Position Papers of the Federated Conference on
Computer Science and Information Systems pp. 53–60

DOI: 10.15439/2015F288
ACSIS, Vol. 6

Unsupervised Extraction of Graph–stream Structure for Purpose of Knowledge Retrieval and Information Fusion

Radosław Z. Ziembiński
Poznan University of Technology
Faculty of Computing
Piotrowo 3, 60–965 Poznan, Poland
Email: radoslaw.ziembinski@cs.put.poznan.pl

***Abstract*—Technologically inevitable introduction of various kinds of sensors to our life resulted in the production of huge amount of data delivered as streams. An improper acquisition of information may lead to errors caused by mixing observations coming from different processes threads. Some remedy can bring a proper representation of information. Hence, this paper introduces a graph–stream structure representing performance of complex multi–threaded process. The proposed network representation can separate information describing multiple threads and allows for modeling causal relationships between them. It gives separated and segregated information opening opportunity for development of qualitatively better and simpler knowledge retrieval algorithms. Further, the paper delivers a method for this representation extraction from multivariate data stream. It would be done by a clustering algorithm particularly designed for this purpose and evaluated quantitatively and qualitatively on example sets of data.**

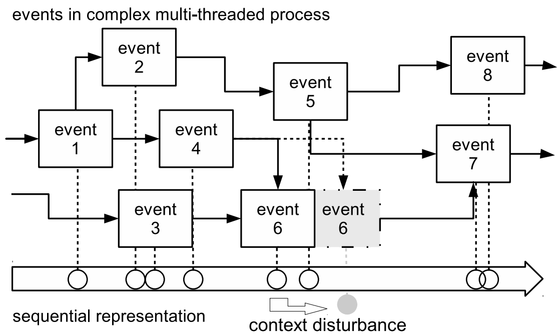

Fig. 1. The structural context "flattening" after the serialization of information.

I. Introduction

A RECENT decade revealed even more profoundly how modern life interleaves technology and habits. Miniaturization gained momentum when the processing performance and storage capacity (in relation to price) reached a threshold allowing for inexpensive execution of complex algorithms used in knowledge retrieval and machine learning. Then, it became possible to adjust functionality and interface of the device to the user expectations in semi–automatic manner using machine learning methods.

Modern smart devices have to handle big amounts of data and provide feedback to the user in a reasonable time. Services quality usually depends on the performance of the knowledge retrieval. Unfortunately, the deadline put on the processing time is rather difficult to met by many categories of data mining algorithms developed for stationary data sets. Particularly it happens, if the search space exponentially depends on the input size.

Another difficulty may arise if the observed process is a complex one (e.g., multi–threaded) but the sensor used to make observations is relatively simple. Then, the acquired data stream contains information which is a superposition of observations coming from different objects. Direct handling by common motifs finding or patterns mining algorithms may be difficult in the case. Simply speaking, we cannot identify particular objects and separate them easily for the mining purpose. Thus, even slight lack of synchronization in the recorded objects behavior can reorder parts of data. It may lead to low quality results whose application poses risks. An example ambiguity introduced by such distortions is illustrated on Fig. 1, where a small shift in events order changes historical circumstances. In the context of information fusion, it is more difficult to benefit from synergy in observations made by different kinds of tools. It happens, because direct comparison of context is not possible after the "collapse" of information collected from differently synchronized threads.

The confidence in the processing can be improved if the simple sequential representation will be replaced by a vessel crafted particularly for the purpose of knowledge retrieval. Coming from such motivation, this paper provides study of the research on a new graph–stream data structure. An introduced container is a network graph capable of storing separated information about episodes observed in the process. Its nodes carry segregated information about events while edges can describe complex causal relationships between them. According to author expectations this structure should solve some from above issues. Mainly, these related to unreliable context construction where events mixture describing different threads try to define causal relationships.

Looking at literature, knowledge retrieval focuses partic-

©2015, PTI

ularly on the patterns mining in data streams. The context separation problem can be partially solved, if data are converted to chronologically ordered set of sequences. Then, each sequence may describe a single subprocess. Such method can be supported by various data mining algorithms [1], [2], [3], [4], [5]. Even, if this approach seems to be better from the single stream approach, this representation still loses information about conditional dependency between constituent sequences. A similar approach has been used also in the context of multivariate data sets [6].

The conditional dependency can be preserved, if sequential representation would be replaced by relevant multidimensional data structure. Particular interest raises graphs as natural extension of sequential representation on multidimensional ones. However, ways to construct graphs from sequential data can be very diverse. The mining of the graph–based representation has been studied for stationary and non–stationary graphs e.g., [7], [8], [9]. Unfortunately, exploration of general graphs is computationally expensive since verification of graph isomorphisms is non–polynomial [10] (but sub–isomorphisms is NP–complete [11]). These considerations lead to the conclusion that the compromise solution should be sought in specific families of graphs e.g., directed acyclic graphs. In this spirit, there have been proposed methods for mining partial orders in sequential data [12], [13].

This paper contributes to the state of art by an introduction of the graph–stream structure and algorithm for its extraction from observations collected by set of sensors. It has a following outline. A next section introduces the graph–stream definition. Then, the paper delivers the extraction algorithm which can be used to obtain the graph–stream from the stream of observations. A following section contains a presentation of results from the algorithm performance evaluation on artificial data sets. Reported experiments have involved finding frequent episodes in the graph–stream and measurement of their autocorrelation. Remaining part focuses on conclusions.

II. Graph–stream definition

The proposed data structure is intended to store information about observations of a complex process. It is assumed that the observed process includes few subprocesses that produce stimulus to sensors simultaneously. Alternatively, the process is observed by different sorts of sensors from various perspectives at once. In a consequence, the acquired data stream usually contains a mixture of information generated by observed subprocesses. Introduced structure and algorithm should keep it separated for the purpose of the following processing.

It was already mentioned that the graph–stream is the customized directed acyclic graph. Let's begin its introduction from a definition of some basic carriers of information. In our case, the processed data stream is described by a set of nominal or continuous attributes A.

Subsets of A determine informational content of graph nodes. Node $n_{p_{type},i}(p_{ts}, A_i) \in N$ is a unit of information collected at one moment. It is described by type p_{type}, event occurrence time–stamp p_{ts} and attributes' values $A_i \in A$. Directed edge $e_{i,j}(n_i, n_j) \in E$ connects nodes n_i and n_j and represents causality relation between nodes. It uses the symbol $\rightarrow$.

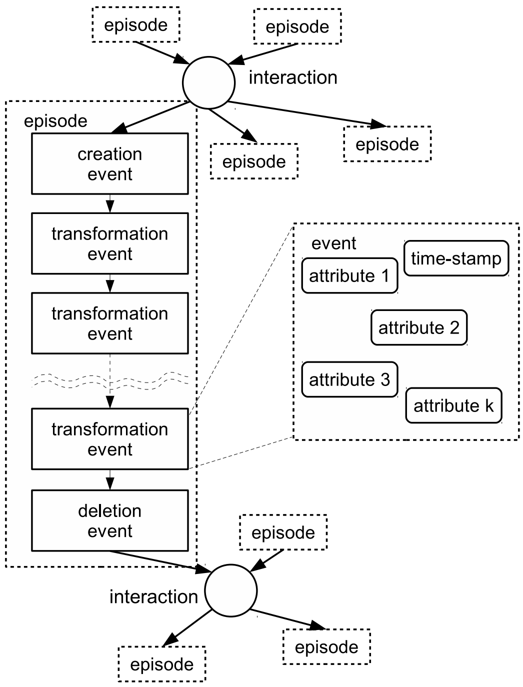

Fig. 2. The graph–stream data representation.

Above definitions lead to formulation of the directed acyclic graph. A directed cycle is a sequence $n_1 \rightarrow n_2, ..., n_{k-1} \rightarrow n_k \in N$ of nodes collected along a path made from directed edges where $n_1 \equiv n_k$. $G_{DAG}(N, E)$ is a directed graph where the cycle is absent.

The graph–stream uses distinct types of nodes to describe static and dynamic properties of observed objects:

- State node $n_{S,i}(p_{ts}, A_i) \in N$ collects all values of attributes describing state of a particular observed object in the process. This node is used to represent a boundary state of the episode just after its creation or deletion. However, it can be produced from the interaction, too.
- Transformation node $n_{T,i}(p_{ts}, A_i) \in N$ describes a modification of a single observed object. However, it is self modification without external influences. This node represents the transformation event.
- Interaction node $n_{E,i}(p_{ts}, A_i) \in N$ describes an observed incident involving an interaction between objects from different threads. At this moment episodes that went into the interaction event collapses and produce a new non–empty set of episodes. It emerges from this definition, that node binds causally set of interacting episodes to their products. This node has only the time–stamp attribute.

There is no constraint in this proposal shaping attributes distribution between state and transformation nodes. It is only a suggestion to keep boundary states information in state nodes and operations or changes of the episode state in transformation nodes.

Above definitions have to be complemented about constraints imposed on edges. The interaction node joins episodes by causal relation. According to graph–stream's structural assumptions it can connect only a non–empty set of deletion nodes to a non–empty set of creation nodes. It is many–to–many relation reflecting causality dependencies between episodes. Relationship between transformation and state nodes have simpler interpretation. They can be connected only by single ingoing and outgoing edges. Hence, they form a sequence beginning from the creation node, lasting through transformation nodes and ending with the deletion node. It is called the episode (body).

Finally, let us define *the graph–stream* $G_{ST}(N, E)$ as a set of episodes S connected by interaction nodes according to above constrains imposed on edges. Illustrations of introduced structures are drawn on Fig. 2 at different levels of details.

III. Graph–stream Extraction Method

A. Episodes identification

Algorithm proposed for the graph–stream extraction converts data stream in three subsequent phases. Due to complexity and size of the algorithm code it is difficult to describe all details. Hence, this description primarily put attention on the most important design features. It should give sufficient hints about algorithm structure to understand its construction. For matter of convenience, the algorithm will be called *GATAC* (GrAph–sTreAm extraCtor).

The first phase of the processing is handled by a dedicated on–line clustering algorithm. Its overall pipeline for processing new data objects and performing the graph–stream extraction presents Fig. 3.

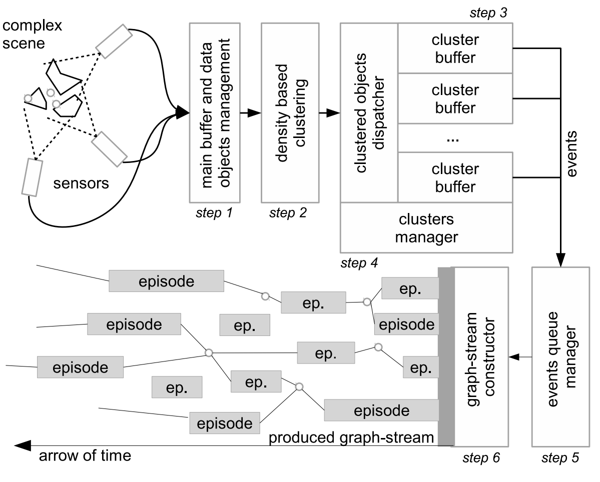

Fig. 3. The process of the graph–stream extraction.

Proposed implementation of the algorithm uses a fixed set of attributes A. However, it does not enforce their full usage for each processed data object. This clustering is performed within groups of data objects described by the same sets of attributes (subsets of A). In a consequence, this method can construct the graph–stream from separate sources of data even if they share some or neither attributes.

Let's now describe life–cycle of a single data object processed by this algorithm. At the beginning, values of data object attributes are normalized. It is a necessary initial step because later a similarity function aggregates partial similarities calculated for compared attributes. The normalization is done according to intervals approximating attributes domains ranges (updated on–line). Then, incoming data objects are sorted in B according to values. If the main buffer appears to be full then the oldest data object is removed from B to make a free space for the new one (step 1, Fig. 3).

After placing the data object in B, the algorithm begins sending messages to objects in the neighborhood to determine the set of the most similar neighbors. Messages that are circulating between data objects are stored in two structures: a message queue and replies sorted list. Those from new data objects are stored in message queues of neighbors. In this implementation all structures storing messages have fixed sizes to delimit the memory usage. Limits prevent from accepting too many messages from other data objects at the cost of neighborhood identification accuracy (step 2).

The procedure is performed in several subsequent iterations to determine the neighborhood of specified size. By this way the algorithm performs iteratively the neighborhood search in the breadth–first manner.

$$
\begin{array}{rcl}
\sigma_o(o_1, o_2) & = & (\delta_d(o_1, o_2) * \delta_t(o_1, o_2) \\
& & *\delta_l(o_1, o_2))^{1/3} - 1 \\
\text{where:} & & \\
\delta_d(o_1, o_2) & = & 1 + (\sum_{i=1..|A_{o_1,o_2}|} |o_1.A[i] \\
& & -o_2.A[i]|)/|A_{o_1,o_2}| \\
\delta_t(o_1, o_2) & = & 1 + (|A_{o_1,o_2}| * |o_1.tstamp \\
& & -o_2.tstamp|)/(|B| * |B.tspan|) \\
\delta_l(o_1, o_2) & = & 2 - (o_1.labels \cap o_2.labels) \\
& & /(o_1.labels \cup o_2.labels)
\end{array}
\quad (1)
$$

Afterward, the algorithm browses all non–empty message queues and prepares replies to their senders. They contain information about similarities between pairs of data objects. The range of replies is delimited by a decreasing skip counter. In a consequence, only the closest neighbors are visited by replies sent to new data objects.

Equation 1 is used to calculate similarity between two data objects. If it is equal to 0, then objects are identical. The similarity computation takes into account three properties of data objects pair i.e., difference of attributes values $\delta_d(o_1, o_2)$, timestamps $\delta_t(o_1, o_2)$ and labels $\delta_l(o_1, o_2)$. Variables and constants used in the equation have following explanation: set of common attributes for pair of objects $A_{o_1,o_2} \subseteq A$, data object's time stamp $o.tstamp$, main buffer B, time span $B.tspan$ (calculated on–line from timestamps of stored data object's), labels associated to data object $o.labels$. Labels are nominal values that can be used for further differentiation of data objects by describing e.g., data source properties.

After the sender data object (the new one) received all replies from close neighbors, a swapping procedure begins to filter out some of them. This operation is performed to preserve smoothness of the cluster distribution. In the result, data objects from subspaces containing different densities become better separated, even if they adjoin. It prevents from merging sparse clusters to denser ones (if the density proportion is above the parametrized threshold) and excludes noise.

If the replies list becomes empty, then the new data object receives a new cluster identifier. However, the new cluster buffer is created only if the second object appears with the same identifier. Hence, a standalone object does not invoke creation of the cluster buffer and new thread. It makes the processing more efficient by eliminating noise.

At this moment, data objects are moved to the cluster buffer assigned to a single thread that can produce one or more subsequent episodes (step 3). An event signaling the cluster creation is thrown if its population passes the parametrized threshold. The threshold ensures that the statistics from data distribution in the cluster buffer are robust.

The newly added data object can merge two or more clusters if both are located in its neighborhood (step 4). The algorithm maintains alteration counters assigned to cluster buffers. If their values become greater than threshold Θ_C, then the algorithm begins a breadth–first introspection of the main buffer. It is performed according to neighborhood information stored in replies lists of objects. This procedure rewrites all identifiers of data objects from clusters that become connected since the previous introspection. The new cluster identifier value is taken from the most populous contributor to preserve the strongest supported thread. This operation throws interaction event, if the merger occurs.

If the main buffer is full, then the algorithm removes the data object according to FIFO rule. Removal procedure also modifies the alteration counter associated to the cluster. So, it may trigger the breadth–first introspection of the main buffer, too. If it happens, a fragmentation of the cluster may be revealed. Then, each new fragment of the original cluster receives distinct new cluster identifier, while the most populous one retains the previous one. Additionally, a relevant interaction event is produced.

The cluster buffer describing episode may be discarded if its support terminates. It happens, after it has not been supported for a time longer than average "time distance" between data objects already stored in the cluster multiplied by the parametrized threshold. Such termination procedure facilitates clusters supported at different rates by input data. Moreover, it is robust to a slow drift of the support frequency.

B. Detection of transformation events

The detection of transformation events is necessary to construct the episode body. It uses the cluster tracing mechanism to detect significant changes reflecting shifts in data distribution. At this stage, it can be done relatively simply due to the fact that the clustering phase binds each episode to the life–cycle of a single cluster.

The data drift tracing begins just after sending of the interaction event related to the episode creation. After this event, the algorithm can calculate plausible statistics and send following transformation events notifying about changes in the data distribution. To prevent from unnecessary recalculation of statistics, the algorithm uses modification counters assigned to clusters. The counter is incremented each time when new data object is added or removed from the cluster buffer. Statistics become recalculated if it passes Θ_C.

Algorithm 1 Procedure for transformation events detection.

Require: Set of clusters - C, set of modified statistics - M, mean threshold - Θ_{mean}, standard deviation threshold - Θ_{mean}, density threshold - Θ_{dens}
Ensure: Generated data object - o

```
for all c ∈ C do
  if c.modificationsCounter > Θ_C then
    c.modificationsCounter = 0
    M = c.calculateStatistics(Θ_ang, Θ_mean,
    Θ_std, Θ_dens)
    if M <> ∅ then
      c.updatePreviousStatistics(M)
      c.fireTransformationEvent()
    end if
  end if
end for
```

In the current implementation, calculated statistics include means and standard deviations calculated for each attribute alone. Additionally, the event may include average density of the cluster. Their calculation is performed according to Alg. 1. Current calculated values are compared to previously determined state ps and the reference state rs reported in the preceding event. If statistics pass adequate thresholds (mean, deviation and density) for attributes, then the procedure sends the transformation event with the current state (or optionally information about changes). The step also involves replacement of modified statistics from the reference state by current ones. Hence, supporting procedure detects drift occurring for linear changes in data statistics. It reduces output information by preventing from a flood of following events delivering information about linear shifts in data distributions.

The construction of the graph–stream is a simple step. Events contain full information about predecessors, successors and episodes affiliation inherited from cluster buffers. It is used for building a graph–stream just by extending a set of episodes tails by incoming events (step 5 and 6). At interaction, participating episodes are closed and bound to result in a form of newly created tails of following episodes. For this purpose, the proposed algorithm uses a map of episodes M that allows for fast manipulation of them.

C. Computational complexity and scalability of the extraction algorithm

The computational cost of the algorithm depends on data dimensionality, sizes of the main buffer and cluster buffers.

TABLE I
PARAMETERS OF THE GRAPH EXTRACTION ALGORITHM.

Parameter:	$\|B\|$	$\|C\|$	Θ_ρ	Θ_σ	Θ_C	Θ_T
Value:	500	40	2	2	16	2
Parameter:	Θ_{mean}	Θ_{std}	Θ_{dens}			
Value:	0.1	1.5	1.5			

The data in the main buffer are sorted thus the complexity is bound to $O(|A| * log(|B|))$, where $|B|$ is the size of the main buffer. Its size determines the assignment of new data objects to cluster buffers and data distribution "forgetting". On the other hand, size of each cluster buffer is relatively small. It has to be sufficiently large for accountable statistics calculation (few dozens of data objects). Data objects update procedure requires $O(|A| * |B|)$ steps for maintaining messages and replies. However, the calculation of statistics requires a double loop on each cluster's data set. Therefore, the complexity is $O(|A| * |C|^2)$ where $|C|$ is the size of the cluster buffer. Fortunately, statistics are recalculated only if the cluster's modification counter becomes greater than the threshold Θ_C. Hence, the rate of statistic recalculation is delimited by $1/\Theta_C$. The construction cost of the graph–stream is in order of $O(|A| * |C| * log_2(|M|))$ if all cluster buffers would send messages to neighbors.

Memory costs of the algorithm have been delimited by fixed buffers sizes. It refers to the main buffer $|B|$, clusters (buffers) $|C|$, both messages queues attached to data objects and used for the neighbors finding. The described implementation prefers the control on memory usage over the results quality. This design assumption allows its deployment on mobile or embedded devices. Of course, it is feasible until small buffer sizes would immerse the processing quality below acceptable level.

Algorithm 2 Synthetic data generation algorithm.

Require: Clusters radixes - $crad$, time step - $cstep$ and variability - $tvar$, data object dimensionality - $|A|$, algorithm mode - $mode$, number of clusters - cno
Ensure: Generated data boject - a
  {make a timeshift}
  $iteration = iteration + 1$
  $timeStep = randomUniform(cstep, cstep + tvar)$
  $phase = phase+speed*timeStep$; $sPhase = sin(phase)$
  $oidx = iteration\%cno$
  if mode == 3 **then**
    $cobj = (int)(phase/PI)$
    if random(0, 1) ¡ abs(sPhase) **then**
      $oidx = cobj\%cno$
    end if
  end if
  {select cluster identifier}
  $eidx = oidx/2 + 1$
  $dirId = (oidx\%2 == 0)?(eidx) : (\ eidx)$
  {calculate attributes' values}
  for all d $\in$ 1...$|A|$ **do**
    $dir = (dirId\&(1 << d))?(1) : (-1))$
    $dir = dir * (1.0 - crad)$
    if mode == 1 **then**
      $disp = randomNormalDist(0, crad)$
      $a[d] = (dir * sPhase + disp)/2 + 0.5$
    else if mode == 2 **then**
      $disp = randomNormalDist(0, 3 * (crad + 0.0001) * absT(sPhase))$
      $a[d] = (dir * crad * 2 + disp)/2 + 0.5$
    else if mode == 3 **then**
      $disp = randomNormalDist(0, crad)$
      $a[d] = (dir * crad * 5 + disp)/2 + 0.5$
    end if
  end for
  return a

IV. EXPERIMENTAL EVALUATION

The purpose of experiments was to evaluate the process description stored in the extracted graph–stream. Described experiments were conducted on data obtained from real and artificially generated streams. Different kinds of experiments were performed to measure the algorithm's performance efficiency and its extraction accuracy. During them, a quality has been performed by measurement of episodes autocorrelation. The process of autocorrelations identification can be related to the problem of finding frequent item sets in nominal data stream [14], [15].

The extraction algorithm was evaluated in experiments with default settings on Table I. The main buffer size was set to $|B| = 500$ and clusters buffers sizes were delimited to $|C| = 40$.

A. Description of data sets

Artificial data sets are generated by procedure that produces data objects describing a group of interacting clusters. There are three modes of the generator described in Alg. 2. The data generation algorithm selects the cluster that produces data objects according to round–robin. Then, the generation is performed by distorting the cluster's center with a random value from Gaussian distribution.

The described generator has three modes of operation. For $mode = 1$, the generator forms hyper–spherical clusters with movement driven by sinusoid. They are moving forth and back, from edges to the center of hyper–box. If $mode = 2$ then it generates pulsating clusters. This process produces data objects intermixed in the middle of the hypercube in repeatable intervals. The final one is generated for $mode = 3$. It generates "blinking" clusters that gain and lose support periodically. In this case, clusters do not change their position or size but their density oscillates. These generated data streams show Fig. 4.

These distributions represent different kinds of interactions between observed objects. In the first case, the behavior of clusters leads to their complete coverage in the middle of the hypercube. Then, subprocesses become indistinguishable. The second stream describes subprocesses that interfere partially.

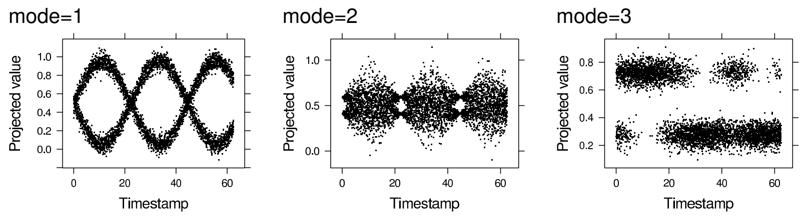

Fig. 4. Artificial data streams obtained from the generator.

The interference manifests itself as overlapped distributions. In the last case, there are no interactions between clusters at all. Observed subprocesses do not interact and their oscillations are relatively fast.

B. The processing performance measurement

Measurement of the processing efficiency has been done on artificial data stream containing 100000 data objects. It was generated for $mode = 1$. This experiment was performed on data streams generated for different clusters numbers at different dimensionality and processed with various main buffer sizes.

The conducted experiment included four sub–experiments from whom results are presented on Fig. 5. The top–left plot of Fig. 5 contains performance measurement of the graph–stream extraction algorithm for data of different dimensionality. This experiment was conducted for $a1 : |A| = 12$, $a2 : |A| = 9$, $a3 : |A| = 6$, $a4 : |A| = 3$ and $|B| = 500$. It can be noticed, that the number of dimensions have impact on the algorithm's iteration performance. It acknowledges the theoretical analysis of the main buffer and messaging performance. The main buffer has size $|B|$ and consists of $|A|$ sorted lists. A low cost of logarithmic access to B is reflected in results. The top–right plot delivers measurement of the computational costs for different main buffer sizes. Results were obtained for data dimensionality $|A| = 9$ and $b1 : |B| = 500$, $b2 : |B| = 800$, $b3 : |B| = 1100$ and $b4 : |B| = 1400$. Impact of main buffer size on the iteration cost is small because the binary search is used to get access to sorted objects. For both above experiments cno was equal to 2. All measurements include time required to store the produced graph–stream in memory. It explains slightly increasing costs during the processing.

Bottom plots describe the dependence of the extraction algorithm performance on data complexity. These experiments have been performed for stream carrying different number of clusters in the stream $c1, d1 : cno = 2$, $c2, d2 : cno = 4$, $c3, d3 : cno = 6$ and $c4, d4 : cno = 8$. The results were measured for $|A| = 9$ and $|B| = 500$. The bottom–left plot contains the processing cost measured per iteration, while the bottom-right one delivers cumulative numbers of generated events by the algorithm for the graph–stream extraction. Intensity of all data streams was the same. Therefore, the frequency of data objects per cluster was lower for streams delivering more clusters. It can be observed that the costs of the processing is the greatest for the smallest number of clusters at $cno = 2$. This result seems to be counterintuitive but it is caused by more intense messages forwarding. For a lower number of denser clusters the count of connections between data objects is relatively higher. Thus, the cost of handling communication is higher, too. Results form the bottom–right plot can be interpret according to intuition since it is clear that more clusters requires proportionally more events generated to describe them.

All experiments involving measurement of the performance were conducted on x64 workstation at clock 3 GHz. Each measurement was an average from 1000 following iterations. Microseconds scale oscillations observed on plots are presumably caused by hardware, operating system or C++11 standard libraries. All experiments have been performed on multi–core hardware. The affinity of processes to cores changed during the processing what might promote oscillations.

C. Similarity of episodes

Evaluation of autocorrelation requires a similarity measure that would allow for mutual comparison of episodes. Similarity computed for a pair of nodes takes into account differences in attributes values, the relative occurrence times (in the respect to timestamps of the episodes heading nodes) and densities. It is a geometrical mean from above partial similarities. To calculate partial similarity, attributes values difference dif is transformed with function $exp(-abs(dif) * weight)$. It standardizes results by scaling them using weights chosen per attribute. Then results are averaged and partial similarity of attributes is computed. The densities and occurrences times are threated separately as qualitatively distinct entities. But their differences are also standardized by that weighted exponential function. Weights help in the similarity function tuning. They can be used when we need to underscore particular property. Calculated similarity takes continuous values from 0 to 1. The value equal to 1 means that nodes are identical and related events have occurred in the same relative time measuring from respective episodes heads.

Episodes are evaluated according to fairly complicated procedure. Initially, pairs of state nodes beginning and terminating episodes are compared. Then, all possible pairs of transformation nodes from both episodes are enumerated to calculate their similarities. Afterwards, pairs are sorted according to similarity value and there are chosen ones with the greatest similarities. During this selection, it is forbidden to take two pairs containing at least one transformation node the same. In the result, only the strongest set of ties between the pair of distinct episodes survives selection. The selection is done

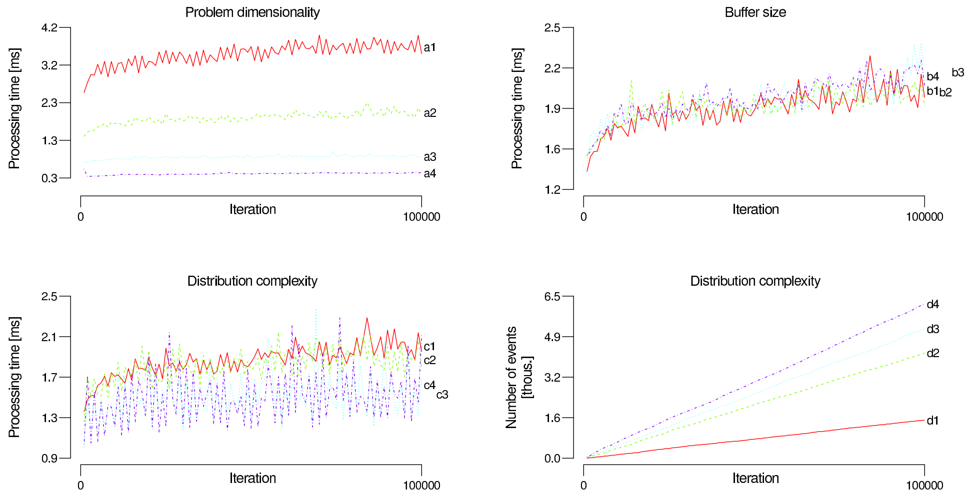

Fig. 5. The graph–stream extraction process efficiency.

regardless sequential order of transformation nodes in the episode. Fortunately, the order of nodes is taken into account elsewhere. Let's remind that the time–stamp difference is considered in events similarity calculation. Therefore, a pair of events occurred at different times in reference to their episodes heads time–stamps have low similarity and chance to be selected.

Global similarity is calculated as an average of similarities obtained from the comparison of pairs state and transformation nodes. It delivers a normalized result, where an episode compared to itself would receive the score equal to 1. The aggregation raises status of state nodes in relation to inner transformation nodes underscoring importance of border states of episodes. The introduced similarity function favors episodes containing the same attributes values in nodes and the same order of transformation nodes. Therefore, pairs of episodes that differ in duration would have lower similarity.

D. Autocorrelation of episodes in experiments

Results presentation begins from ones obtained for artificial data streams. These streams provided data describing 4 dynamic clusters (subprocesses) in 9 dimensions. Heat maps presents mutual similarities for episodes that belong to graph–streams and extracted from artificial data sets are presented on Fig. 6. They show only 200 of the most mutually similar episodes. Two figures for each graph–stream represent similarities of episodes (left) and contexts preceding them (right). Bottom legend contains information about correlation coefficient value between episodes and their contexts. Episodes are sorted according to their length (a number of transformation nodes) and the longest ones are located in the top–right corner of each heat map.

Fig. 6 reveals existence of similarities between episodes for periodic processes. There are observable groups of mutually similar episodes that have comparable sizes. They form "squares" located on the diagonal. They describe different stages of the subprocess evolution in a cycle and reveal characteristic lattice patterns inside. It is caused by the fact that data carry the description of 4 clusters that behave symmetrically. Thus, episodes describing them have very similar sizes and find their places in the same "square". Autocorrelation between episodes tells about a periodicity in the observed process and acknowledges the generator properties.

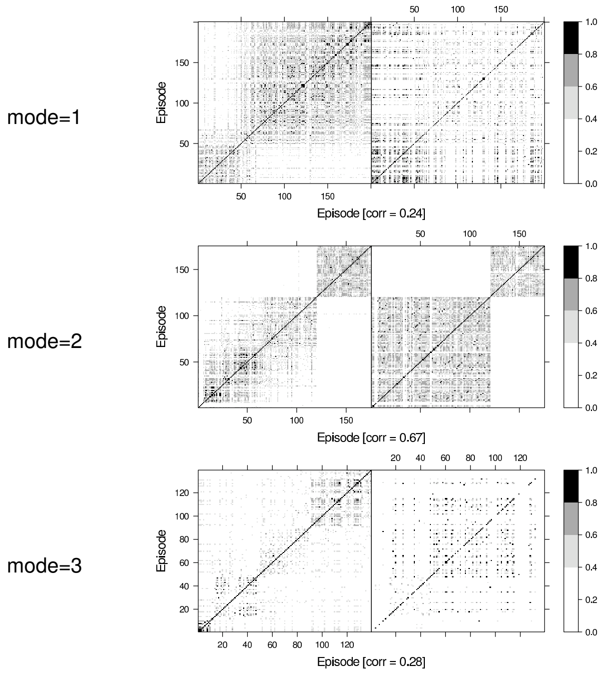

Fig. 6. Heat maps representing autocorrelation of episodes (the left column) and their the best contexts contributors (the right column) for artificial data streams.

Contributing contexts similarities are weakly correlated to related episodes. Aside from the artificial set generated at $mode = 2$, almost all correlation coefficients are very low. This suggests that the past context information has to be analyzed by looking deeper to the past. It is in line to conclusions made for the sequential patterns mining where frequent elements of the pattern may be separated by many infrequent elements [1]. However, for some data sets e.g., ones generated at $mode = 2$ the correlation occurs using this algorithm even for adjacent episodes. Improvement in the cluster buffer tracing algorithm and casting transformation event may increase the efficiency of the correlated episodes finding.

Obtained results acknowledge that the exploration methods for finding patterns in graph–stream threads have to look deeper into past contexts of episodes. The "lattice–square" patterns on heat maps prove that the method correctly identifies episodes from distinct subprocesses. Because all experiments used the same parameter values therefore we can expect a further results improvement for algorithm parameters tuned to properties of particular data stream.

V. Conclusions

The proposed graph–stream extraction algorithm can produce directed acyclic graphs describing multivariate and multi–modal data streams. It has some advantages over commonly used sequential representation. In the first place, it can separate individual subprocesses within the observed complex process. This feature may be useful when we observe the process through array of heterogeneous detectors with one sink of data. Concluding, the data structure proposed in this paper may contribute in following areas to the state of art:

- It can help to identify undisturbed context of events. Events from different threads (subprocesses) become segregated and organized in episodes. Such structure is more robust when it comes to issues with synchronization for concurrent subprocesses. This allows for building simpler algorithms for knowledge retrieval and information fusion. This representation does not transform information stored in attributes. It just makes observations more understandable for the following processing by better organization.
- Network structure can be used to model complex causal relationships between events. Causality can be better reflected since DAG can represent relations many–to–many between dependent episodes. Preservation of attributes values from the original data stream and exposition of data dynamism would be useful in observation and analysis of dynamic processes.
- There are different sorts of nodes for representing interactions, transformations and border states. They form a grammar of a simple language e.g., like Feynman diagrams in physics this language is sufficiently capable to express a description of almost all discrete processes.
- It can lead to novel algorithms better exploring information about relationships between subprocesses in the complex process. This makes new opportunities for episodes clustering, classification, forecasting and correlations mining. In my opinion, a domain related to correlations mining between subprocesses is particularly interesting (regarding a subsequent information fusion). It may lead to new methods of information processing benefiting from synergy of data retrieved simultaneously from many qualitatively different sensors.

Acknowledgment

This paper is a result of the project financed by National Science Centre in Poland grant no. DEC-2011/03/D/ST6/01621.

References

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. of the Eleventh International Conference on Data Engineering*, ser. ICDE '95. Washington, DC, USA: IEEE Computer Society, 1995. ISBN 0-8186-6910-1 pp. 3–14.

[2] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, p. 215. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2001.914830

[3] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, "Multi-dimensional sequential pattern mining," in *Proc. of the Tenth International Conference on Information and Knowledge Management*, ser. CIKM '01. New York, NY, USA: ACM, 2001, pp. 81–88. [Online]. Available: http://dx.doi.org/10.1145/502585.502600

[4] R. Ziembiński, "Algorithms for context based sequential pattern mining," *Fundam. Inf.*, vol. 76, no. 4, pp. 495–510, Dec. 2007.

[5] M. Plantevit, A. Laurent, D. Laurent, M. Teisseire, and Y. W. Choong, "Mining multidimensional and multilevel sequential patterns," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, pp. 1–37, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1145/1644873.1644877

[6] D. Marinazzo, M. Pellicoro, and S. Stramaglia, "Causal information approach to partial conditioning in multivariate data sets," *Comput Math Methods Med.*, p. 17, 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/303601

[7] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proc. of the 2002 IEEE International Conference on Data Mining (ICDM02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 721. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2002.1184038

[8] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin, "On dense pattern mining in graph streams," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 975–984, Sep. 2010. [Online]. Available: http://dx.doi.org/10.14778/1920841.1920964

[9] A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, "Mining frequent closed graphs on evolving data streams," in *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 591–599. [Online]. Available: http://dx.doi.org/10.1145/2020408.2020501

[10] U. Schöning, "Graph isomorphism is in the low hierarchy," *J. Comput. Syst. Sci.*, vol. 37, no. 3, pp. 312–323, Dec. 1988. [Online]. Available: http://dx.doi.org/10.1016/0022-0000(88)90010-4

[11] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976. [Online]. Available: http://dx.doi.org/10.1145/321921.321925

[12] H. Mannila and C. Meek, "Global partial orders from sequential data," in *Proc. of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 161–168. [Online]. Available: http://dx.doi.org/10.1145/347090.347122

[13] R. Gwadera, G. Antonini, and A. Labbi, "Mining actionable partial orders in collections of sequences," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, Eds. Springer Berlin Heidelberg, 2011, vol. 6911, pp. 613–628. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23780-5_49

[14] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *Proc. of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 487–492. [Online]. Available: http://dx.doi.org/10.1145/956750.956807

[15] M. Deypir and M. H. Sadreddini, "A dynamic layout of sliding window for frequent itemset mining over data streams," *J. Syst. Softw.*, vol. 85, no. 3, pp. 746–759, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2011.09.055