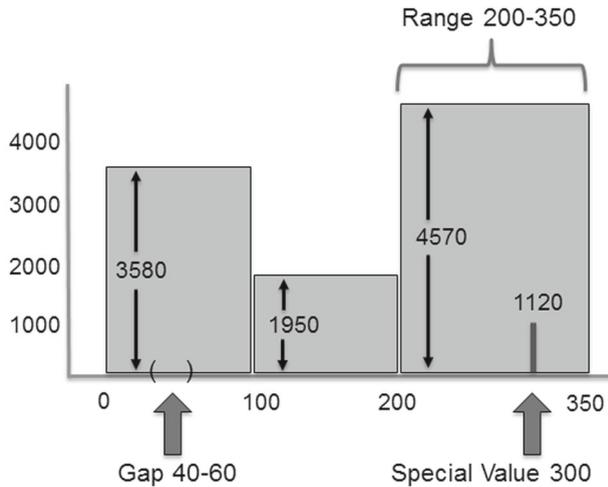


Fig. 3 A summary of a data pack corresponding to a single column, over a single packrow. Histogram bars are composed of ranges and frequencies. Frequencies of special values are represented by additional bars within histogram bars. Algorithms for deriving ranges and gaps are designed so as to assure that their borders actually occurred in the data

irregular domains. Similar ranking is created for gaps. Any value occurring in a given pack can be a potential special value. Any interval between consecutive (along an ordering defined for a given domain) values occurring in a data pack is a candidate for a gap. However, only the most significant gaps and values can be explicitly represented. Further discussion about heuristic ranking functions that are worth considering during the process of creation of single-dimensional summaries can be found, e.g., in Chądzyńska-Krasowska et al. (2017).

Gaps play an important role in estimating local value distributions within histogram ranges. As it will be clearly visible in Section 4, such estimations are crucial for proper behavior of data summary transformations and final query result generation. For instance, referring to Fig. 3, we know that in a given packrow, over a given data column, there were 3580 rows with values dropping into the range between 0 and 100. Let us assume that the greatest common divisor stored for the corresponding data pack (denoted as the field *gcd* in Fig. 5 described in Section 3.3) equals to 2. Thus, if a given query execution requires the estimation of the number of distinct values occurring in the considered range, then knowledge about a gap between 40 and 60 may improve that estimation by avoiding counting 9 potential values that did not occur in the original data.¹ Knowledge components related to occurrence of distinct values will be further referred to as domain representation.

3.2 Co-occurrences of histogram bars and special values

The remaining task is to describe co-occurrences between (ranges of) values of different columns. This is quite a significant differentiator in our approach when compared to other

¹Gaps are interpreted as open intervals.

methodologies of utilizing summaries in approximate query processing. The key objective is to set up a multi-column representation that is simple enough to operate with at the levels of both knowledge capture and query-specific knowledge transformations. This drove our decision to restrict ourselves to modeling co-occurrences for only pairs of columns and represent such pairwise relationships only in a partial way. Although one might say that such a model cannot be sufficiently informative, remember that it is applied locally for each of separate packrows. This allows our ranking algorithms to focus on different pairs of columns and their corresponding values for different fragments of the ingested data, leading towards ability to express far more complex trends and patterns than one might expect.

To keep a low footprint of summaries, we store co-occurrence-related information only for a limited number of important pairs of bars and special values. For packrow t and columns a and b , let us refer to a 's and b 's histogram bars using iterators i and j , respectively. Let us denote by $p_t(r_i^a[i])$, $p_t(r_i^b[j])$ and $p_t(r_i^a[i], r_i^b[j])$ data-derived probabilities of occurrence of a 's values within its i -th bar, b 's values within its j -th bar and pairs of a 's values within its i -th bar and b 's values within its j -th bar, respectively. We use the following ranking function to express the importance of pairs of histogram bars:

$$bars_t(r_i^a[i], r_i^b[j]) = p_t(r_i^a[i]) p_t(r_i^b[j]) \left| \frac{p_t(r_i^a[i], r_i^b[j])}{p_t(r_i^a[i]) p_t(r_i^b[j])} - 1 \right| \tag{1}$$

Formula (1) evaluates how much accuracy could be lost by a query execution process based on the product $p_t(r_i^a[i]) p_t(r_i^b[j])$ instead of the actual quantity of $p_t(r_i^a[i], r_i^b[j])$. For a given packrow t , we use function $bars_t$ to jointly evaluate all pairs of bars for all pairs of columns. This results in devoting relatively more footprint to pairs of columns, which are more interdependent than others. After choosing a fixed amount of the most important pairs of ranges, for given columns a and b , we store two types of information. For the chosen pairs $(r_i^a[i], r_i^b[j])$, we store the following co-occurrence ratios:

$$\tau_t(r_i^a[i], r_i^b[j]) = \frac{p_t(r_i^a[i], r_i^b[j])}{p_t(r_i^a[i]) p_t(r_i^b[j])} \tag{2}$$

For not chosen pairs, we need an approximate average ratio for the purposes of, e.g., our WHERE-related version of the belief propagation process (Neapolitan 2003) (see Section 4). For the pairs of histogram bars, such default ratio (denoted by *default_bar_bar_tau* in Fig. 5) can be defined as follows:

$$\tau_t(a, b) = \frac{1 - \sum_{i,j:\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{\tau}} p_t(r_i^a[i]) p_t(r_i^b[j]) \tau_t(r_i^a[i], r_i^b[j])}{1 - \sum_{i,j:\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{\tau}} p_t(r_i^a[i]) p_t(r_i^b[j])} \tag{3}$$

where by writing ' $\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{\tau}$ ' we mean that the ratio $\tau_t(r_i^a[i], r_i^b[j])$ is chosen to be stored in the summarized representation of packrow t . Let us refer to Fig. 4, which shows how the ratios are calculated (Fig. 4).

The ratio (3) minimizes a weighted error comprising quantities of the form $|\tau_t(a, b) - \frac{p_t(r_i^a[i], r_i^b[j])}{p_t(r_i^a[i]) p_t(r_i^b[j])}|$, for pairs of ranges whose co-occurrence ratios are not stored. Formulation of $\tau_t(a, b)$ enabled us to redesign and adapt some of classical probability estimation and revision methods (which would have – in their original versions – quadratic complexity

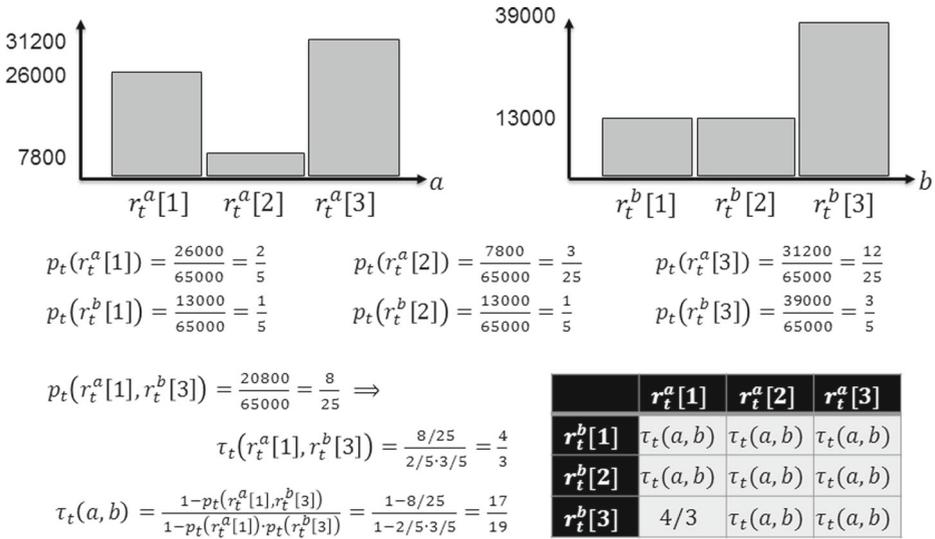


Fig. 4 Example of quantized probabilities in a packrow t embracing 65000 original rows, for columns a and b , each described using three histogram bars. Under the assumption that the engine stores co-occurrence information about only one pair of a 's and b 's ranges – $\tau_t(r_t^a[1], r_t^b[3]) = 4/3$ – the default ratio for other pairs equals to $\tau_t(a, b) = 17/19$

with respect to the number of bars / values), so they work linearly for the knowledge transformation purposes (see Section 4). In Chądzyńska-Krasowska et al. (2017), we reported also some interesting mathematical properties of $\tau_t(a, b)$ with regards to machine learning methods.

An analogous approach can be introduced for pairs of special values. Let us denote by $s_t^a[k]$ and $s_t^b[l]$ the k -th and l -th special values for columns a and b within packrow t , respectively. Let us denote data-derived probabilities of their occurrence and co-occurrence as $p_t(s_t^a[k])$, $p_t(s_t^b[l])$ and $p_t(s_t^a[k], s_t^b[l])$. The ranking used in the knowledge capture layer compares co-occurrence ratios of special values to the ratios for their “parents”, i.e., histogram ranges that $s_t^a[k]$ and $s_t^b[l]$ drop into – denoted by $s_t^a[k]^\uparrow$ and $s_t^b[l]^\uparrow$ – respectively:

$$\text{specs}_t(s_t^a[k], s_t^b[l]) = p_t(s_t^a[k]) p_t(s_t^b[l]) \left| \frac{p_t(s_t^a[k], s_t^b[l])}{p_t(s_t^a[k]) p_t(s_t^b[l])} - \frac{p_t(s_t^a[k]^\uparrow, s_t^b[l]^\uparrow)}{p_t(s_t^a[k]^\uparrow) p_t(s_t^b[l]^\uparrow)} \right| \quad (4)$$

Default co-occurrence ratios analogous to formula (3) are stored for special values as well.² They are utilized by the engine during the knowledge transformation processes whenever there is a need to refer to pairs of special values that were not chosen to be stored. For chosen pairs of special values, we store their corresponding ratios $\tau_t(s_t^a[k], s_t^b[l])$ defined similarly to (2).

²In Fig. 5, one can find two types of default ratios – *default_special_special_tau* and *default_not_covered_special_special_tau* – corresponding to the cases of “parents” of special values whose ratios are stored and are not stored, respectively.

3.3 Analytical testing environment

Our engine stores data summaries in binary files accessible by approximate query execution methods via internal interfaces. From a logical viewpoint, the contents of those files can be represented as a collection of data summary tables displayed in Fig. 5. For diagnostic purposes, we implemented a converter that transforms the contents of the binary files into an explicit relational form. As a result, one can use standard PostgreSQL to access and analyze the data summary tables that store information about histogram frequencies and ranges, special values, gaps, as well as bar-to-bar and value-to-value co-occurrences, per each database, table, column and packrow, independently from the implemented knowledge transformation mechanisms.

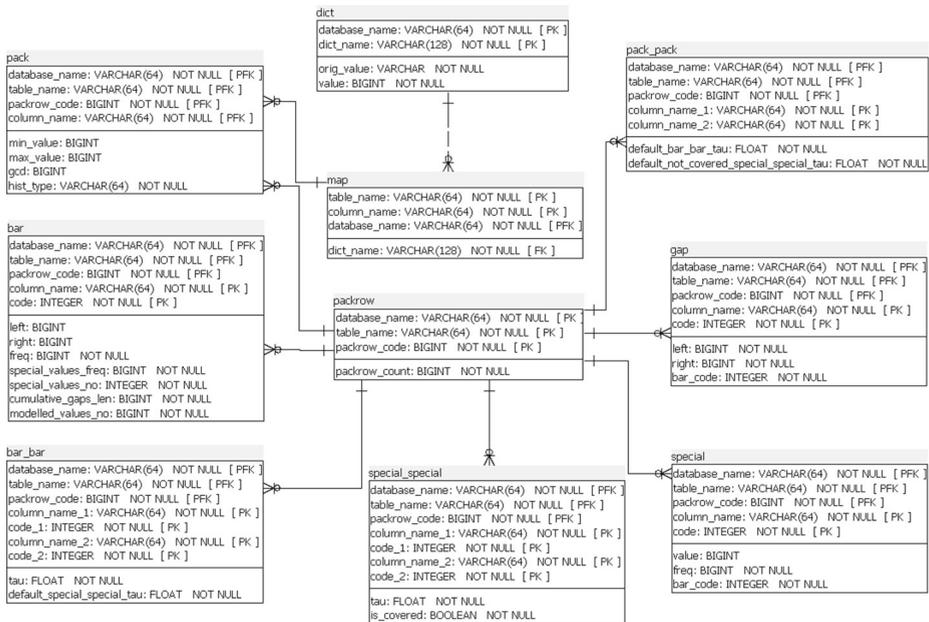


Table Name	Description of Table Contents
<i>packrow</i>	Number of rows gathered in particular packrows (2 ¹⁶ by default)
<i>pack</i>	Basic information about values occurring in particular data packs
<i>bar</i>	Information about particular bars – their supports, ranges, etc.
<i>special</i>	Information about special values, including bars they belong to
<i>gap</i>	Information about gaps – their ranges and bars they belong to
<i>pack_pack</i>	Compressed information about co-occurrence ratios that are not represented in tables <i>bar_bar</i> and <i>special_special</i> (Subsection 3.2)
<i>bar_bar</i>	Information about co-occurrence ratios for the most interesting pairs of bars and compressed information about co-occurrence ratios of pairs of special values belonging to the considered pairs of bars that are not represented in table <i>special_special</i>
<i>special_special</i>	Co-occurrence ratios for the most interesting special value pairs
<i>dict</i>	Dictionary codes assigned to values of low-cardinality columns
<i>map</i>	The mapping between the contents of table <i>pack</i> and table <i>dict</i>

Fig. 5 Data summary tables available for diagnostic and analytical purposes

This kind of an alternative access to the outcomes of our knowledge capture layer proved helpful when developing our quality testing environment. Moreover, the schema presented in Fig. 5 is a great means for elucidation to our customers. With a use of simple SQL, one can check which pairs of columns are most correlated by means of co-occurrences of their histogram ranges and special values (by querying *bar_bar* and *special_special*, respectively), what is the degree of repeatability of special values in different data packs of the same data column (by querying *special*), whether values of particular data columns evolve from packrow to packrow (by querying *pack* and *gap*), etc.

This kind of extra knowledge can be helpful for the users to better understand quality and performance characteristics of our query execution processes. Moreover, they may wish to explore data summary tables directly to do basic analytics, e.g., approximately visualize demographics of particular columns. We also work with data summaries in their relational form visible in Fig. 5 while prototyping new algorithms. For instance, in Chądzyńska-Krasowska et al. (2017) we reported some examples of feature selection methods performed entirely on our data summary tables. The obtained empirical results prove that our one-dimensional and two-dimensional structures described in the previous subsections can provide completely new means to scale machine learning algorithms (Peng et al. 2005).

4 Knowledge transformation operations

For each SELECT statement, the engine *transforms* available data summaries iteratively to build a summary of the query result. Different transformation mechanisms are dedicated to different operations such as filtering, joining, aggregating, etc. Each subsequent operation in a query execution chain takes as its input relevant summaries produced by previous operations. Once the summaries corresponding to the final stage of query execution are assembled, we use them to produce the outcome interpretable as a standard SQL result. This idea becomes particularly interesting when compared to approximate query techniques based on data sampling, where – after all – the whole computational process remains at the original level of atomic data.

4.1 Filtering-related transformations

Let us consider the operation of filtering, i.e., taking into account SQL clauses such as WHERE (in the case of summaries of the original tables or dynamically derived summaries of the nested SELECT results) or HAVING (in the case of dynamically derived summaries of earlier GROUP BY operations). For the engine introduced in this paper, this means revising frequencies for histograms, special values and co-occurrence ratios for columns relevant for further query execution stages, basing on filters specified over other columns.

The filtering process adapts the tree-based belief propagation (Neapolitan 2003), which is a popular tool in decision making and modeling (Weiss and Pearl 2010). Over the years, many improvements and extensions of the original belief propagation algorithms were proposed, including their SQL-based implementations (Gatterbauer et al. 2015). Nevertheless, this paper presents the first production-ready attempt to embed this idea *inside* an approximate query engine, where – in a loop over packrows – the most interesting co-occurrences of bars and special values of different columns are used as partial knowledge about data-derived conditional probability distributions. For more examples of probabilistic

information flow models that operate with partial knowledge about value co-occurrences, let us refer to Pawlak (2005).

Let us consider a simple query of the form `SELECT SUM(a) FROM T WHERE b > x`, which involves columns *a* and *b* in data table *T*. From the perspective of our knowledge transformation layer, the process of executing this query is split into two separate phases: 1) the filtering transformation: calculating new counts and frequencies of *a*'s histogram bars (and special values) subject to the condition *b > x* and 2) the result generation: assembling `SUM(a)` based on the outcome of filtering, in combination with information about values occurring inside particular ranges of *a*. Herein, we concentrate on the first of the above phases. The second one will be discussed in Section 5.

Figure 6 illustrates how *b > x* influences *b*'s and *a*'s representation. For a simpler query, i.e., `SELECT COUNT(*) FROM T WHERE b > x`, the whole procedure would be finished by revising the heights of *b*'s bars – contribution of an exemplary packrow *t* into estimation of the amount of the original rows satisfying *b > x* would be then roughly 43333. However, to estimate `SUM(a)`, we need to propagate information about *b > x* onto *a*. Namely, we need to replace probabilities $p_t(r_t^a[1])$, $p_t(r_t^a[2])$ and $p_t(r_t^a[3])$ that reflect the input data with revised probabilities $p'_t(r_t^a[1])$, $p'_t(r_t^a[2])$ and $p'_t(r_t^a[3])$ that reflect the filtered data. Contribution of packrow *t* to overall calculation of `SUM(a)` can be then derived using estimation of the number of its corresponding rows holding *b > x* and the revised probabilities of *a*'s bars.

Equation $p'_t(r_t^a[1]) = p_t(r_t^a[1]|r_t^b[1])p'_t(r_t^b[1]) + \dots + p_t(r_t^a[1]|r_t^b[3])p'_t(r_t^b[3])$ in Fig. 6 illustrates a general idea of belief propagation. The revised probability $p'_t(r_t^a[1])$ is calculated using classical total probability that combines previously-revised marginal

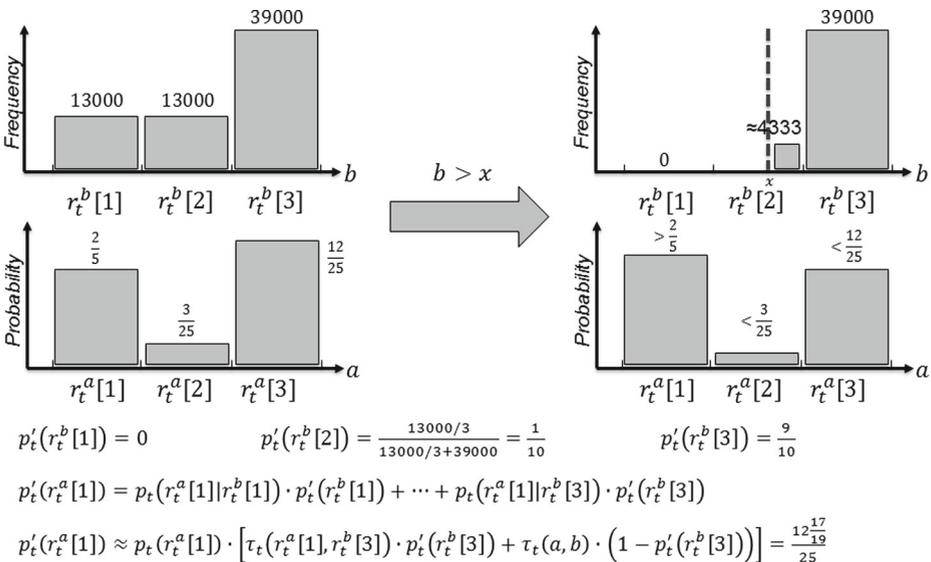


Fig. 6 Revision of probabilities over columns *a* and *b* subject to *b > x*. For packrow displayed in Fig. 4, it is estimated (based on *b*'s domain representation) that *b > x* holds for 33% of the original rows with *b*'s values dropping into the second range of *b*'s histogram. After recalculation of partial counts, the revised probability $p'_t(r_t^b[3])$ of *b*'s third range becomes higher than the original $p_t(r_t^b[3])$. As this range is known to have positive co-occurrence with the first range on *a*, the revised $p'_t(r_t^a[1])$ is higher than $p_t(r_t^a[1])$

probabilities over b with fixed conditional probabilities of a subject to b . In our approach, conditionals $p_t(r_i^a[i]|r_i^b[j])$ can be retrieved as $\tau_t(r_i^a[i], r_i^b[j]) p_t(r_i^a[i])$ or, for not stored co-occurrences, approximated by $\tau_t(a, b) p_t(r_i^a[i])$. This leads to the following:

$$p'_t(r_i^a[i]) / p_t(r_i^a[i]) = \sum_{j:\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{t}} [p'_t(r_i^b[j]) \tau_t(r_i^a[i], r_i^b[j])] + [1 - \sum_{j:\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{t}} p'_t(r_i^b[j])] \tau_t(a, b) \tag{5}$$

The above proportion is a foundation for the WHERE-related knowledge transformations implemented in the engine. In further subsections, we show how to extend it towards multi-column scenarios. As for now, let us just note that it can be easily generalized onto the case of operating with both histogram bars and special values. Moreover, let us emphasize that the mechanism of revising the whole vector of probabilities over a using (5) can be implemented in such a way that its computational cost remains linear with respect to the explicitly stored co-occurrences. As briefly outlined in Section 3.2, this is a great performance advantage when compared to a non-optimized quadratic style of working with conditional probability distributions.

4.2 Dynamic construction of propagation trees

Let us now explain how the idea of belief propagation can be adapted to serve for the WHERE-related knowledge transformations. The most basic propagation algorithm operates on a directed tree spanned over variables assigned with finite domains of values. A directed edge in the tree expresses causal relation, whereby a parent makes its child independent from its remaining non-descendants. Such relation is encoded in a form of conditional probability distribution of the child node subject to its parent. For each packrow t , our task is to construct and use a tree spanned over columns involved in a given query, with probabilities defined by means of available histograms and co-occurrence ratios. In the general case, distributions are defined over special values and histogram ranges embracing the remaining “non-special” values. For the sake of clarity, we limit ourselves to a simplified scenario where one-dimensional column representations correspond only to histogram ranges.

Let us refer again to the example in Fig. 6. In that case, the corresponding tree can be envisioned as $a \leftarrow b$, that is a is the child of b . Single-variable probability distributions are defined over ranges $r_i^a[1], r_i^a[2]$ and $r_i^a[3]$, as well as $r_i^b[1], r_i^b[2]$ and $r_i^b[3]$. (Actually, one can see that this is one of differences when compared to classical model of belief propagation, where the single-variable distribution is stored only for the root of a tree.) Furthermore, conditional distribution $p_t(r_i^a[i]|r_i^b[j])$ can be retrieved as either $\tau_t(r_i^a[i], r_i^b[j]) p_t(r_i^a[i])$ in the case of $\tau_t(r_i^a[i], r_i^b[j]) \in \tilde{t}$, or $\tau_t(a, b) p_t(r_i^a[i])$ otherwise.

As another example, let us consider the statement `SELECT b, SUM(a) FROM T GROUP BY b WHERE b > x`. In this case, the propagation tree $a \leftarrow b$ looks like before. However, in addition to single-column distributions, we need to pass further also revised knowledge about co-occurrences involving a 's and b 's histogram ranges. This is because of the specifics of the `GROUP BY` operation, which – as briefly outlined at the beginning of Section 7 – transforms summaries reflecting the filtered data into summaries representing packrows of tuples indexed by the grouping values and their corresponding aggregation coefficients. Thus, if appropriately recalculated ratios between a and b are not provided, then the whole mechanism would produce incorrect final summaries. We go back to this topic in Section 4.3, where it is discussed how to use propagation trees to revise knowledge about co-occurrence ratios.

Algorithm 2 WHERE-related belief propagation

Input: \tilde{t} : summarized representation of packrow t of table T ;
 B : columns in T that will be required at further stages of query Q ;
 C : columns in T that correspond to conjunction of conditions in Q ;
Output: \tilde{t}' : summarized representation of the filtered content of packrow t ;

- 1: **begin**
- 2: initiate undirected graph G_t with nodes corresponding to columns in $B \cup C$;
- 3: connect each pair of columns represented by at least one co-occurrence ratio in \tilde{t} ;
- 4: set $p_t(Q) \leftarrow 1$ – estimated ratio of rows holding Q 's filters on C ;
- 5: **for all** G_t 's connected components denoted by G_t^X , defined over $X \subseteq B \cup C$ **do**
- 6: construct spanning tree S_t^X maximizing the sum of values of I_t (equation (6));
- 7: find $a \in X$ with the minimum average distance to nodes $c \in C \cap X$ in S_t^X ;
- 8: use a as the root of directed tree \vec{S}_t^X created at the basis of S_t^X ;
- 9: **for all** $c \in C \cap X$ **do**
- 10: create variable c_Q with two single-valued ranges $r_Q^c[1]$ and $r_Q^c[2]$;
- 11: extend tree \vec{S}_t^X by adding node c_Q and making it a child of c ;
- 12: use equation (7) to set probabilities $p_t(r_Q^c[1]|r_Q^c[i])$ of c_Q subject to c ;
- 13: **end for**
- 14: use equations (10-11) to run λ -downward \vec{S}_t^X -based belief propagation;
- 15: use equation (14) to derive $p_t(Q^{\downarrow X})$ – ratio of rows holding filters on $C \cap X$;
- 16: $p_t(Q) \leftarrow p_t(Q) \cdot p_t(Q^{\downarrow X})$;
- 17: **end for**
- 18: **if** $p_t(Q) > 0$ **then**
- 19: **for all** directed trees \vec{S}_t^X **do**
- 20: use equations (12-13) to run π -upward \vec{S}_t^X -based belief propagation;
- 21: use proportion (9) to add to \tilde{t}' the revised summaries for columns in $B \cap X$;
- 22: **for all** pairs of columns $a, b \in B \cap X$ which were connected in G_t **do**
- 23: **if** a and b are not directly connected in \vec{S}_t^X **then**
- 24: **if** \vec{S}_t^X contains directed path from a to b or from b to a **then**
- 25: temporarily replace b 's parent with a or a 's parent with b ;
- 26: **else**
- 27: do one of the above, replacing temporarily an edge with lower I_t ;
- 28: **end if**
- 29: **end if**
- 30: use equation (15) to add to \tilde{t}' the revised co-occurrence ratios for a and b ;
- 31: **end for**
- 32: **end for**
- 33: **else**
- 34: flash \tilde{t}' and ignore packrow t during further execution of Q ;
- 35: **end if**
- 36: **end**

As yet another example, let us consider the query in Fig. 7, where it is required to revise data summaries with respect to conjunction of two conditions – $b > x$ and $c < y$. In this case, in order to follow the idea of tree-based belief propagation, we need first to construct a tree over nodes a, b and c . Initially, we actually used to derive such trees during the phase of knowledge capture. The tree structures could be quite different for different packrows but, on the other hand, we used to assume that – for a given packrow t – it would be reasonable to have a single fixed tree ready for usage in the case of arbitrary queries. That approach turned out to be not flexible enough, as different queries – involving potentially different subsets of columns – may require different trees somewhat optimized with regards to their needs. To better address this issue, we developed a query-adaptive method of dynamic tree derivation.

Algorithm 2 addresses the task of tree construction in lines 2-3 and 6. The first stage is to connect pairs of columns, for which there are corresponding co-occurrence ratios stored in the summary of packrow t . This process is limited to the set of columns $B \cup C$, where B gathers columns that will be required at further query stages and C denotes columns with query conditions specified. The idea of not connecting pairs of columns with no co-occurrence representation is based on the assumption that such pairs are relatively weakly correlated with each other. Indeed, according to function $bars_t$ defined by (1), such complete lack of two-dimensional information means that the given two columns – when treating them as quantized random variables with data-derived probabilities p_t – seem to be *approximately* independent from each other.

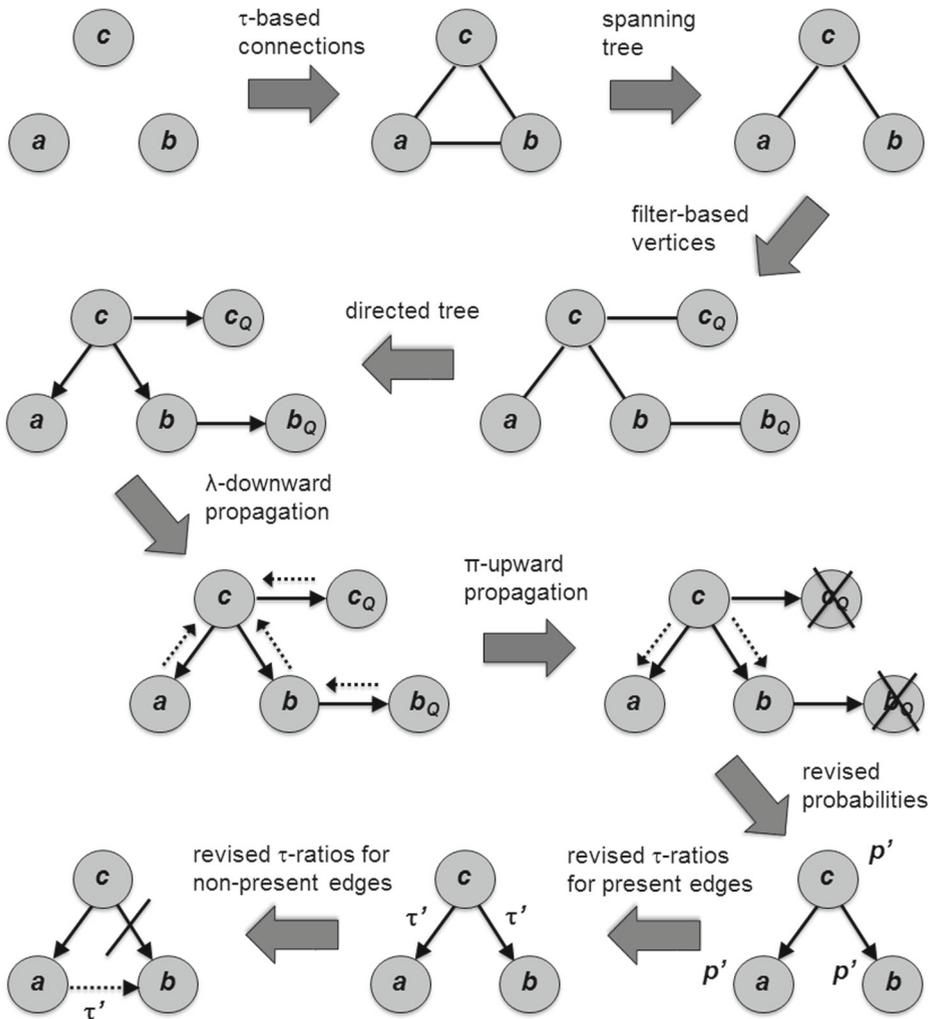


Fig. 7 Construction and usage of a directed tree for the WHERE-related belief propagation corresponding to the query `SELECT a, b, c FROM T WHERE b > x AND c < y`. For packrow t , it is assumed that there are some cases of co-occurrence ratios stored for each of pairs of the considered columns a , b and c . The approximated mutual information measure linking a and b is assumed to be weaker than in the cases of a and c , as well as b and c

If the undirected graph G_t constructed as a result of lines 2-3 has multiple connected components – further denoted by G_t^X for some column subsets $X \subseteq B \cup C$ – then, as above, we can assume that those subsets are approximate independent from each other. This means that further calculations can be conducted for each of such components separately and then merged together. Let us denote by $p_t(Q^{\downarrow X})$ the estimated ratio of rows in packrow t that satisfy filters specified in query Q on columns in $C \cap X$ (line 16). Then, the estimated ratio of rows in packrow t that satisfy all conditions of Q – denoted as $p_t(Q)$ – can be calculated as the product of coefficients $p_t(Q^{\downarrow X})$ (line 17). Similarly, we can run belief-propagation-based mechanisms aimed at revision of histograms and co-occurrence ratios over particular components G_t^X and then readjust them to obtain final summaries representing the filtered data.

In order to perform calculations with respect to a given $X \subseteq B \cup C$, we need to construct a tree based on G_t^X . This is initiated in line 6 of Algorithm 2. We employ the measure of mutual information (Peng et al. 2005) computed for pairs of columns to span an undirected tree S_t^X over G_t^X . Precisely, we use a summary-based approximation of that measure, as it was investigated in Chądzyńska-Krasowska et al. (2017):

$$I_t(a, b) = \sum_{i, j: \tau_i(r_i^a[i], r_i^b[j]) \in \tilde{\tau}} p_t(r_i^a[i]) p_t(r_i^b[j]) \tau_i(r_i^a[i], r_i^b[j]) \log \tau_i(r_i^a[i], r_i^b[j]) + \left[1 - \sum_{i, j: \tau_i(r_i^a[i], r_i^b[j]) \in \tilde{\tau}} p_t(r_i^a[i]) p_t(r_i^b[j]) \tau_i(r_i^a[i], r_i^b[j]) \right] \log \tau_i(a, b) \tag{6}$$

The usage of mutual information is justified by relationships between the data-derived information entropy of a graphical model and the *accuracy* of conditional independence assumptions that it represents (Ślęzak 2009). The usage of spanning trees that maximize mutual information is actually related to the foundations of second-order product approximations of joint probability distributions, which were deeply investigated in the literature (Meila 1999).

In Section 4.3, we describe how to calculate the revised probabilities and ratios within each $X \subseteq B \cup C$. To do this, we need to transform an undirected tree S_t^X into its directed version \vec{S}_t^X . Algorithm 2 addresses this task in lines 7-8. In order to minimize the amount of further computations, we choose as the tree’s root a column that is on average *closest* to the elements of $C \cap X$. Later, during the belief propagation process, it is also worth considering line 18 that checks whether complete calculations are really necessary. Namely, if the first phase of propagation provides us with $p_t(Q^{\downarrow X}) = 0$ over any of connected components of G_t , then packrow t can be skipped as irrelevant.

Surely, we realize that the whole introduced approach raises a number of questions with regards to the accuracy of estimations represented by transformed data summaries. First, one needs to take into account quantization of original column domains as discussed in Section 3.1. Second, as elaborated in Section 3.2, we can afford storing only a fraction of co-occurrence ratios representing relationships between quantized / discretized domains of different columns, which leads, e.g., to potentially inaccurate judgements with respect to data-derived probabilistic independencies. Third, in order to adapt the tree-based belief propagation mechanisms to model the process of filtering, we partially neglect some of two-column relationships that are evaluated as weaker than the others. Still, despite all those potential inaccuracy factors, the approximate results of analytical queries that we tested on large real-world data sets are usually *close enough* to their exact counterparts. Thanks to the diagnostic analysis performed on data summary tables described in Section 3.3, we can conclude that this is because our histogram generation, as well as special value / domain gap / co-occurrence ratio ranking functions are indeed able to store the most meaningful aspects of the original data.

4.3 Adaptation of belief propagation algorithm

In this subsection, we focus on calculations over a single tree \vec{S}_t^X , for a column subset $X \subseteq B \cup C$ corresponding to one of connected components of graph G_t . As before, we refer to Algorithm 2. The discussed procedures are also visualized by Fig. 6 that actually concerns the case of a connected graph, i.e., $X = B \cup C$, where $B = \{a, b, c\}$ and $C = \{b, c\}$.

The idea of adapting tree-based belief propagation is motivated by a need to develop an efficient method to model conjunctions of conditions specified over multiple columns. In this paper, we do not consider other logical operations (such as, e.g., disjunctions). We do not consider conditions defined over derived columns either (such as, e.g., arithmetic or CASE WHEN expressions – see Section 7 for further discussion). Such enhancements are quite straightforward to develop in the near future. Moreover, the current engine already contains some significant extensions when compared to the classical characteristics of propagation mechanisms (Weiss and Pearl 2010). In particular, it enables us to work with conjunctions of more complex single-column conditions than the “variable = value” filters that would be supported by standard model (Neapolitan 2003).

Let us denote by c_Q a two-valued variable characterizing whether a given original row in packrow t could hold the condition specified by query Q over column $c \in C \cap X$. Let us represent the values of c_Q as $r_Q^c[1]$ and $r_Q^c[2]$ corresponding to rows satisfying and not satisfying the considered condition, respectively. (Note that we do not index those values with t , as they have the same meaning for all packrows of a given table.) Using such new notation, we can rewrite probabilities discussed in Section 4.2 as $p_t(Q^{\downarrow X}) = p_t(\bigwedge_{c \in C \cap X} r_Q^c[1])$ and $p_t(Q) = p_t(\bigwedge_{c \in C} r_Q^c[1])$. Moreover, the main task of this subsection can be now formulated as estimation of new frequencies and co-occurrence ratios for (pairs of) columns in B as conditionals subject to $\bigwedge_{c \in C} r_Q^c[1]$.

As shown in Fig. 7 (see also line 11 in Algorithm 2), we extend tree \vec{S}_t^X with nodes corresponding to variables c_Q and specify their parents as the corresponding columns $c \in C \cap X$. Such new causal relations are equipped with conditional probabilities of the following form (line 12):

$$p_t(r_Q^c[1]|r_i^c[i]) = \frac{\|r_i^c[i] \cap Q\text{'s condition on } c\|_t}{\|r_i^c[i]\|_t} \tag{7}$$

where the nominator estimates the number of rows in packrow t whose values on c drop into its i -th range and in the same time satisfy the considered WHERE clause over c , while the denominator simply denotes the height of the i -th bar for c . Let us refer to Fig. 6, where – for b_Q representing filter $b > x$ – we obtain $p_t(r_Q^b[1]|r_i^b[1]) = 0$, $p_t(r_Q^b[1]|r_i^b[2]) = 1/3$ and $p_t(r_Q^b[1]|r_i^b[3]) = 1$. For this particular example, Algorithm 2 would proceed with the tree $a \leftarrow b \rightarrow b_Q$. Let us also note that computation of $p_t(r_Q^b[1]|r_i^b[2])$ would be the only place referring to more detailed knowledge about the domain of b . Namely, the analysis of gaps would influence estimation of the proportion of values observed in $r_i^b[2]$ that are greater than x . At all the remaining stages, Algorithm 2 would operate entirely with histograms and co-occurrence ratios.

After extensions described in lines 11-12, the given tree is ready to perform belief propagation, where query conditions are modeled by setting appropriate coefficients at the leaves corresponding to variables c_Q , $c \in C \cap X$. For $a \in X$ and tree \vec{S}_t^X , let us denote by D_t^a and N_t^a the sets of all descendants and non-descendants of a (including a itself), respectively. Let:

- $p_t^\downarrow(r_t^a[i])$ be the probability that conjunction $\bigwedge_{c \in C \cap D_t^a} r_Q^c[1]$ holds in packrow t subject to observing values within range $r_t^a[i]$ on column a ,
- $p_t^\uparrow(r_t^a[i])$ be the probability that values within range $r_t^a[i]$ are observed on column a subject to satisfaction of conjunction $\bigwedge_{c \in C \cap N_t^a} r_Q^c[1]$.

Given the independence of N_t^a and D_t^a subject to a in the considered tree-based probability model, we shall assume the following proportion:³

$$p'_t(r_t^a[i]) \propto p_t^\downarrow(r_t^a[i]) p_t^\uparrow(r_t^a[i]) \tag{8}$$

The left hand side of (8) represents the revised probability distribution on the quantized domain of a , so it sums up to 1 for all considered indexes i . This means that it is sufficient to find any value proportional to the above right hand side. The major advantage of tree-based belief propagation is its ability to recursively determine parameters $\lambda_t(r_t^a[i]) \propto p_t^\downarrow(r_t^a[i])$ and $\pi_t(r_t^a[i]) \propto p_t^\uparrow(r_t^a[i])$, so as to replace (8) with the following one:

$$p'_t(r_t^a[i]) \propto \lambda_t(r_t^a[i]) \pi_t(r_t^a[i]) \tag{9}$$

Let us denote by $Ch_t^a \subseteq D_t^a$ the set of all children of column a in \vec{S}_t^X . The standard way of calculating parameters λ_t and π_t is as follows:

- for each node c_Q , we put $\lambda_t(r_Q^c[1]) = 1, \lambda_t(r_Q^c[2]) = 0$,
- for all other leaves in \vec{S}_t^X , we put $\lambda_t \equiv 1$,
- for each $b \in X$ which is not a leaf in \vec{S}_t^X , we put:

$$\lambda_t(r_t^b[j]) = \prod_{a \in Ch_t^b} \lambda_t^a(r_t^b[j]) \tag{10}$$

where coefficients $\lambda_t^a(r_t^b[j])$ should be calculated as $\sum_i p_t(r_t^a[i]|r_t^b[j]) \lambda_t(r_t^a[i])$. However, as the engine stores only partial knowledge about data-derived probabilities, the classical computation of $\lambda_t^a(r_t^b[j])$ is replaced by:

$$\lambda_t^a(r_t^b[j]) = \sum_{i: \tau_t(r_t^a[i], r_t^b[j]) \in \bar{t}} \tau_t(r_t^a[i], r_t^b[j]) p_t(r_t^a[i]) \lambda_t(r_t^a[i]) + \tau_t(a, b) \left[\alpha_t^a - \sum_{i: \tau_t(r_t^a[i], r_t^b[j]) \in \bar{t}} p_t(r_t^a[i]) \lambda_t(r_t^a[i]) \right] \tag{11}$$

where $\alpha_t^a = \sum_i p_t(r_t^a[i]) \lambda_t(r_t^a[i])$. (10–11) are the basis for the λ -downward phase of the developed WHERE-related belief propagation. (12–13) considered below enable us to run the π -upward phase. Let us note that – when compared to the general layout – we do not need to initiate π_t for nodes c_Q , as those are leaves in the extended version of \vec{S}_t^X .

First, for the root of \vec{S}_t^X , we put $\pi_t(r_t^{root}[i]) = p_t(r_t^{root}[i])$. Then, for each $a \in X$ which is not the root, one would normally specify $\pi_t(r_t^a[i])$ as equal to $\sum_j p_t(r_t^a[i]|r_t^{\hat{a}}[j]) \pi_t^a(r_t^{\hat{a}}[j])$, where \hat{a} denotes the parent of a and:

$$\pi_t^a(r_t^{\hat{a}}[j]) = \pi_t(r_t^{\hat{a}}[j]) \prod_{b \in Ch_t^{\hat{a}} \setminus \{a\}} \lambda_t^b(r_t^{\hat{a}}[j]) \tag{12}$$

³Let us recall that we deal with a dynamically derived *approximate* probability model, with all its consequences discussed at the end of Section 4.2. The investigation how accurately such models reflect the data is one of our main analytical testing scopes.

However, for the same reason as above, we change the classical way of deriving $\pi_t(r_t^a[i])$ based on coefficients $\pi_t^a(r_t^{\hat{a}}[j])$ to the following one:

$$\pi_t(r_t^a[i]) = p_t(r_t^a[i]) \left[\sum_{j:\tau_t(r_t^a[i], r_t^{\hat{a}}[j]) \in \tilde{t}} \tau_t(r_t^a[i], r_t^{\hat{a}}[j]) \pi_t^a(r_t^{\hat{a}}[j]) + \tau_t(a, \hat{a}) \left[\beta_t^a - \sum_{j:\tau_t(r_t^a[i], r_t^{\hat{a}}[j]) \in \tilde{t}} \pi_t^a(r_t^{\hat{a}}[j]) \right] \right] \tag{13}$$

where $\beta_t^a = \sum_j \pi_t^a(r_t^{\hat{a}}[j])$. Due to the specifics of our tree construction, there is a straightforward way to estimate the ratio of rows in packrow t that satisfy conditions of query Q defined on columns belonging to $C \cap X$:

$$p_t(Q^{\downarrow X}) = \sum_i p_t(r_t^{root}[i]) \lambda_t(r_t^{root}[i]) \tag{14}$$

We can retrieve $p_t(Q^{\downarrow X})$ right after the λ -downward phase of belief propagation and, in the case of multiple connected components of graph G_t , derive the final $p_t(Q)$ according to line 16 in Algorithm 2. This leads towards already-discussed potential performance benefits. Namely, if $p_t(Q) = 0$, we can skip further calculations for packrow t and jump directly to line 34.

Another aspect of performance acceleration relates to parameters $\lambda_t^a(r_t^b[j])$ and $\pi_t(r_t^a[i])$. Let $max_no_of_bars$ denote the maximum number of histogram bars, as in Algorithm 1. In their original form, formulas $\lambda_t^a(r_t^b[j]) = \sum_i p_t(r_t^a[i]|r_t^b[j]) \lambda_t(r_t^a[i])$ and $\pi_t(r_t^a[i]) = \sum_j p_t(r_t^a[i]|r_t^{\hat{a}}[j]) \pi_t^a(r_t^{\hat{a}}[j])$ would lead toward the quadratic cost $\mathcal{O}(|X| \cdot max_no_of_bars^2)$. On the contrary, (11) and (13) provide us with computational complexity at the level $\mathcal{O}(\max(|X| \cdot max_no_of_bars, |\tilde{t}|))$, where $|\tilde{t}|$ denotes the number of co-occurrence ratios stored by the considered engine for packrow t .

The last aspect of Algorithm 2 is to use λ/π -coefficients to deliver revised single-column distributions and co-occurrence ratios (denoted in general as \tilde{t}') as an input to subsequent query execution steps or the final phase of query result generation (discussed in Section 5). With regards to histograms, one can refer to proportion (9). As about co-occurrence ratios involving connected columns, it is enough to note that – for a given a and its parent \hat{a} – the revised probability $p_t'(r_t^a[i]|r_t^{\hat{a}}[j])$ could be resolved by the belief propagation process performed with Q 's conditions extended by $\lambda_t(r_t^{\hat{a}}[j]) \leftarrow 1$ (and 0 otherwise). Simple recalculations lead then towards the following formula for the revised co-occurrence ratio derived as $p_t'(r_t^a[i]|r_t^{\hat{a}}[j])$ divided by $p_t'(r_t^a[i])$:

$$\tau_t'(r_t^a[i], r_t^{\hat{a}}[j]) = \frac{\lambda_t(r_t^a[i]) \tau_t(r_t^a[i], r_t^{\hat{a}}[j]) p_t(r_t^a[i])}{\lambda_t^a(r_t^{\hat{a}}[j]) p_t'(r_t^a[i])} \tag{15}$$

The above equation can be used to recalculate all co-occurrence ratios stored in \tilde{t} . For pairs of columns that are not connected, we can use the same formula for a slightly modified tree as described in lines 23–29 of Algorithm 2. This part of the process is also highlighted as the last step in Fig. 7. The idea is to temporarily replace one of the existing edges with the edge directly connecting the considered columns. The whole operation is performed in such a way that a loss of summary-based approximation of mutual information I_t – and therefore also a loss of the tree entropy – is minimized. The revised default ratios $\tau_t'(a, b)$ are then obtained by using the formula analogous to (3).

5 Generating final query results

Once the summary of a query output is calculated, the engine translates it into the standard SQL result format. Prior to this stage, as visible in Fig. 8, knowledge being transformed throughout subsequent query execution stages is highly condensed and therefore it requires only a fraction of resources of a traditional database engine to produce the results. However, at the end, we need to deliver a result in an externally interpretable format. (Unless this is the case of a nested query, where the result of subquery can remain in a summarized form, like we did with rough queries in Kowalski et al. 2013.) This phase could be referred to as materialization, though it should not be confused with a standard meaning of materialization in columnar databases (Abadi et al. 2013).

Alternatively, if the knowledge capture layer is regarded as responsible for aforementioned information granulation (Zadeh 1997), then translation of query result summaries into final approximate results can be treated as *information degranulation*. This way, the overall design of our engine fits the idea of calculations on information granules, with a special emphasis on their transformation and reorganization. More inspirations – related in particular to switching between granular and detailed data representations – can be found in Liu et al. (2013).

Query result generation can also be understood as transition from a result summary to a result comprising crisply-valued tuples. For simple queries, such as aggregations optionally grouped with respect to low-cardinality data columns, this stage is quite straightforward. As an example, let us go back to the SELECT statement considered in Section 4.1. Its execution consists of transforming information about *a* subject to the clause $b > x$ and then producing the final outcome $SUM(a)$. In this case, the result can be computed as a total sum of outputs produced by each packrows. For a given packrow, we take into account the transformed frequencies of *a*'s special values and we add *value · frequency* scores to the constructed sum. The remaining part is to calculate contributions of histogram bars after subtracting frequencies of their special values. For this purpose, we construct an estimate of an average “non-special” value within each bar. Such estimate can be obtained based on domain representation discussed at the end of Section 3.1.

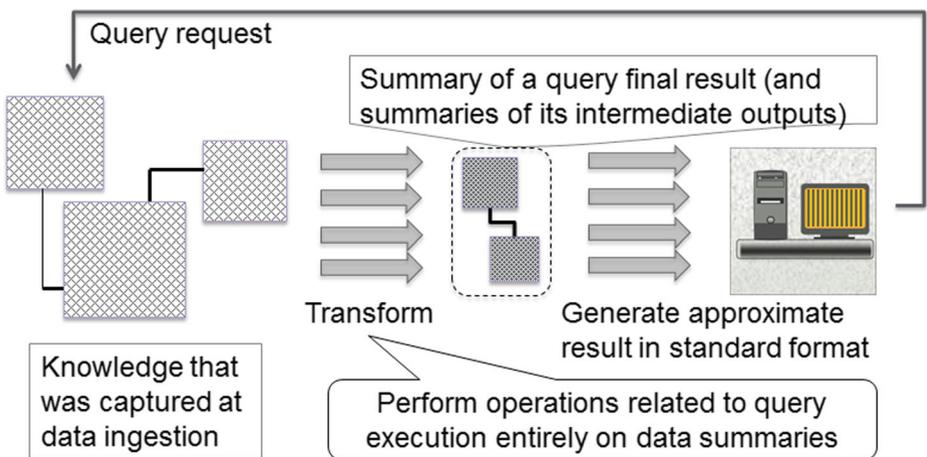


Fig. 8 Query execution process understood as a sequence of data summary transformations, with the query result generation as its very last step

The situation changes if high-cardinality columns are involved. Then, it is especially beneficial to model all data operations at the level of summaries and switch to the detailed values just prior to shipping the query result outside. Let us consider query `SELECT a, b, c FROM T WHERE b > x and c < y` highlighted in Fig. 7. In this case, after belief propagation, we need to generate collections of the resulting a - b - c tuples corresponding to each of (non-zeroed) packrow summaries. First, we use those summaries (both single- and multi-dimensional ones) to create collections of tuples labeled with codes of special values and histogram ranges (symbolizing occurrences of “non-special” values). At the end, we replace codes with final values generated using – again – domain information available for the involved columns.

In conclusion, query result generation relies both on operations presented in previous subsections and the domain representation used at the last stage. Let us recall that in addition to knowledge summarized in terms of (co-)occurrences of special values and histogram bars, we store the most significant gaps and the greatest common divisors of values observed in the original data packs. Referring to the theory of rough sets (Pawlak and Skowron 2007), we can say that special values whose frequencies were not pushed down to zero during query execution constitute a kind of domain’s positive region, i.e., these are values that *should contribute* to the query result. On the other hand, gaps, greatest common divisors, dictionaries (if available) and zeroed frequencies can be used together to define the domain’s negative region, i.e., values that *should not contribute* to the result. Utilization of such regions during result generation is one of the areas of our ongoing improvement, as it strongly relates to elimination of non-existing values in the approximate query outcomes (Chądzynska-Krasowska et al. 2017).

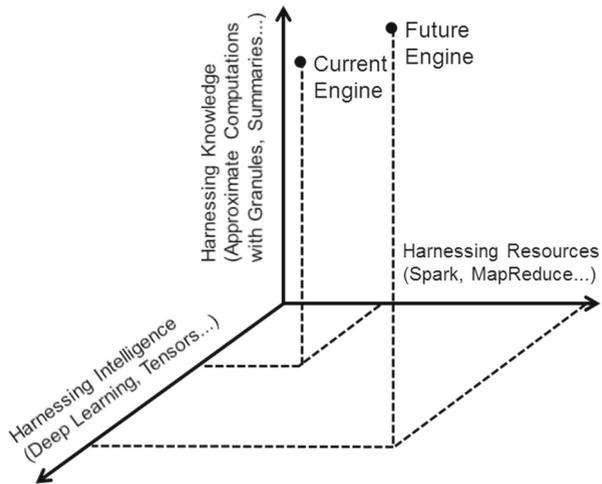
6 Empirical evaluation

Let us now discuss how we approach the topic of the engine evaluation. From the user perspective, the most important aspect is to look at the speed versus accuracy of deriving approximate results, in comparison to both exact and sample-based styles of calculations. From an architectural roadmap standpoint, we should also consider abilities of our approach to scale with respect to available computational resources and perform sufficiently well for increasingly complex operations. This is in line with Fig. 9 that shows our ultimate goal to combine summary-based computations with other paradigms.

Figure 10 illustrates a typical scenario of comparing our solution with the others. Surely, one can expect an increase in performance at the cost of losing accuracy. The point is to find out the right balance between these two aspects for particular case studies involving expected queries, data characteristics and data owner requirements (Glick 2015). The initial task is to assist the users in evaluating the accuracy. As mentioned in Section 2.2, several approximate query engines enrich results with confidence intervals that reflect the output’s credibility when comparing it to a hypothetical actual result (Mozafari and Niu 2015). However, first we need to establish the means for expressing similarities between approximate and explicitly computed standard results, so as to confirm the user expectations with respect to the speed-versus-accuracy trade-offs.

A reasonable measure of similarity between exact and approximate outcome of queries should certainly satisfy some basic mathematical properties. Moreover, it should be coherent with human *perception* of the meaning of both results in practice. In our framework, if

Fig. 9 High-level roadmap for combining summary-based calculations with modern trends related to functionality (see e.g. Hinton 2007) and scalability (see e.g. Zaharia et al. 2016) of data analytics



an outcome of query Q is supposed to take a form of a single numeric value, then – denoting by $out_Q / \widetilde{out}_Q$ the exact / approximate results, respectively – we use the following:

$$similarity(out_Q, \widetilde{out}_Q) = 1 - |out_Q - \widetilde{out}_Q| / (|out_Q| + |\widetilde{out}_Q|) \tag{16}$$

Queries with GROUP BY deliver tuples labeled by grouping columns and aggregate scores. The formula (16) is then averaged over all groups that occur in either exact or approximate outcomes. In the case of query in Fig. 10, the groups are induced by column tm_day . Fortunately, the same tm_day values occur for both exact and approximate results. Moreover, for each tm_day , its corresponding exact and approximate aggregates are very close to each other. This is why the overall similarity is close to the maximum. However, if an approximate result included a large amount of grouping values that should actually not occur at all, then its overall similarity to the exact result would be significantly lower. This is one more reason to pay special attention to the final query result generation, as it was discussed in Section 5.

Referring to Fig. 10 again, one might claim that comparing ourselves to Hive / Spark SQL is not fair because these are products with quite different aims. Indeed, such solutions are expected to cooperate rather than compete with each other, at the level of both external characteristics and the underlying computational paradigms. Actually, we have already conducted experiments related to re-implementation of our knowledge transformation operations in native Spark environment (Zaharia et al. 2016). Preliminary results are highly encouraging, as such implementation allows us to take advantage of query acceleration factors corresponding to both summary-based computations and resource-related scalability used, e.g., by Spark SQL (Armbrust et al. 2015). On the other hand, it is certainly important to empirically compare our approach with other approximate query processing solutions. However, as mentioned in Section 2.2, this is a difficult challenge because those solutions will still require far more development work to scale and provide sufficient accuracy for complex queries.

The engine should be also evaluated from the viewpoint of complex analytical pipelines. Figure 11 illustrates the idea of using query results within machine learning algorithms.

Hadoop Test Environment	Hive	Spark SQL	Ours
<ul style="list-style-type: none"> Original data size: 11 TB Our storage size: 99 GB Parquet data size: 2.4 TB (1.2 TB with replication) 	Used Nodes: 4	6	1
	RAM (GB) Total: 192	240	64
	Cores Total: 32	32	32
	Proc. Speed (MHz): 2400	2400	2400

Average Performance (18 queries):	our solution 3300x faster than Hive
	our solution 244x faster than Spark SQL

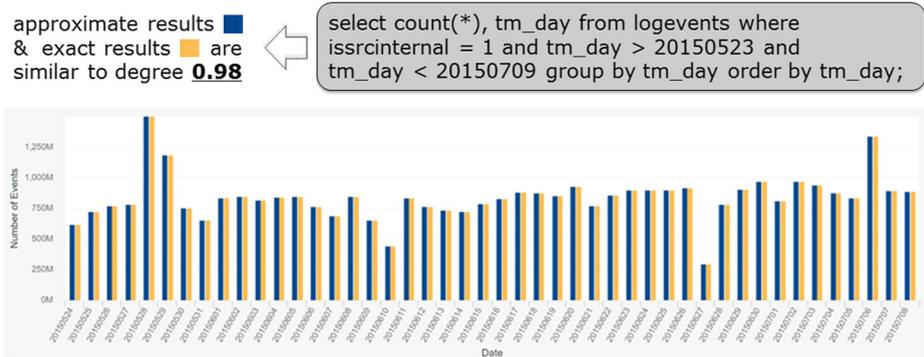


Fig. 10 Our solution versus standard settings of Hive (Thusoo et al. 2009) and Spark SQL (Armbrust et al. 2015). Queries were executed over a data set provided by one of our customers. In the case of Hive and Spark SQL, we used Hadoop / Parquet (<https://parquet.apache.org/>). For an exemplary query, the similarity between its exact and approximate results is equal to 0.98

Indeed, there are a number of scalable implementations of decision tree induction (Nguyen and Nguyen 2005) or data clustering (Ordonez and Cereghini 2000), that are based on iterative execution of ad-hoc SQL statements. This gives us an opportunity to evaluate approximate results not only by means of similarity measures but also by means of comparing the final outcomes of SQL-based machine learning algorithms while feeding them with approximate versus exact calculations. This seems to refer to the aforementioned analogy between approximate querying and perceptual image processing methods, whereby the main point is to focus on functionally important aspects of managed information.

As shown in Fig. 11, one can also run some machine learning methods directly against data summaries, instead of using classical SQL-based interface. Any steps toward this direction would fit the harnessing knowledge dimension illustrated in Fig. 9. (We will discuss it further in Section 7.) Moreover, from the perspective of evaluation of the engine components, it would let us evaluate the knowledge capture layer decoupled from SQL-specific knowledge transformation mechanisms. That was actually our motivation to develop the already-mentioned summary-based feature selection approach reported in Chądzyńska-Krasowska et al. (2017). Namely, we considered one of standard minimum redundancy maximum relevance (mRMR) feature selection techniques (Peng et al. 2005) and executed it for some real-world data sets using: 1) mutual information scores computed over the original data, 2) mutual information scores computed over 15%-samples and 3) approximations of mutual information scores derived directly from the stored data summaries. The outputs – interpreted as rankings of columns produced by each of the three above runs of the mRMR algorithm – turned out to be encouragingly similar to each other, while summary-based approximate calculations were incomparably faster than for both other cases.

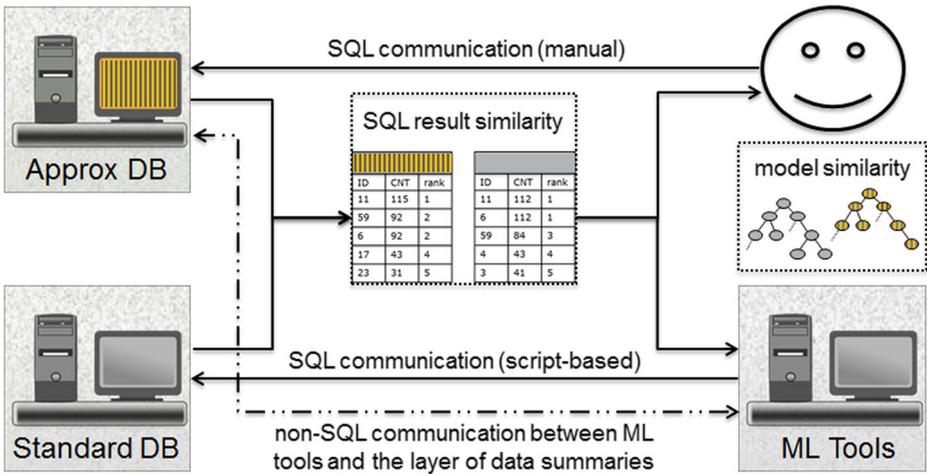


Fig. 11 Different modes of comparative evaluation of accuracy of approximate queries. Solid lines represent analytical processes based on SQL statements executed manually or automatically by, e.g., machine learning (ML) algorithms (Nguyen and Nguyen 2005). Dashed line represents a more direct style of implementation of ML methods over data summaries (Chądzyńska-Krasowska et al. 2017)

7 Further discussion and development directions

There are a number of important topics that were not addressed in this paper in a detailed way. We summarize them briefly below.

Coverage of SQL operations Besides data filtering, the engine supports also other operations, such as JOIN, GROUP BY, etc. While joining two tables, we need to create summaries of collections of tuples that would belong to a standard output of JOIN operation. This is done by *amalgamating* the pairs of summarized packrows of input tables. Such amalgamation is analogous to the filtering operations, though it can be preceded by a single-table process of merging *similar* packrow summaries (the outcomes of Algorithm 2, if any filters were applied prior to joining). As for GROUP BY, its standard output would consist of vectors labeled by aggregate results and the corresponding values of columns used for grouping. Herein, the goal is to produce summaries of such vectors without generating them explicitly. More examples of our knowledge transformation operations will be described in the near future.

Data types and derived columns In general, some components of above-mentioned operations are agnostic, while the others depend on semantics of particular columns. For example, in Algorithm 2, almost all calculations are conducted on probabilities of abstracted partition blocks, except line 12. This is worth remembering while developing new knowledge capture techniques. For instance, for some alphanumeric columns one could consider histogram bars labeled by prefixes instead of ranges. For such new types of histograms, some fundamental knowledge transformation functions – such as (7) – would require re-implementation. There is also requirement for dynamic creation of both one-dimensional and two-dimensional summaries reflecting new columns derived as complex expressions. Certainly, computations at this level need to rely on the types of original columns as well.

Disk and memory management Many database mechanisms can be adapted to work with granulated summaries. Their contents can be clustered into even bigger chunks – “mega-packs” – and labeled with higher-level descriptions. For the considered engine, it would be a step toward the paradigms of multi-level granular data analytics (Yao 2016). Yet another aspect refers to vertical data organization (Abadi et al. 2013), which – in our case – means an independent access to collections of histograms, special values, gaps, co-occurrence ratios, etc. This way, for every query, we can grasp these components of stored data summaries that are required to execute particular operations. Such components can be cached in memory for the purposes of future queries. Moreover, summaries representing intermediate outputs of query execution stages can be managed in memory in a pipeline style like in the case of standard database engines.

Knowledge capture improvements It is still possible to enhance our criteria for choosing ranges, special values, etc. Improvements can be also achieved by putting more intelligence into the process of assigning the original data rows to buffered packrows, based on our earlier experiences (Ślęzak et al. 2013). Moreover, we have started integrating our knowledge capture layer with some components of the Apache ecosystem (<https://kafka.apache.org/>). We also work on the synchronization of distributed knowledge capture processes with respect to maintenance of global structures, such as dictionaries for low-cardinality columns (see table *dict* in Fig. 5). Our goal is to build a modular architecture (see Fig. 12), where a layer of reconfigurable knowledge capture agents is the means for connecting to different types of data locations and preparing summaries available within different analytical platforms (Zaharia et al. 2016).

Accuracy trade-offs and hybrid scenarios In Section 3, we pointed out that our engine should store only a *limited* fraction of knowledge, specified by means of maximum amounts

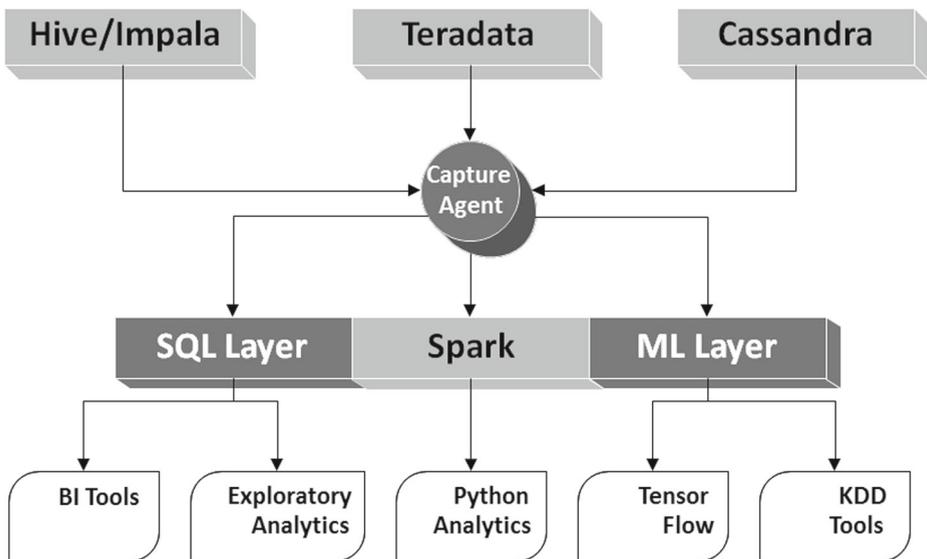


Fig. 12 Combining the components of our solution with other tools and systems

of histogram bars, special values, gaps and co-occurrence ratios. Although all those amounts are constrained by some defaults, it is possible to vary them with respect to particular columns or the whole data tables. For instance, it may be desirable to optionally access some of tables in a classical form and join them with summarized contents of other tables within hybrid query execution schemes. Such flexibility fits real-world scenarios of incremental data analytics (Fisher et al. 2012), but it requires careful maintenance of information that allows the algorithms to extract the original data from remote sources. From the perspective of Fig. 12, this kind of information needs to be managed by knowledge capture agents. Parameters enabling the engine to trace the original data should constitute a kind of analytical knowledge grid, by analogy to the concept of knowledge grid considered in semantic web (Cannataro and Talia 2003).

Accuracy measures and confidence intervals In Section 6, we referred to the accuracy as a measure of similarity between exact and approximate query results, as well as similarity between the outcomes of machine learning algorithms working with detailed and summarized data sets. One can conduct a deeper discussion on a need of designing the quality measures of data summaries and investigating relationships between the accuracies of input summaries and the expected accuracies of query results. Finally, special attention should be devoted to development of confidence interval calculations, which would assist the users in their everyday work, as is used with some sampling-based approximate query engines. To accomplish this, our plans are twofold. First, we intend to enhance our framework with a kind of *query explain* functionality enabling the users to trace potential inaccuracies emerging at particular stages of query execution. Second, we intend to create a tool, which maintains a sample of original packrows and uses it – in combination with the core engine – to produce confidence intervals that characterize similarities between exact and approximate outcomes of SQL statements.

Ability to expand beyond SQL Figure 12 illustrates our plans with respect to extending the engine with a library-style support for machine learning and knowledge discovery (KDD) processes. As discussed in Section 6, it is indeed possible to use our data summaries to conduct processes like mRMR feature selection (Peng et al. 2005). Some other feature selection approaches seem to be easy to consider as well (Ślęzak 2009). The next steps can include implementations of some elements of deep learning (Hinton 2007) and redesign of some SQL-based decision tree induction algorithms (Nguyen and Nguyen 2005). Let us note that most of such algorithms rely on iterative data and model transformations. This constitutes a kind of conceptual equivalence between the work that we have already done within our framework with respect to approximate SQL and the work that is still ahead in the field of intelligent data processing. By analogy to Section 4, we need to develop efficient and accurate methods of expressing atomic machine learning operations by means of transforming summaries of their inputs into summaries of their outputs. Abstracting such operations can lead towards a powerful environment for interactive analytics, wherein the users could work with summaries of intermediate inputs / outputs via their favorite interfaces.

Integrating with other computational paradigms Figure 12 also summarizes opportunities related to putting our architectural strengths together with other trends in the areas of scalable and distributed data processing. SQL Layer denotes the currently existing layer of relational-query-focused transformations that – as outlined in Section 6 – can be successfully embedded into Spark environment (Zaharia et al. 2016) (see also the harnessing

resources dimension in Fig. 9). ML Layer represents methods discussed above (see also Chądzyńska-Krasowska et al. 2017). Capture Agent refers to the distributed knowledge capture layer that can save constructed histograms, etc., in a format uniformly accessible by SQL Layer, ML Layer and Spark's generic tools enabling the users to work directly with data summaries. This last aspect is especially interesting when combined with the previously mentioned idea of interactive analytics, which – in addition to intermediate inputs / outputs of machine learning algorithms – can embrace an access to summaries of simply filtered, joined and grouped data sets.

8 Conclusions

We presented a novel approximate query processing engine, which works by means of SQL-related transformations of granulated data summaries. The engine does not assume an access to the original data. Instead, it processes quantized histograms and a low percentage of co-occurrence ratios reflecting multi-column interdependencies detected in the buffered chunks of ingested data tables. Switching from the level of atomic data rows to low-footprint descriptions of their bigger clusters enables us to significantly decrease the computational cost of operations corresponding to filtering, joining, aggregating, etc. On the other hand, by dealing with summaries at the level of data chunks – unlikely in the case of other database solutions that do it for larger partition blocks or the whole tables – we are able to better control their quality and address complex trends occurring in the original data.

The discussed engine is available in its first production version and it is heavily tested by the first customers. The empirical observations confirm that it can be suitable for data owners and data-based services providers who cannot currently cope with exploring permanently growing data, or who simply want to lower the cost of resources required for data maintenance and analytics. Some of the already validated use cases include network traffic analytics and intrusion detection, digital advertising, as well as monitoring of industry processes. More generally, the users of similar engines can come from the fields of online applications, internet of things, sensor-based risk management systems and other tools related to machine-generated data.

We envision a number of interesting directions for further research and development. First of all, we can see a huge opportunity in strengthening connections of our overall approach with the theories of rough sets and granular computing, e.g., with respect to designing multi-level models of approximate summary-based computations. Moreover, although the existing approximate querying approaches based on data sampling do not seem to scale sufficiently in real-world scenarios, they provide the users with very helpful functionalities such as confidence interval derivations. Thus, one of our goals is to introduce such functionalities into our own framework, potentially by enriching our granulated data summaries with small subsets of original rows. However – like in the case of any other extensions – this will need to be done very carefully, so as to maintain the current performance characteristics, especially from the viewpoint of working with distributed and remote data sets.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., & Madden, S. (2013). The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases*, 5(3), 197–280.
- Agrawal, A., Choi, J., Gopalakrishnan, K., Gupta, S., Nair, R., Oh, J., Prener, D.A., Shukla, S., Srinivasan, V., & Sura, Z. (2016). Approximate computing: challenges and opportunities. In *Proceedings of ICRC* (pp. 1–8).
- Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., & Zaharia, M. (2015). Spark SQL: relational data processing in spark. In *Proceedings of SIGMOD* (pp. 1383–1394).
- Bruno, N., Chaudhuri, S., & Gravano, L. (2001). STHoles: a multidimensional workload-aware histogram. In *Proceedings of SIGMOD* (pp. 211–222).
- Cannataro, M., & Talia, D. (2003). The knowledge grid. *Communications of the ACM*, 46(1), 89–93.
- Chądzyńska-Krasowska, A., Betliński, P., & Ślęzak, D. (2017). Scalable machine learning with granulated data summaries: a case of feature selection. In *Proceedings of ISMIS* (pp. 519–529).
- Chądzyńska-Krasowska, A., Stawicki, S., & Ślęzak, D. (2017). A metadata diagnostic framework for a new approximate query engine working with granulated data summaries. In *Proceedings of IJCRS (1)* (pp. 6230–643).
- Cormode, G., Garofalakis, M.N., Haas, P.J., & Jermaine, C. (2012). Synopses for massive data: samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3), 1–294.
- Cuzzocrea, A., & Saccà, D. (2013). Exploiting compression and approximation paradigms for effective and efficient online analytical processing over sensor network readings in data grid environments. *Concurrency and Computation: Practice and Experience*, 25(14), 2016–2035.
- Fisher, D., Popov, I.O., Drucker, S.M., & Schraefel, M.C. (2012). Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of CHI* (pp. 1673–1682).
- Ganter, B., & Meschke, C. (2011). A formal concept analysis approach to rough data tables. In *Transactions on Rough Sets XIV* (pp. 37–61). Springer.
- Gatterbauer, W., Günemann, S., Koutra, D., & Faloutsos, C. (2015). Linearized and single-pass belief propagation. *Proceedings of the VLDB Endowment*, 8(5), 581–592.
- Gibbons, P.B., Matias, Y., & Poosala, V. (2002). Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems*, 27(3), 261–298.
- Glick, R. (2015). Current trends in analytic databases – Keynote Talk at FedCSIS. https://fedcsis.org/2015/keynotes/rick_glick.
- Hinton, G.E. (2007). Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11, 428–434.
- Hu, Y., Sundara, S., & Srinivasan, J. (2007). Supporting time-constrained SQL queries in oracle. In *Proceedings of VLDB* (pp. 1207–1218).
- Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., & Suel, T. (1998). Optimal histograms with quality guarantees. In *Proceedings of VLDB* (pp. 275–286).
- Kowalski, M., Ślęzak, D., & Synak, P. (2013). Enhanced rough SQL for correlated subqueries. In *Proceedings of FedCSIS* (pp. 1455–1462).
- Lazaridis, I., & Mehrotra, S. (2001). Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proceedings of SIGMOD* (pp. 401–412).
- Liu, Y., Li, D., He, W., & Wang, G. (2013). Granular computing based on Gaussian cloud transformation. *Fundamenta Informaticae*, 127(1-4), 385–398.
- Meila, M. (1999). An accelerated Chow and Liu algorithm: fitting tree distributions to high-dimensional sparse data. In *Proceedings of ICML* (pp. 249–257).
- Mozafari, B., & Niu, N. (2015). A handbook for building an approximate query engine. *IEEE Data Engineering Bulletin*, 38(3), 3–29.
- Naouali, S., & Missaoui, R. (2005). Flexible query answering in data cubes. In *Proceedings of DaWaK* (pp. 221–232).
- Narayanan, D., Donnelly, A., Mortier, R., & Rowstron, A. (2008). Delay aware querying with seaweed. *The VLDB Journal*, 17(2), 315–331.
- Neapolitan, R.E. (2003). *Learning Bayesian networks*. Prentice Hall.
- Nguyen, H.S., & Nguyen, S.H. (2005). Fast split selection method and its application in decision tree construction from large databases. *International Journal of Hybrid Intelligent Systems*, 2(2), 149–160.
- Ordóñez, C., & Cereghini, P. (2000). SQLEM: fast clustering in SQL using the EM algorithm. In *Proceedings of SIGMOD* (pp. 559–570).

- Pawlak, Z. (2005). Flow graphs and intelligent data analysis. *Fundamenta Informaticae*, 64(1-4), 369–377.
- Pawlak, Z., & Skowron, A. (2007). Rudiments of rough sets. *Information Sciences*, 177(1), 3–27.
- Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8), 1226–1238.
- Ramnarayan, J., Mozafari, B., Wale, S., Menon, S., Kumar, N., Bhanawat, H., Chakraborty, S., Mahajan, Y., Mishra, R., & Bachhav, K. (2016). SnappyData: a hybrid transactional analytical store built on spark. In *Proceedings of SIGMOD* (pp. 2153–2156).
- Ślęzak, D. (2009). Degrees of conditional (in)dependence: a framework for approximate bayesian networks and examples related to the rough set-based feature selection. *Information Sciences*, 179(3), 197–209.
- Ślęzak, D., Synak, P., Wojna, A., & Wróblewski, J. (2013). Two database related interpretations of rough approximations: data organization and query execution. *Fundamenta Informaticae*, 127(1-4), 445–459.
- Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., & Murthy, R. (2009). Hive – a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2), 1626–1629.
- Weiss, Y., & Pearl, J. (2010). Belief propagation: technical perspective. *Communications of the ACM*, 53(10), 94.
- Yao, Y. (2016). A triarchic theory of granular computing. *Granular Computing*, 1(2), 145–157.
- Zadeh, L.A. (1997). Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90(2), 111–127.
- Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11), 56–65.