https://doi.org/10.2352/ISSN.2470-1173.2018.01.VDA-377

A Step Towards Automatic Visual Analytics Pipeline Generation

Benjamin Karer, Inga Scheler, and Hans Hagen
University of Kaiserslautern, Germany

Abstract

Automatic generation of data visualizations allows to quickly deploy data visualizations. In visual analytics, the combination of automatic and human analysis increases the effort necessary to achieve similar effects substantially. Where automatic visualization only needs to map the data, in visual analytics the whole data preparation and processing pipeline has to be considered. The user is interested in representations reflecting certain interpretations of the data, for example the idea that different groups represent different clusters in the data. In this paper, we prove that an information-driven automatic design of visual analytics pipelines is feasible. To this end, we prove that the ability of an analysis system to derive and visualize data supporting inquired information is decidable – at least for real-world applications. Having overcome this major obstacle, we outline a general algorithm scheme that can be implemented on a wide range of data and information models.

Introduction

Combining automatic data analysis with human reasoning based on visualization, visual analytics has become an integral component of modern data analysis applications. While individual advances in visualization, data mining, and machine learning contribute this success, the key element of visual analytics is the efficient combination of the different techniques to obtain solutions fostering the derivation of new insight. However, even today, more than ten years since visual analytics emerged as a field, this integration can still be quite challenging. While tools have been developed to support the efficient generation of data preparation and processing pipelines, finding a combination of algorithms that reveals the insight an analyst is aiming to obtain still depends on the analyst's understanding of the algorithms' effects on the data and experience in their application.

Towards a more efficient process of generating data preparation and processing pipelines for visual analytics, we propose a partial automatization of the process. The key idea is to let the computer reproduce information inquired by the user. To achieve this aim, it needs to find a sequence of transformations that derive data supporting the inquired information from the available raw data. Yet, there is a twist. The algorithm would necessarily decide whether it is actually possible to derive this information as the interpretation of the result of a sequence of transformations applied to the raw data. This is an instance of the halting problem – and therefore impossible. Fortunately, there are special cases, in which a restricted version of the problem is decidable. A second problem is that by information, we do not mean data but its interpretation with respect to the analysis question. An example would be treating distance as an indicator for neighborhood. Therefore, the user can only query for information that is already to be found somewhere in the deduction system – and thus already known. As it turns out, the trick is to specify the characteristics of the results of data analysis and visualization algorithms. The analyst then asks for a view on the data that has specific properties and for interpretations that are explicitly associated with the results of data transformations as properties of the transformation's algorithm. Ideally, this view allows an efficient evaluation of conjectures and hypotheses against data but can still be explored easily.

In this paper, we introduce an approach to the automatic design of visual analytics pipelines driven by the information associated with the results of data transformations. To this end, we investigate under which conditions it is decidable whether some information can be derived from the raw data by sequences of data transformations. The actual algorithm treats the transformation procedures as building blocks in a directed graph of possible transformation sequences. We show how this graph can be generated from a schematic description of the data transformation and the resulting change of information associated with the data taking place in each transformation procedure. In particular, we claim the following contributions, along which we also structure the discussion below:

1. We prove that the problem whether it is possible to derive information from raw data is in general undecidable but can be decided for special restrictions which are typically the case for real world applications.
2. We extend this model to visualization and derive the conditions for the ability of a visualization system to present inquired information.
3. We outline an algorithm scheme for the automatic generation of visual analytics pipelines covering the whole span from raw data to visualization.

Related Work

To put our work in context, we rely on a classification of elements of visualization theory [16]. The authors describe three branches of possible foundations of a theory of information visualization. In particular, they differentiate between three categories: A data-centric predictive theory allowing to characterize visualization by how well they are suited to present different types of data, an approach based on information theory concerned with visualization content and its communication, and two kinds of descriptive formal models describing the user's interaction and a constructive model of visualization design. Indeed, these three categories apply quite well to classify visualization theory which we show by applying it to describe the general context of the other related work.

What we present in this paper remarkably falls between the categories. Our findings are based on a derivation structure transforming raw data gradually into a visualization and therefore clearly being a constructive model. However, the findings

we make when applying this technique are not about visualization design but about visualization content, interestingly deriving the information content from a data-centric formalism defining information as interpretations of data. We also propose to let the user ask for data representations based on the properties of available visualization techniques – a data-centric approach steering the composition of visualizations by the expected quality of their elements. As a side node, we conclude that maybe these fields are not quite as different as expected.

Taking a closer look at automatic generation of visualizations, we of course have to mention Jock Mackinlay's seminal work on a presentation toolkit [12]. Mackinlay was among the first researchers to interpret visualization systems as formal languages generated by grammars. Other authors followed this idea, although focusing on various different aspects, like covering different types of data [22], a strong focus on tasks [4, 17], a focus on the content of linked data [19], or perception-oriented embedding [6], to mention only a few examples. Our aim is not to reinvent the wheel. Instead, we are interested in how to combine models for automatic data analysis and visualization design to deliver optimal data representations to the user. All of the models listed here are of a constructive nature, describing the structure of visualization. All of them can be combined with our method to cover different aspects of the properties the user needs the representation to have – if we assume we have a set of visualization techniques available that have been evaluated on the data-centric level beforehand, so we can make the proper quality predictions.

Indeed such systems exist. VisIRR, for example, is a semiautomatic visualization toolkit for information retrieval [5]. Being heavily data-centric, it lets the user query for data and propose visualizations hat are predicted to provide good quality presentations. A similar approach, although more on the construction side, is followed by the idea of a form-semantics-function coming from the field of visual data mining [18]. Here, the composition of visualization systems is derived from the semantics of the data to be visualized. This approach is quite appealing as it attempts to steer the visualizations shape by the information it is meant to convey. Again, our intention is not to automatically define the visualization but the whole data preparation and processing pipeline. However, we also intend to apply information models to steer this process.

From this content-oriented point of view, interesting work has been done composing visualizations by the semantics of the visualization's elements rather than the data to be visualized [13]. Very appealing is the idea to let the user define search queries based on examples. By the focus on semantics, the user can specify a query based on content based similarity, i.e. a user can literally ask for "something like this" and point at some data. Perhaps the most popular technology in this domain is the semantic web [1] along with its data exchange format RDF [10], providing a simple, graph based representation of basic facts about the relations between data. An example for a more sophisticated formalisms for ontologies based on the semantic web is the web ontology language (OWL) [11]. From the perspective of our work, the semantic web would be a well-suited candidate to represent the information associated with data and algorithms. Leveraging a combination of SPARQL [15], a query-language for the semantic web based on pattern matching in RDF-graphs, and VOWL [14], an interactive visualization for OWL-ontologies, we may be able to interactively navigate the space of investigable information during explorative analysis. Our focus, however, is not on querying for existing data and semantics but for information supported by data that can be algorithmically derived from the data we have. In essence, we do not have the data we ask for, we only have some tools at hand that might be helpful in getting it. Now, our question is whether the toolset we have is capable of providing us with the information we seek. Our intention is to find ways to automatically provide views on the data of which we know that if the information we seek can be derived from the data, it must be contained in the visualization we show the user. Therefore, inference tools from the semantic web or other domains are not directly applicable to the question we ask in this paper.

Since we are interested in the design of visual analysis pipelines, we need to investigate their shape. Fortunately, there is a recent well-written survey on the development of visual analytics pipelines that up to specializations, Daniel Keim's original model of the visual analytics pipeline is well established [21, 8]. We therefore opt to follow the same approach in the principal construction, visualizing data either directly or after being transformed by data mining, or machine learning procedures. We deliberately allow machine learning methods to be incorporated due to its great potential and increasing influence in the field which is well described in a recent survey [7]. Concerning the definition of data transformation pipelines to be executed prior to analysis, graphical languages like the one offered by KNIME [2] have been developed to control the process. Such languages can be applied by the user to steer or correct the automatic approach we propose if necessary. An example workflow using some of the tools mentioned here could be to define the transformation paths in a graphical language like the one provided by KNIME and load the result into AutoVis [22]. Once set up, the pipeline is executed automatically. Yet, there still is some space left for further automatization. Our algorithm for automatic pipeline generation would attempt to derive a complete cover of the information inquired by the user as some formal representation of the analysis goals.

Data and Information

In this section, we explain the basic concepts we apply in this paper with a focus on the connection between data and information. When speaking about **data**, we refer to qualitative or quantitative variables which we assume can be obtained through measurement or otherwise provided. We distinguish between raw and refined data, the former being the data provided "as is", the latter the result of data transformations. In general, we will assume the data to be related to an **observation** in the sense that the data is *about* the observation. **Information** is an interpretation of data, typically but not necessarily with respect to the observation. We assume information to be a composition of *atomic facts* that can be collected in sets. We also demand the atomic facts to be comparable within the same set of information, i.e. it is decidable whether two atomic facts are equal. The representation of information is often based on logics, predicate logic to be precise. However, the details are domain- and application specific. For example, modal logic can be applied to model systems with several alternatives and (linear) temporal logic allows to model processes. Another way to represent information is the semantic web [1], where data semantics are modeled in terms of a graph given as a set of triples encoding edges between graph nodes. To

establish a relationship between data and information, we further require the existence of two partial maps establishing that connection. Given a set of data $\mathbb{D}$ and a set of information $\mathbb{I}$, we define the **interpretation** $\Delta : D \subseteq \mathbb{D} \to I \subseteq \mathbb{I}$ as a partial function from the data to the atomic elements in the information and the **anchoring** $\alpha : I \subseteq \mathbb{I} \to D \subseteq \mathbb{D}$ as the partial inverse of Δ. We say that some data $D \subseteq \mathbb{D}$ **supports** some information ι if $\iota \in \Delta(D) \subseteq \mathbb{I}$. From here on, we refer to $\mathbb{D}$ and $\mathbb{I}$ as global objects demanding that they contain all data and all information that is either directly available or can be computed or otherwise obtained from other data and information. In contrast, D and I refer to sets of known data and information.

For the following discussion, we do not need a more thorough specification in the definitions of data and information. These details are application specific whereas the results we are interested in are of a more general nature. Hence, for the sake of generality, we do not make further assumptions to the structure of data and information. There is, however, one last concept we need to introduce. The question of containment is a natural consequence of organizing information in sets. For finite sets, decidability of set containment follows trivially since we required the atomic elements of information to be comparable and thus can be checked for equality. Concerning the possibility to derive certain information from the raw data, the situation is quite a bit more complex: We have not defined a map between sets of information so far and refrain to do so in order to preserve generality and allow a wide range of information models. Instead, we apply an indirect approach. Let $\tau : \delta_0 \to \delta_1$ be a transformation transforming some data $\delta_0 \in \mathbb{D}$ into some data $\delta_1 \in \mathbb{D}$. The change of data resulting from the application of τ can result in a change of the associated information – although this is not necessarily the case. We can thus define an indirect mechanism for information transformation by following the data: Let ι_0 and ι_1 be the information associated with δ_0 and δ_1 respectively. Since in this case the relation between data and information is explicit, we know that the anchoring and interpretation operations are defined in both directions and cover the respective sets completely as inverses. Therefore, we can bind the transformation $\sigma : \iota_0 \to \iota_1$ of information $\iota_0 \in \mathbb{I}$ into $\iota_1 \in \mathbb{I}$ to the transformation of data and express it indirectly by $\sigma(\iota_0) = \Delta(\tau(\alpha(\iota_0)))$. Since data transformations (if applicable to the data) can be chained, we can now define **derivability of information** by the existence of a sequence of data transformations from the source data to some data whose interpretation covers the target information. That is, information ι is derivable from some data if and only if there is some transformation $\sigma^* := \iota_0 \to^* \iota$, such that there is a sequence of data transformations $\tau^* \; \alpha(\iota_0) \to^* \alpha(\iota)$, i.e. we have $\exists \tau^* . \iota = \Delta(\tau^*(\alpha(\iota_0)))$. Extending this notion to the derivability of information I from some data D, we obtain that I is derivable from D if and only if for the elements $\delta_k \in D$, we have

$$\exists \Sigma . \big(\forall \delta_k \in D . \big(\exists \sigma_k^* . \big(\sigma_k^*(\iota_0) = \Delta(\delta_k) \big) \big) \big)$$

For single elements, data transformations and σ^* are transitive, the latter being a direct consequence of the former. The *upward closure of contained information* $I^\uparrow$ is thus all information that is directly assigned with or can be derived from some data D, we know that for some information $I = \Delta(D)$, $I^\uparrow$ is precisely the transitive hull of the information associated with the data along every transformation path starting in $D = \alpha(I)$. By the construction, some information ι is derivable from I if and only if $\iota \in I^\uparrow$. Of course, the question remains whether containment in $I^\uparrow$ is actually decidable. The answer to this question determines under which conditions we can prove information to be derivable from raw data and thus available for visualization.

Derivability of Information

The idea of a visualization system in which we can prove the containment of information is quite appealing. Not only would it allow to infer the completeness and correctness of depicted information for solving a given task but it could also enable the user to specify information patterns that would be derived automatically by the system. Unfortunately, it turns out that information derivability is, in general, undecidable. However, there are certain special cases where it is indeed possible to prove information derivability – and even to do so automatically. In the following, we discuss the details about these observations.

Foundational Considerations on Decidability

In principle, the proof that information can be derived from some data is simple. One may just stop searching if a solution is found. Unfortunately, it is in general not possible to prove that some information cannot be derived from a data set. Intuitively speaking, if the inquired information cannot be derived from the data, a data analyst blessed with infinite creativity could literally spend eternity trying to figure out a solution with no way to prove that the effort is ultimately futile.

Theorem 1: Information derivability is undecidable.

Information I is derivable from some data D if there is a continuous path of transformations linking the data of the source with some data whose interpretation function supports I. If our intention is to check algorithmically whether this is the case, allowing arbitrary transformations to be applied would allow us to apply a potentially infinite set of operations to assess whether I can be derived directly from the raw data. Hence, the general problem of deciding whether information is derivable is an instance of the halting problem and thus undecidable [20]. □

Still, we can at least achieve partial success. If an algorithm finds a transformation path generating data that supports the inquired information, it terminates and returns a correct result.

Lemma 2: Derivability is positively semi-decidable.

For the proof, consider an algorithm that simply tries every possible transformation and checks whether the result supports the inquired information. Since information containment is decidable, a positive result will be identified correctly once it is found. If the information is derivable, such a result exists and the algorithm will eventually find and correctly return it. □

Restricting the problem to a limited set of finitely many transformations, things become a lot easier. Since we have only finitely many decisions which transformation to apply to each set of data, the tree of all transformation sequences can only branch finitely in any node. By König's Lemma [9], this tells us that if the tree should be infinite, there must be a branch of infinite depth. However, along such an infinite path, at least one transformation

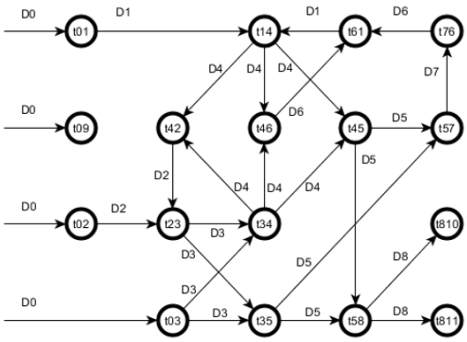

Figure 1. *Automaton of transformation sequences. This finite state automaton links the applicable transformations t by the data formats they exchange when being executed sequentially. Every state is an accepting state. Thus, given any sequence of data formats recognized by the automaton, the sequence of transformations generating the last data entry can be read from the nodes along the path.*

has to occur twice because the number of available transformations is less than the path length. In fact, by the same argument, at least one transformation has to occur infinitely often. The execution chains of transformations define a regular language. Assuming we know the possible combinations, we can easily construct an automaton linking each transformation to its possible successors. Every state in this automaton is accepting. An example of such an automaton is shown in Figure 1. In automata theory, there is a construction extending regular language to words of infinite length called a Büchi Automaton [3]. It recognizes words that pass a certain state in the automaton infinitely often. For the transformation sequences, this means that some transformation occurs infinitely often which is the case if and only if we have an infinite transformation chain. Hence, Büchi automata decide whether infinite paths occur. From here, we obtain:

Lemma 3: The following properties hold for the restricted problem of information derivability with only a limited number of applicable transformations:

1. It is decidable whether transformation chains of infinite length can occur.
2. If no transformation chains of infinite length are possible, information derivability is decidable.
3. If such chains can occur, the problem is still semi-decidable.

The third proposition is probably the easiest to prove since it follows trivially from the semi-decidability of the general problem. Where infinite sequences are concerned, the possibility of such a chain does not even require the full power of the Büchi Automaton. Because we are only interested in the existence, it suffices to check the finite state automaton of transformation sequences for the presence of loops. If no loops occur, the automaton is a directed acyclic graph which we can turn into a tree by separating joins of paths into different branches. Since we only have chains of finite length and only a finite number of transformations to branch in each node, by König's Lemma, the tree must be finite. If so, an algorithm simply needs to follow all transition chains and check whether the data corresponding to the respective nodes supports the inquired information which is required to be decidable by definition. □

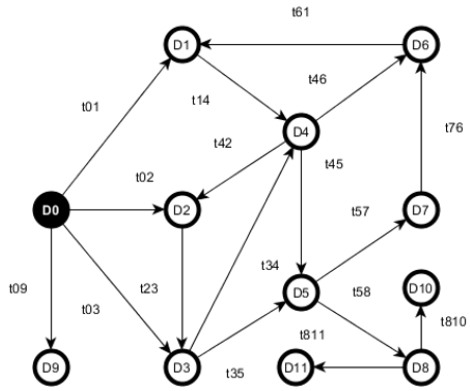

Figure 2. *The transformation graph. Black nodes indicate starting states. Every state is accepting. This automaton provides the data and therefore the information passed when executing a sequence of transformations. The data transformation or information derivation graph is the dual of the automaton of transformation sequences with the additional condition the state's data format must now be compatible with the transformation's for the edge to be allowed in the graph. Data D and, by the interpretation function, information $\Delta(D)$ are derivable from the raw data $D0$ if and only if a path in this graph connects the raw data node $D0$ with the node for D.*

As it turns out, not allowing any cycles at all is restricting the problem a little too far. Recall that in the definition we bind transformations directly to the data they are applicable to. Let us integrate this restriction into our language of possible transformation chains. The resulting automaton is sketched in Figure 2. Loops can now only occur if the data admits it. At a first glance, this renders the problem harder since some of the infinite transformation chains that before necessarily were loops are now open paths. However, we can still detect these open paths in the non-constrained automaton. Considering that the description of a loop is a finite sequence of transformations, we can even distinguish them from the ones in the constrained construction by comparing the transformation sequences. From here, we only need to require that cycles must be compatible with the data.

Theorem 4: If, in a graph of transformation sequences, the applicability of a transformation is determined by compatibility with the data in the source and target of a transformation, infinite sequences are either open paths or closed loops. For graphs with no infinite open paths, the restricted information derivability problem is decidable.

For the proof, it suffices to show that we can contract cycles into a single state merging all the cycle's states into a single one that is also the combined source for all transformations leaving the original cycle and the target for all transformations reaching any node in the original cycle. To see this, consider an arbitrary node in the cycle, let D be its data and I its associated information. Whether some Information J is derivable from D can be assessed by attempting to find J in $I^{\uparrow}$, the upward closure of I, containing all information derivable from D. Obviously, every node in the cycle is reachable via transformations starting at D. Therefore, $I^{\uparrow}$ contains the information of every data node in the cycle. Indeed, it even contains every upward closure of these information sets.

Since this applies to every node in the cycle, $I^{\uparrow}$ is identical for all of them. Therefore, we can contract the cycle into a single node that serves as a unified source and target for all transformations entering or leaving the original cycle and represent the contained information by either computing the upward closure of the information contained in any of the cycle nodes or by computing the union $I^C \subseteq I^{\uparrow}$ of the information sets of the nodes along the original cycle. From the latter, the complete upward closures can be computed by following the original cycle's outgoing edges. □

Theorem 4 is indeed quite remarkable because we can still cover these cases if we limit the length of transformation sequences after contraction of every cycle in the graph. The idea behind contracting cycles is illustrated in Figure 3. Within an upper bound to the length of transformation sequences, the derivability problem is decidable with respect to the bound. We can thus restrict the further consideration accordingly. In the next section, we develop an automatic solution to assess derivability of information and return proper transformation chains to compute the data supporting this information.

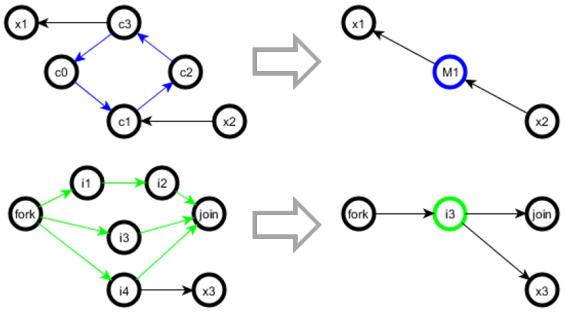

Figure 3. *Contraction of cycles (upper) and fork-chain-structures (lower). Cycles are contracted into a single node holding their internal structure and their individual knowledge and access to transformations internally. The same applies to fork-join-structures with the essential difference that the actual fork and join nodes remain unchanged. Figure 4 shows a complete run of the contraction procedure on an artificial example.*

Automatic Extraction of Information

Our aim is to find an algorithm enabling the automatic extraction of information from available raw data. To this end, it has to assess whether the inquired information can be derived from the data and to transform the data into a form supporting the information. Having observed that this is decidable if the maximum number of consecutive transformations and there is only a finite number of transformations available for application. For real-world applications, both requirements can safely assumed to be met. Applying too many transformations to the data is infeasible, especially for large data sets and the number of applicable transformations is limited by the functionality offered by the analysis software. Even if additional custom algorithms can be implemented on demand, the number of applicable transformations is still finite for any given point in time. However, we want to include iterative procedures converging towards asymptotic results, e.g. as a solution towards optimization problems. The procedures executed along an iteration define a loop. If the information to be represented by the data is chosen properly, the change of data along the iteration does not affect the information. In this case, the upward closure $I^{\uparrow}$ of the information associated with any state of the data along such an iteration cycle is finite. Recall that by I, we denote a subset of known interpretations which in real-world applications would have to be provided by some ontology and typically have to be mapped to the data a priori.

In this work, we assume that the partial map $\Delta : \mathbb{D} \to \mathbb{I}$ interpreting any data we may find during the process to some subset in the possibly infinite set $\mathbb{I}$ of information hypothetically derivable from $\mathbb{D}$, is already defined for the data states we know. We further assume that a set of transformations $T \subseteq \{\tau | \tau := \mathbb{D} \to \mathbb{D}\}$ linking differently formatted sets of data is known beforehand. T is a finite subset of the set of all hypothetically possible data transformations on $\mathbb{D}$. T induces a graph $G = (D^{\uparrow} \subseteq \mathbb{D}, T)$, where $D^{\uparrow}$ is the set of all results of transformation sequences $\tau^* \subseteq T$ applied to a set D_0 of raw data that is to be analyzed. If we restrict the length of the sequences τ^* to some $k \in \mathbb{N}$ and contract the cycles in G as described in the proof of Theorem 4, it is decidable whether some information can be derived from the raw data within an upper bound of k steps. In the following, we introduce a simplification procedure enabling efficient algorithms to assess derivability and infer proper transformation sequences.

Simplification by Contraction

In the proof for Theorem 4, we have already seen a simplification procedure contracting cycles in the graph into a single node representing all the contained states and associated information. Of course, when we apply this type of contraction, we need to keep track of the transformation paths within the cycle. However, if we label each transformation with a unique identifier, this is trivial to achieve. The procedure itself is illustrated in Figure 4. Once all cycles have been removed from the graph, there are two other useful contraction mechanisms. The second step is to contract forks and joins of paths into a single node given that all transformation paths branching from a given node (the fork) meet in the same target node (the join). Fork-join-structures can easily be assessed by applying cycle detection interpreting the graph's edges as undirected after all cycles in the directed graph have been contracted. Any cycle detected this way is a candidate for being part of a fork-join-structure. Of course, one still has to check whether all branches actually meet the join node. Therefore, it makes sense to start with small contract fork-join-structures and gradually increase the size during contraction. By reducing the branching within nested fork-join-structures, this also reduces the task's overall complexity. The result of applying these steps is a tree rooted in a single starting node representing the raw data. Note that a simple union of data structures into a single object can create this node in the case the raw data stems from multiple sources. Therefore, we can assume the raw data to always be represented by a single data node. The last step, although optional, is to contract the simple paths between branching nodes. Since we restricted the interpretation map Δ to map the data D_k represented by any node to finite sets I_k of information, the unions $I_k^C \subseteq I_k^{\uparrow}$ of information sets computed as part of the contraction procedures are also finite sets. Therefore, information containment is decidable and we can assess derivability by traversing the resulting tree and checking containment for the information sets associated with each individual node. Performing the assessment for each element of a set of inquired information also directly provides the percentage of how much inquired information can indeed be derived from

the data.

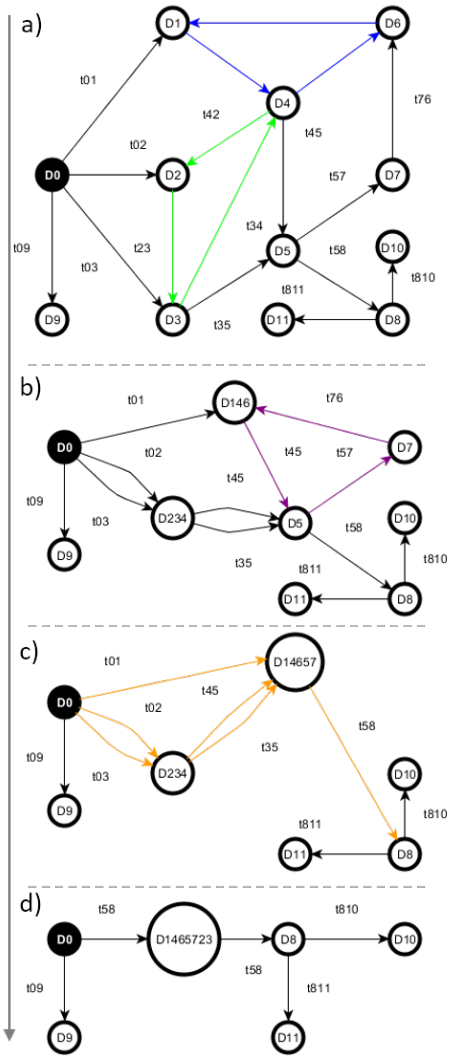

Figure 4. *Example of a contraction procedure applied to a transformation graph. Starting with the smaller cycles the algorithm first contracts two cycles in the initial graph (a), followed by a third cycle containing one of the previously contracted nodes (b). Now that no cycles are found anymore, a fork-join-structure is identified and merged (c). The result of the contraction procedure (d) is a directed acyclic graph of nodes that cannot be further contracted by either of the definitions provided in Figure 3.*

Assessing Derivability

In principle we can directly assess information derivability from the upward closure $I_0^{\uparrow} = \bigcup_{D_k \in D_0^{\uparrow}} \Delta(D_k)$ of the information contained in the raw data. However, in order to obtain an efficient procedure to obtain transformation sequences to be applied to transform the data into a form supporting the inquired information, we should keep track of the applicable transformations for each node in the original graph $G = (D,T)$ of data and transformations. This is exactly what we do during contraction.
When contracting data and computing the union I^C of information assigned to the corresponding nodes in the original graph, we can associate each subset I that has been merged into I^C with the transformation path needed to derive it from some starting point in the data. For cycles, this set is the same for every node and since for every node D_k in the cycle, all nodes are in $D_k^{\uparrow}$, we can just pick any node as the starting node. For fork-join-structures, the starting point is he fork, and for simple paths, the starting point is the path's first point. After the simple paths have been contracted, we have a hierarchy of derived information. The edges are labelled with transformation sequences and the data is associated with tuples of information and the further transformations that need to be applied to obtain this information. Instead of collapsing this tree, we directly apply it as the data structure in which we infer information containment. If we implement the sets of information as lists, we can connect the list of each remaining data node, providing us with one large set of information which is exactly $I_0^{\uparrow}$, the set of all information derivable (within k steps) from the raw data. Assessing the derivability of information from the raw data with respect to the available algorithms therefore reduces to checking whether the inquired information is contained in $I_0^{\uparrow}$. Since we still have the tree structure, we can infer the transformation paths by following the tree's edges backwards until the root is reached once the information has been found. The actual transformation sequence is then a concatenation of the tree's edge labels. The remaining transformations to be applied to obtain the actual shape as it was prior to contraction are stored in each node along with the corresponding set of data.

The algorithm terminates and returns the correct information and transformation if a solution is found. If not, the algorithm also terminates since we restricted the length of transformation paths to be checked. However, the result obtained in general is only whether it was possible to derive the information from the raw data in up to k transformation steps. A negative answer is reliably correct if and only if the original structure does not permit infinite chains of consecutively applied transformations. Fortunately, this is decidable since as we have seen in the discussion of Lemma 3, such infinite paths map to infinite words in a regular language and can thus be recognized by a Büchi Automaton. Therefore, if we encountered such a situation, we need to inform the user about the unreliability of a negative result.

Considerations on Runtime Complexity

Where runtime complexity is concerned, we need to distinguish three categories of algorithms. The first one is the computation of the actual transformations. This is highly specific to the data and the applied algorithm and thus not in the scope of this work. It should be noted, however, that the procedure can be rather time consuming, especially if translations between data structures have to be applied before the subsequent algorithm can be executed.

The second category is the assessment of derivability and the inference of the corresponding transformation path. Finding the correct path actually depends on the length of the longest branch in the tree obtained from the contraction procedures. Considering that before contraction the maximum path length was bounded either by a number $k \in \mathbb{N}$ or, in presence of infinite open paths, limited to k, this is an upper bound for the tree depth. Since the format of information and therefore the equals-relation depends on the application, we cannot make any assumptions on the run-

time needed to compare two sets of information and just assign it the funtion $\mathscr{R}$. In the worst case, every transformation is applicable to every data set and yields data with globally unique information. In this case, the derivation graph with respect to n applicable transformations becomes an n-ary tree of depth k. We observe that even for comparably small chains of applicable transformations the runtime complexity skyrockets if we provide powerful toolsets for the analysis. The worst case runtime complexity for assessing information derivability is $\mathscr{O}(k^n \cdot \mathscr{R})$. Even for only 10 algorithms and an upper bound of five consecutive transformations, even if we search only one information item and the largest set contains only five items, this amounts to the comparison of 500,000 individual information elements. As it seems, the worst case upper bound quickly skyrockets to ridiculous amounts of runtime complexity even for comparably simple examples. However, this is unlikely to happen, especially since we are concerned with visualization. While there are algorithms like sorting and searching that may be almost universally applicable, the majority of the algorithms is not. For example, in a visualization system, a wide variety of visualization techniques may be offered. However, these options are applied only once, drastically reducing te complexity. Let us assume that our hypothetical example is a classification task offering two filters for outlier detection as preprocessing, two distance measures, two classification techniques, one color map showing the ground truth, and three different visualization techniques. To solve the classification task, data must be preprocessed, compared, classified, colored, and rendered to the screen – a total of five steps. Counting the possibilities, the five consecutive computations yield 24 different states for the worst case runtime. Note that we still have a maximum path length of five and ten applicable transformations. If we again assume the user to compare one information item to sets of five items, we need to perform only 100 comparisons.

From the considerations on the hypothetical visualization task, we conjecture that in real world applications, we can safely assume the derivation graph to be rather sparse and the upper bound of runtime complexity to be far from being met in actual applications. Note that this claim is more bold than it may appear at a first glance. While, for example, a human programmer will usually not invoke a sequence of several sorting algorithms immediately after each other, a computer only looking at the compatibility of data formats will definitely consider this a valid option. Therefore, the information about what kind of transformations should be combined with each other should be added to the system, probably as part of the information associated with the transformation itself. From the hypothetical example we learn that grouping of the algorithms into steps and implementing rules for their proper combination can already reduce the theoretically possible complexity tremendously.

The last category are the algorithms for contraction. Contraction consists of three steps: contracting cycles in the directed graph, contracting fork-join-structures, and contracting the remaining simple paths. The runtime complexity of connecting the linked storing of the remaining information sets is negligible since connecting to linked lists can be performed in $\mathscr{O}(1)$ and the other steps' complexity is more than linear in the number of graph nodes. Towards the detection of cycles in the graph G, recall that the graph actually models $D_0^{\uparrow}$, i.e. all data sets D that can be computed from the raw data D_0 by transformations $\tau \in T$. We can therefore detect cycles by computing the topological ordering of the nodes starting in the node representing the raw data D_0. If the derivation graph is not acyclic, the algorithm will eventually detect a cycle to be contracted. Since the topological order can be established using depth-first search (DFS) , it can be established in $\mathscr{O}(|D_0^{\uparrow}| + |T|)$, i.e. the computation is linear in the number of graph node and edges. Note that if the graph contains cycles, the number of edges can be significantly larger than for an acyclic graph. Note that in general, $|D_0^{\uparrow}| + |T| \ll k^n$. Because the actual sparsity of the derivation graph depends on the domain and application, we cannot make any restricting assumptions without sacrificing the generality of our discussion. Since DFS logs visited nodes as part of its execution, the transformation paths describing cycles are obtained together with the cycle. Hence, contracting a cycle is linear in the number of its nodes since the union of their information sets can be computed by concatenating the linked lists storing the sets of information. If a cycle is found, it is immediately contracted and the DFS is continued from the new node, dropping the results of the former search. The procedure is continued until no cycles are left. In the worst case, the data is aligned along a long line where D_0 is a starting node and DFS identifies the first cycle as being the cycle containing the tree's single leaf and its immediate parents. If the new node is again part of a cycle of two nodes, this cycle is also merged. Continuing this procedure up to the root requires as many steps to go down until the leaf is reached as it requires to contract the cycles on its way back up. The worst case is therefore linear in the number of nodes and we obtain an upper bound of $\mathscr{O}(|D_0^{\uparrow}| + |T|)$. For the detection of fork-join-structures, the procedure is similar but we need to either apply breadth first search (BFS) or interpret G to be an undirected graph. While the latter method is again based on cycles and can apply the same algorithm as before, it requires an additional reconstruction step to obtain the actual paths contributing to the structure. BFS not only reveals these paths directly but also covers complex branching situations. The runtime complexity is the same as for the contraction of cycles.

After contraction, we are down at a tree structure again and the remaining upper runtime bound for assessing derivability in the graph is, again overestimated, $\mathscr{O}(k^n \cdot \mathscr{R})$. Contraction thus allows to asses derivability of information more efficiently, speeding up the process especially in applications like business intelligence or process control monitoring.

Automatic Pipelines for Visual Analytics

In 1986, Jock Mackinlay published his seminal paper on the automatic design of visualizations of relational information [12]. He describes data visualizations as the words and sentences of a graphical language describing the behavior of the applied visualization technique in terms of predicate logic. The connection between the elements of the visualization and the data they represent is achieved by an encoding relation. Defining predicates using the encoding relation, a hierarchy can be defined, defining structures like the points in a scatterplot or the arrangement of scatterplots in a matrix. This establishes an implicit connection between graphical elements in the visualization and the information they support. Mackinlay proposes and automatic visualization design process based on the notions of expressiveness and effectiveness. Expressiveness denotes the fact that a graphical language – and therefore

a visualization technique – is available, that *can* depict the inquired information, demanding the existence of a sentence within the language that encodes exactly the data to be visualized. Effectiveness is an empirical value of visualization performance which is applied to select the optimal visualization if among available alternatives. In this section, we extend Mackinlay's technique to an automatic tool for the design of visual analytics pipelines.

Derivation and Inference in Visualizations

If we relax the expressiveness criterion to only demand that the visualization should represent the information $I = \Delta(D)$ associated with data D to be best extent possible and avoid to incorporate more data than necessary, Mackinlay's expressiveness is equivalent to our notion of derivability. We claim, however, that in case we do not have the tools available to extract and visualize the whole information we inquire, we should still try to visualize the information to the best extend possible and to inform the user about what is missing in the visualization. Inspired by Mackinlay's work, we model graphical languages and thus visualizations using our derivation graphs. Technically, describing the visualization as a set of transformations to be applied to data is the same as implicitly defining a graphical language using predicate logic constraints. To establish compliance with our annotation, we extend the *Encodes*-relation to relate an element of the visualization (graphical sentence) not only to the data but also to the information associated with this data. The data to be visualized is the raw data of visualization technique's transformation graph. Because expressiveness determines whether some information can be visualized within a graphical language or not, we can say that the capability of a visualization to depict certain information depends on whether this information is derivable from the raw data and that it is contained in the upward closure of the information directly associated with at least one visualization element. Since we consider visualization systems that only offer finitely many options to represent data and we allow to restrict the number consecutive transformations, this setup satisfies the preconditions for Theorem 4. A direct consequence of this is:

Lemma 5: Let V be a visualization system with k algorithms for data representation or transformation. The ability of V to represent certain information after a sequence of not more than n consecutive data transformations is decidable.

Derivability is a necessary condition for the ability of a visualization system to represent inquired information. However, it is not sufficient since not all derivable information is necessarily encoded by some visualization element that is actually rendered to the screen. There are two cases of such non-explicit information derivability in visualization systems which in this work we refer to as **intrinsic** and **extrinsic inferability**. Intrinsic inferability does not expose the information to the user. Typically the user is shown a consequence of the actual information. A trivial example for this case is a progress bar. Although the system does no expose to the user what action it is currently performing, the consequence, namely the progress made so far, is visualized and provides the user with useful information. Likewise, extrinsic inferability involves the user into the process. This information is indeed encoded by visualization elements but not by the basic ones. Hence, it is not presented explicitly and the user has to infer it from the data context. This is typically the case when the information is encoded by a history of states or by relations between explicitly depicted items. Examples are the trajectory of a driving car or tasks where the user has to infer abstract properties from Gestalt principles.

Interchangable Pipeline Blocks

Now that we know we can seamlessly integrate visualization into the theory developed so far, we are able to define building blocks of dynamically adjustable visual analytics pipelines. If we define transformation graphs for each technique to be applied during data preparation and for data visualization, we can construct pipelines by highlighting not only the raw data nodes but also distinguished output nodes marking data that is exposed to the outside. If we now connect the output of one graph to the raw data input of another, this effectively concatenates the applicable transformation paths and thus merges the information stored in both graphs. For example, we can attach a visualization technique to a data preparation step like Figure 5 demonstrates it for the visualization of a projection obtained from principle components analysis (PCA) as points in a scatterplot.

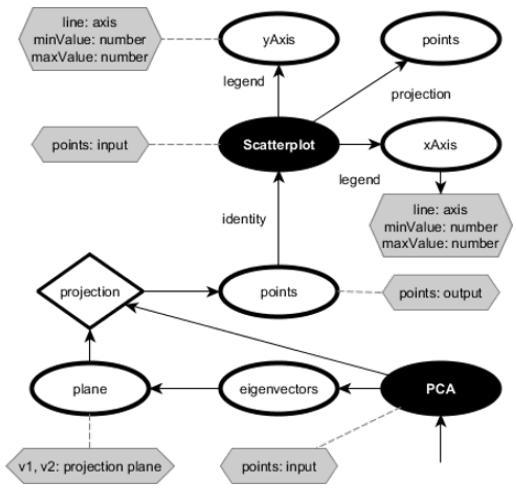

Figure 5. *Results of Principal Components Analysis (PCA), visualized as a scatterplot. Grey fields indicate an informal representation of the interpretation-function where entries are of the format (data:information). The output of PCA is a set of points in a plane which are passed to the scatterplot visualization module. This determines the scatterplot's raw data and thereby applies the visualization to the computed data. The diamond shape is a blank node used as a shortcut for functions taking multiple input parameters.*

Towards more sophisticated analysis, consider an example where an analyst tries to identify and highlight clusters in some multivariate data set. Using parallel coordinates, the analysis expert infers discriminating features based on the distribution of lines along the different axes and projects the data to these dimensions. Using some distance measure, the analyst runs a clustering algorithm, subdividing the data into k subsets in an iterative procedure. Feeding the cluster distribution back into the original data as an additional dimension, a new axis indicates the clusters which proves that in this setup, a parallel coordinate plot is capable of visualizing the inquired information which data item belongs to

which cluster. However, the addition of an additional axis makes it hard to infer the clusters when scrolling sideward through the diagram. Therefore, the analyst chooses to extend the visualization by a color coding for the classes. The whole interaction and the corresponding transformation graphs are shown in Figure 6.

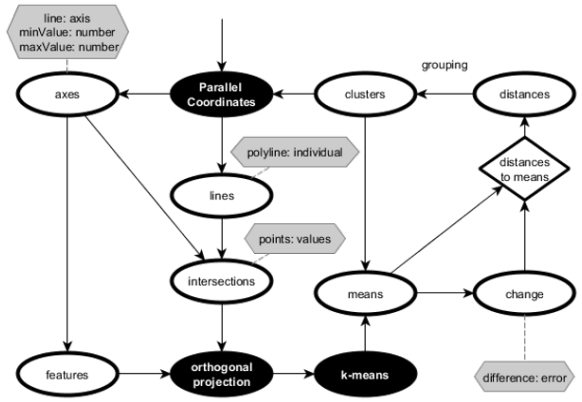

Figure 6. *Parallel Coordinates plots can visualize class membership. To prove that a visualization is possible, it does not suffice to provide the example – one has to prove the information is actually there. Here, we need the plot and an arbitrary clustering method, e.g. k-means. For the proof, we feed the clusters back into the visualization system as part of the raw data. This is equivalent to adding another table entry to the original data and thus interpreted by the plot as an additional axis. With the new axis the visualization supports the inquired information about (computed) class compositions.*

In the example, the algorithms behind the different operations are simplified and much information is hidden in the transformations. However, it would be unsound to model every detail of the procedure, especially the sequential parts. Apart from the fact that this would not provide any further information, the sequences would be consumed by contraction anyway. In the example, we deliberately did not apply contraction yet to show the algorithms in detail. We observe that the whole graph actually consists of four blocks, namely the parallel coordinates as the applied visualization technique, the projection, the clustering procedure, and the color map. These blocks are essentially independent of each other with the only restriction that blocks can only be connected if the output data format of the one block fits the raw data format of the other. We can assume this to be the case for real world applications of visual analytics suites – or achievable through the invocation of translation procedures. For example, there are tools to define data preprocessing pipelines by graphical programming like KNIME [2] in which this obviously has to be the case. Otherwise, the constructed pipeline would not be executable. The major difference between our approach and these tools is that we actually do not intend to ask the user to define the paths manually. Instead, the user asks for nontrivial information patterns which the system extracts and visualizes automatically. Figure 7 shows an example of a more powerful collection of analysis and visualization techniques. Although the collection is still rather small, it can already serve nontrivial requests like the demand for a visualization technique which optimally preserves some feature that is only present in derived data, for example cluster membership or the alignment of scatterplots in a scatterplot matrix. This is possible because the visualization blocks hold information not only about their general properties but also about their applicability and specific advantages.

Automatic Pipeline Generation

Before we define the algorithm, let us summarize the relationships between the previously obtained results and discuss how they enable an automatic generation of visual analytics pipelines which always terminates and provides correct results. We define a result as correct if it either returns that the inquired information is not derivable or it derives the information completely and correctly and presents it in a suitable visualization with respect to the information to be conveyed and the user's additional requirements. In this sense, the algorithm can be understood as an extension of Jock Mackinlay's APT to visual analytics.

Since each block is a set with only finitely many data transformations and only a finite number of blocks is available, the conditions of Theorem 4 apply if we also require a finite upper bound for the length of sequences of connected blocks. Alternatively, we can demand each sequence of blocks to eventually contain a visualization and the computation to stop there until further processing is manually invoked by the user. Since this also guarantees termination of the computation, Theorem 4 still holds. Likewise and derivability of information is decidable. Since this implies that Lemma 5 also holds, the system's ability to visualize information – be it explicitly or by intrinsic inference – is also decidable. Note that extrinsic inference is never decidable since it depends on the user's understanding of the data. The fact that it is possible to infer the correct information does not necessarily prevent misinterpretation. Since functionality can be wrapped into blocks which can be aligned in transformation sequences, we can compute a variety of different views on the data. Theorem 4 and Lemma 5 trivially hold across execution blocks if one chooses to connect the blocks by sets of identity transformations. Since the amount of data is finite, the transformations also terminate. Since every visualization block supports information about its own applicability and quality, the algorithm terminates, is capable of deriving the visualization correctly, and to visualize it with the most proper available method. The final piece in the puzzle is the contraction procedure. Being applied to each block, it ultimately reduces the representation to the form of the system shown in the upper left corner of Figure 7, representing each process by a single node after abstracting away the implementation. Therefore, we can apply existing software suites to actually compute the transformation sequences obtained by the algorithm – as long as we provide correct descriptions of the information being derivable in the respective modules.

For a visualization system with a set of data transformation blocks and a set of visualization blocks, the algorithm first computes the contraction for each of the transformation graphs constituting the several blocks. In a second step, it establishes the links between the blocks by checking for each pair of blocks whether the information in the output nodes is contained in the information of the other block's input node. The algorithm assumes that data formats are compatible if the information matches. The resulting graph is exposed to the user as a means to communicate the transformation paths applied towards visualizing the data. This way, the procedure is transparent of the user who can manually readjust transformation paths if needed. This preprocessing is only needed while setting up the program. As long as no new blocks are added

to the system, its results can actually be saved in a configuration file. The last preparation step is to execute contraction on the graph of blocks as illustrated in Figure 7. Recall that contraction gathers all information in a linked list under a tree of transformation paths connecting contracted data nodes. To infer the correct visualization for some set of information, the algorithm first verifies the derivability of each item in the set using the algorithm outlined in the section on automatic extraction of information. If the item is derivable, the corresponding path is also available. For each of the inferable elements, the corresponding subset of the linked list is scanned for contained information about visualizations applicable to this information. Again, the transformation paths are known since they have been preserved by the contraction procedure. Traversing the tree, the algorithm finds the sections in the linked list of information items that correspond to the respective visualization techniques and traverses them for information to which kind of data they apply best. With this information, a ranking is computed which part of the derived data should be visualized with which technique. Note that if additional criteria on the visualization have been inquired by the user, the corresponding techniques have already been detected as part of the first steps and are ranked higher than the remaining techniques. Note that visualization themselves can also be treated as data allowing for aligned and nested visualizations if the user requires them. Actual implementations of the models for data and information, as well as the ranking procedure are application specific. One solution for the ranking is to apply the effectiveness criterion Mackinlay proposes for APT. Some other applicable data and information models are mentioned in our review of related work.

An Example

For a more intuitive access to how the system works, let us briefly discuss an example. Assume we are investigating the well-known Iris flower data set containing measurements of the petals of 150 Iris flowers of three different species in four dimensions. Let us further assume we are applying the analysis system shown in Figure 7. As a preprocessing step, the system's graph would be collapsed just like it is shown in the figure. A first query to the analysis system could now be to ask for *a single-projection overview that minimizes the projection error*. The system recoginizes that it can load data, does not need to apply any preprocessing and recognized that among the informaton associated with PCA as a projection techniques, there is an entry formalizing that PCA optimizes variance and minimizes the projection error. Since the output of PCA is data points, it would then opt for the scatterplot as the visualization technique of choice. Since the collapsing collects the transfomation paths and stored them for each collapsed cell, the computer can retrieve the transformation algorithms directly. The path obtained by connecting the blocks would this be exactly the graph depicted in Figure 5. From this overview we would be interested in *investigating local neighborhoods of similar items*. Going through the collapsed graph immediately reveals that we are capable of providing this information because we do have neighborhood graphs in our toolset and node-link diagrams have the information associated that they can show neighborhoods. The computer selects to compute the k nearest neighbors for each node since we asked for local neighborhoods and applies the scatterplot from before to position the graph's nodes. From this image, we get the impression that there must be three groups among the flowers. We thus ask to *somehow group the flowers by their similarity*. Clustering algorithms are associated with the information that they group elements, so the computer attempts to find a suitable path through the derivation graph containing a clustering algorithm. k-means is chosen since it is similarity-based and locally. As a result, the flowers are tagged with their class names and the tags are passed to the scatterplot together with a colormap for the groups. We thus identified the three groups of flowers in the data set. With a more sophisticated system than the one we apply in this szenario, we could for example ask to *refine the clustering based on local cluster outliers*. Given the necessary information in some nodes, the system could derive strategies like filtering outliers in each single cluster and attempt to rematch them to the cluster they best in the best.

Discussion and Future Work

We proposed a general model for the derivation of information in terms of interpreted data and investigated the conditions under which the derivation process can be automatized. The obtained decidability results motivate the outline of an algorithm to he automatic derivation and visualization of data with respect to inquired information. Being independent from specific data and information models, the technique is generally applicable. However, this comes at the cost of a certain vagueness in the descriptions of the algorithms whenever the data structure or the organization of information becomes important for the discussion.

We organize the execution of a sequence of data transformations in combinable building blocks. Each of them is associated with its own information, allowing the system to leverage this information to derive insights about the data. Again, keeping the system general means that this information is not data-specific. Therefore, in a plug-and-play scenario where some data is passed to an analyst who then intends to use our system for the investigation, only general information about the respective algorithm's effect on the data can be inquired. In contrast, data specific information can be added in analysis environments where the information associated with data does not change frequently, allowing for more sophisticated information inquiries tailored to the specific application. In this work, we avoided the necessity to add additional information directly to the data since this would require an information model and thus limit the generality of our approach. However, the potential of interactive feedback of analysis results into the data and transformation model should not be underestimated. For example, reintegration of information can help to iteratively optimize information derivation procedures or to quickly reproduce analysis results on other data.

Probably the most important problem of keeping the procedure general is that, as of now, we can provide a detailed interaction scheme for the user to invoke the pipeline generation by inquiring information. In fact, since the information model depends on the application, the ways to query for information will differ among different implementations. However, due to its descriptive power, we conjecture that the semantic web will be a good candidate to model the information in many applications.

One weak point of automatic visualization is the ranking of the different methods to be applied for different tasks. It could be expected this to become even worse if the automatization is extended to the whole pipeline, for example when multiple algorithms for clustering or classification are applicable to the same

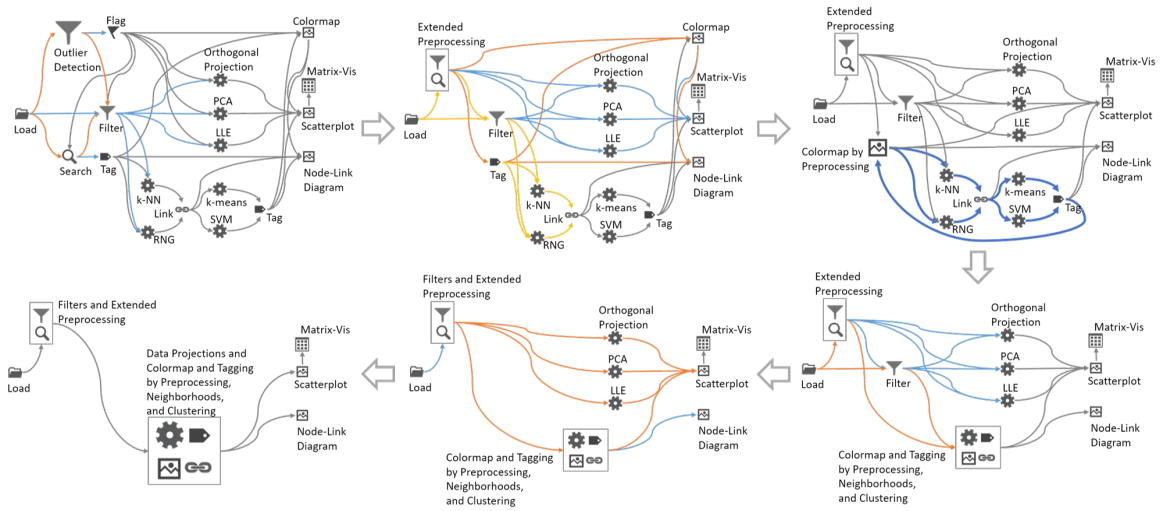

Figure 7. *Collapsing applied to a set of transformation and visualization algorithms. To indicate that the algorithms have already been contracted and to emphasize their role in the visualization system, we apply different icons for their visualization. The diagram is to be read clockwise; the entries to the left are the original state and the final state when no further contraction is possible. Determination of fork-join-structures is based on a breadth first search starting in a given center. Blue edges denote states already covered but without success, yellow denotes paths of the graph that are candidates to which contraction has not yet been applied due to the orange structures containing less nodes or being closer to the original data. Whenever something changes, the algorithm starts to check the corresponding entries.*

data. Fortunately, this is only partially true. Indeed, problems occur wherever a ranking of the performance of methods is involved. However, the mathematical toolset applied for automatic analysis and data preparation can be measured or at least compared relative to the individual algorithms' performance on representative test data whereas the performance of visualization has to be evaluated based on human perception. We conclude that in most applications the uncertainty bottleneck in finding optimal solutions will remain with the identification of the proper visualization technique. In the future, models will have to be found that allow to integrate the insights obtained from evaluating visualization techniques into the information associated with the algorithms. Instead of having to inquire the optimization of statistical or numerical properties like "the least amount of clutter" or "best preservation of distances", a user of a system equipped with this information could literally ask for "the best view on my data".

Making the final transformation procedure transparent to the user is crucial for the reproducibility of results, especially because it explains the visualization. We establish this documentation of the program's performance by presenting the graph of linked blocks to the user. The graph also serves as an interface for interaction and exploration, allowing to recursively unfold or collapse the nodes to investigate the transformation procedures they represent and the transformation sequences chosen by the algorithm. Interaction for editing or extending transformation sequences allow the user to improve and extend the presentation if needed.

There are some open questions left where the implementation of the algorithm for automatic information derivation is concerned. However, one could also replace the automatic approach by a semiautomatic one, where the computer only proposes viable transformation paths rather than deciding between them. Because the optimal solution depends partly on the user, a semiautomatic implementation letting the user decide between the alternatives allows to avoid the necessity to provide measures and rankings for optimal data representation. On the other side, the fully automated approach is easier to use, especially for novice users. Our main concern to propose the fully automatic algorithm was to show that it is actually possible to define a structure of data and its associated information in which it is – even if further restrictions have to apply – such that derivability and the ability to visualize certain facts are decidable. This possibility yields that there is a formalism in which every visualization can be expressed (by its pipeline) and in which it is decidable or at least positively semi-decidable for visualizations whether the information they are intended to convey is derivable and can be visualized. The implication here is clear: Given formal models for data and for the associated information, we can prove the correctness for visualizations.

Conclusion

In this paper, we prove the feasibility of an automatic generation of data processing and visualization pipelines for visual analytics based on a user's query for information being contained in the data and provided by the visualization. To this end, we investigate the problem of deriving information from raw data. Our assumption is that (known) information is given in terms of a (partial) interpretation function and can be derived from raw data if the raw data can be transformed such that the inquired information is in the interpretations' range. The resulting reachability problem can be solved for almost all real world applications. Based on these considerations, we outline a schematic for an algorithm for the automatic generation of visual analytics pipelines. Our results are of a general nature, leaving the definitions of the applied models for information and data to the implementation. Especially in systems offering large collections of data transformations and visualization techniques, generating high-quality visualizations optimized towards the conveyance of specific information will benefit substantially from automatic data processing pipeline generation.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web (Berners-Lee et. al 2001). 2001.

[2] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. *KNIME: The Konstanz Information Miner*, pages 319–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[3] J. R. Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[4] S. M. Casner. Task-analytic approach to the automated design of graphic presentations. *ACM Trans. Graph.*, 10(2):111–151, 1991.

[5] J. Choo, C. Lee, E. Clarkson, Z. Liu, H. Lee, D. Horng, Chau, F. Li, R. Kannan, C. D. Stolper, D. Inouye, N. Mehta, H. Ouyang, S. Som, A. Gray, J. Stasko, and H. Park. Visirr: Interactive visual information retrieval and recommendation for large-scale document data. 2013.

[6] C. Demiralp, C. E. Scheidegger, G. L. Kindlmann, D. H. Laidlaw, and J. Heer. Visual embedding: A model for visualization. *IEEE Computer Graphics and Applications*, 34(1):10–15, 2014.

[7] A. Endert, W. Ribarsky, C. Turkay, B. W. Wong, I. Nabney, I. D. Blanco, and F. Rossi. The state of the art in integrating machine learning into visual analytics. *Computer Graphics Forum*, 2017.

[8] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Information visualization. chapter Visual Analytics: Definition, Process, and Challenges, pages 154–175. Springer-Verlag, Berlin, Heidelberg, 2008.

[9] D. König. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Litt. ac. sci. Szeged*, 3:121–130, 1927.

[10] O. Lassila, R. R. Swick, W. Wide, and W. Consortium. Resource description framework (rdf) model and syntax specification, 1998.

[11] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016.

[12] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[13] R. Möller, V. Haarslev, and B. Neumann. Semantics-based information retrieval. In *In Int. Conf. on Information Technology and Knowledge Systems*. Springer Verlag, 1998.

[14] S. Negru, S. Lohmann, and F. Haag. VOWL: Visual Notation for Ontologies (Specification of Version 2.0), 2014.

[15] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.

[16] H. C. Purchase, N. Andrienko, T. J. Jankun-Kelly, and M. Ward. *Theoretical Foundations of Information Visualization*, pages 46–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[17] H. J. Schulz, T. Nocke, M. Heitzler, and H. Schumann. A design space of visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2366–2375, 2013.

[18] S. J. Simoff. *Form-Semantics-Function – A Framework for Designing Visual Data Representations for Visual Data Mining*, pages 30–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[19] K. Thellmann, M. Galkin, F. Orlandi, and S. Auer. *LinkDaViz – Automatic Binding of Linked Data to Visualizations*. 2015.

[20] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[21] X.-M. Wang, T.-Y. Zhang, Y.-X. Ma, J. Xia, and W. Chen. A survey of visual analytic pipelines. *Journal of Computer Science and Technology*, 31(4):787–804, 2016.

[22] G. Wills and L. Wilkinson. Autovis: Automatic visualization. *Information Visualization*, 9(1):47–69, 2010.

Author Biography

Benjamin Karer *received his MS in Computer Science from the University of Kaiserslautern in 2016. Currently he is a PhD candidate in the Computer Graphics & HCI Group at University of Kaiserslautern. His research concentrates on foundational aspects of the encoding of information in data visualizations.*

Inga Scheler *received her Diploma in Civil engineering from the University of Kaiserslautern (1999) and her PhD in computer science from the University of Kaiserslautern (2008). Since then she has worked in the Computer Graphics & HCI Group at University of Kaiserslautern and since 2012 as Managing Director in the Regional University Computing Center at University of Kaiserslautern. Her main focus is on the development of new visualization techniques supporting basic infrastructure systems.*

Hans Hagen *received a BS in Computer Science (1976) and a MS in Mathematics (1979) from the University of Freiburg and his PhD in Mathematics in 1982 in Mathematics (Differential Geometry) from the University of Dortmund. His main research interests are Geometric Modeling and Scientific Visualization. He was awarded the IEEE Visualization Career Award in 2009, the John Gregory Memorial Award in 2002 and the Solid Modeling Pioneer Award in 2016.*