Opening the Black Box: Strategies for Increased User Involvement in Existing Algorithm Implementations

Thomas Mühlbacher, Harald Piringer, Samuel Gratzl, Michael Sedlmair and Marc Streit

Abstract— An increasing number of interactive visualization tools stress the integration with computational software like MATLAB and R to access a variety of proven algorithms. In many cases, however, the algorithms are used as black boxes that run to completion in isolation which contradicts the needs of interactive data exploration. This paper structures, formalizes, and discusses possibilities to enable user involvement in ongoing computations. Based on a structured characterization of needs regarding intermediate feedback and control, the main contribution is a formalization and comparison of strategies for achieving user involvement for algorithms with different characteristics. In the context of integration, we describe considerations for implementing these strategies either as part of the visualization tool or as part of the algorithm, and we identify requirements and guidelines for the design of algorithmic APIs. To assess the practical applicability, we provide a survey of frequently used algorithm implementations within R regarding the fulfillment of these guidelines. While echoing previous calls for analysis modules which support data exploration more directly, we conclude that a range of pragmatic options for enabling user involvement in ongoing computations exists on both the visualization and algorithm side and should be used.

Index Terms—Visual analytics infrastructures, integration, interactive algorithms, user involvement, problem subdivision

1 Introduction

A tight interplay between visualization, interaction, and analytical computation is the core aspect of Visual Analytics [25, 45]. The motivation is to combine cognitive and perceptual capabilities of human analysts with computational capabilities for tasks like statistical modeling, planning, and decision making [43]. In addition to intelligent visualization and interaction concepts, involving the user in the analysis process implies delivering results and visual feedback within at most a few seconds [9] and ideally less than 100 ms [42]. Particularly for large data, this requirement contradicts the computational effort of many advanced algorithms like clustering or dimension reduction.

As a compromise, a growing number of systems apply strategies like early visual feedback of partial results [17, 23], cancellation on arrival of new input [36], or active steering of a computation in progress [10]. In current practice, however, this type of application-specific fine-tuning often involves a reimplementation of algorithms by researchers and practitioners in Visual Analytics [15]. The obvious disadvantages include a sub-optimal use of skills and resources and an explosion of proprietary implementations rather than standardized and tested solutions.

In contrast, existing systems and languages for data analysis have widely been used for a long time and offer a variety of proven algorithms. In fact, an increasing number of academic and commercial visualization tools stress the integration with software like MATLAB and R (e.g., the R integration in Tableau[1] or [38]). Two key goals are to offer the algorithmic functionality within the visualization tool and to increase the acceptance by data analysts who have been working with these script-based environments for years. Packages like RServe[2] or the MATLAB API[3] make the integration reasonably easy from a software engineering point of view. However, as pointed out by Fekete, "*computation and analyses are often seen as black boxes that take tables as input and output, along with set of parameters, and run to completion or error without interruption*" [15]. In general, it is commonly stated in the Visual Analytics community that exploration is not taken into account in most infrastructures for analysis computation [25], explaining "*calls for more research* [...] *on designing analysis modules that can repair computations when data changes, provide continuous feedback during the computation, and be steered by user interaction when possible*" [15].

Motivated by and echoing these calls, we structure requirements and formalize strategies to achieve them. Using this formalization, we argue that a range of possibilities for implementing these strategies already exists based on currently available computation infrastructures. Specifically, the focus of this paper is on studying conceptual possibilities for tightly integrating analytical algorithms of existing computation software into interactive visualization tools. A main goal of the paper is to increase the awareness and the understanding of these possibilities within the Visual Analytics community. Another goal is to improve the understanding of the needs of Visual Analytics applications within communities focusing on algorithm design like Knowledge Discovery and Data Mining. To this extent, the contributions of the paper can be summarized as follows:

- A structured characterization of visual exploration needs concerning user involvement in ongoing computations.
- A formal characterization and comparison of strategies for achieving user involvement in different types of algorithms
- Considerations for implementing these strategies either as part of the visualization tool or as part of the algorithm, including an identification of requirements and guidelines for the design of algorithmic APIs in favor of a tight integration.
- A survey of frequently used algorithms for knowledge extraction and model building of multivariate data regarding the fulfillment of these guidelines as a case study based on the software R.

2 Related Work

Over the last decades, the interplay between computational environments and visualization has been addressed in numerous research papers and commercial systems. On the one hand, visualization systems

- *Thomas Mühlbacher is with VRVis Research Center, Vienna, Austria. E-mail: tm@vrvis.at.*
- *Harald Piringer is with VRVis Research Center, Vienna, Austria. E-mail: hp@vrvis.at.*
- *Samuel Gratzl is with Johannes Kepler University Linz, Austria. E-mail: samuel.gratzl@jku.at.*
- *Michael Sedlmair is with University of Vienna, Austria. E-mail: michael.sedlmair@univie.ac.at.*
- *Marc Streit is with Johannes Kepler University Linz, Austria. E-mail: marc.streit@jku.at.*

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014. Date of publication 11 Aug. 2014; date of current version 9 Nov. 2014. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.
Digital Object Identifier 10.1109/TVCG.2014.2346578

[1]http://www.tableausoftware.com

[2]http://rforge.net/Rserve

[3]http://www.mathworks.com/products/matlab

integrate computational tools to perform calculations, as found in research [24, 38, 47] as well as in commercial products like Tableau, JMP[4], or Spotfire[5]. However, these implementations often boil down to a black box integration that is insufficient for realizing interactive exploration of large datasets as envisioned in this paper. On the other hand, there are graphical libraries developed for extending computational environments by visualization capabilities, such as the GGobi[44] package for R. However, these extensions are usually not designed for dealing with large datasets and do not allow users to actively interact with ongoing computations.

The visualization community has already identified the need for intermediate results, which state-of-the-art computational environments cannot provide in most of the cases. According to Fekete [15], analytical environments are not designed for exploration and algorithm designers often make no effort to provide such early results during computation. In the VisMaster book [25, p. 97f], the authors take the same line by explicitly identifying needs and goals for realizing interactive visual analysis. The major goals are to get fast initial response with progressive refinement, to provide means for triggering recomputation following small changes, and to allow analysts to steer the computation. The work by Fisher et al. [17] confirms the need for early feedback during computations by a user study on incremental visualization. A more general discussion of user involvement in online algorithms together with a description of example implementations has been provided by the CONTROL project [23]. We take this requirements analysis one step further, and provide a detailed discussion of how different types of early information exchange support user involvement in interactive exploration.

User interaction in a more general sense was investigated by Yi et al. [52], who proposed seven interaction categories for visualization based on the user's intent, Card et al.'s venerable work [9] on three levels of time constraints in interaction, or in Nielsen's book on usability engineering [32]. While these works cover important needs of user involvement in general, we focus on bidirectional user involvement in ongoing computations (Sec. 3), and we derive strategies for achieving the desired involvement in practice (Sec. 4).

To enable earlier user involvement, strategies to accelerate result availability have been proposed in various contexts. Examples include pre-aggregation strategies for databases such as OLAP and data cubes [28], as well as sampling and filtering techniques for progressive refinement in online aggregation [16, 17, 23], enumerative queries [23], or data mining [51]. For subdividable problems, Divide-and-recombine (D&R) approaches split up a problem into multiple parts, solve the parts individually, and finally recombine the partial results. Examples include MapReduce [12] or RHIPE [21]. However, these examples focus more on speeding up the computation of large data by parallelization, rather than actively involving the user.

The need of visualizing incremental results can be found in many different application contexts beyond multivariate data analysis. Progressive drawing is a well-known approach in volume rendering [8], map rendering applications such as Google or Bing Maps, or the drawing of function graphs [35]. Particularly interesting in this respect is the work by Angelini and Santucci [1], as it provides a formal model that allows characterizing and evaluating incremental visualizations regardless of the application context. Furthermore, much research has gone into visually representing the uncertainty of incomplete results [17, 20, 34]. While this is important, the focus of this paper lies on achieving intermediate feedback in the first place, while particular visualization techniques are out of scope.

In summary, many approaches have been proposed to achieve user involvement in ongoing computations. Building on these possibilities, our primary goal is to provide a more formal characterization and comparison of strategies for achieving user involvement for different types of algorithms, together with a discussion in the context of integration with existing computational environments.

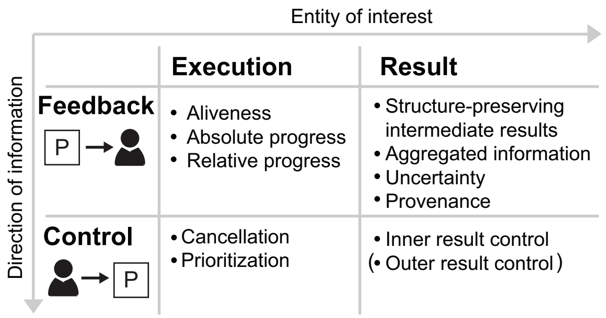

Fig. 1. Types of User Involvement (TUI) structure the needs of visual exploration concerning user involvement in ongoing computations along two directions: the direction of information and the entity of interest.

3 Types of User Involvement

As a starting point for discussing integration concepts, this section describes four different *Types of User Involvement* (TUI) that an interactive visualization may support for an ongoing computation of an algorithm *P*. In the context of this paper, the main goal of the TUI is to discuss the needs of visual exploration regarding the integration with ongoing computations more specifically.

The scope of the TUI is limited to the time between the start and the end of a computation of *P*, i.e., it does neither include a priori parameterization, nor any further application of the final result r_P. Accordingly, we define the scope of *P* such as to include any bounded computation or algorithm that has a well-defined end. Note that this explicitly excludes problems that can change over time, such as accounting for new data that concurrently arrives via streaming. We will refer to a variety of algorithms from multivariate analysis for illustrating the TUI and other concepts. In particular, we will relate to the well-known algorithms k-means clustering (as in the R method `kmeans`) and model-based feature subset selection (as in the R method `regsubsets`) as recurring examples whenever reasonable, and refer to them using the abbreviations KMEANS and SUBSETS.

We define the TUI based on two orthogonal dimensions (see Fig. 1):

1. **The direction of information.** We distinguish between *feedback* and *control*. Feedback comprises information which is passed from the computation to the user and requires an appropriate visual representation to enable an efficient and correct interpretation by the user. Control is information passed from the user to the computation and requires appropriate interaction techniques.

2. **The entity of interest.** We distinguish between information concerning the *execution* of the computation, and information concerning final or intermediate *results* of *P*.

The four TUI are defined as the Cartesian product of these dimensions and will be discussed in the following sections: execution feedback (Sec. 3.1), result feedback (Sec. 3.2), execution control (Sec. 3.3), and result control (Sec. 3.4). This classification of TUI is independent of any specific algorithm or the structure of its result as well as any particular implementation and strategy *how* a certain involvement has been realized. We emphasize that our focus is on the question *what* can be visualized and controlled and *why* from a user's perspective rather than on the issue *how*, which depends on the particular algorithm *P*. We also stress that this paper does not assume every type of user involvement to be indiscriminately beneficial for each situation. User involvement may incur costs and complexities on multiple levels including the implementation, the computation, and the application by users. Identifying the most appropriate degree of user involvement is a key topic of Visual Analytics [3, 11, 27, 46] and depends on the algorithm and the application context. Our focus is on the classification of known types of user involvement and on general strategies to accomplish it on a technical level rather than on the assessment, when specific TUI are appropriate.

[4]http://www.jmp.com
[5]http://spotfire.tibco.com

3.1 Execution Feedback

This TUI comprises any kind of feedback about the ongoing computation of P as such. Common types of information include:

- **Aliveness** confirms that the computation is in progress and no event has occurred that may cause failure to eventually deliver the final result, e.g., a crash, a deadlock, or a lost connection.
- **Absolute progress** includes information about the current execution phase of P which may be qualitative (e.g., "computing distance matrix...") or quantitative (e.g., "iteration 12" for KMEANS or the number of processed data items [1]).
- **Relative progress** includes information about the degree of completeness of P which is frequently provided as percentage or as an estimate of the remaining time.

From the user point of view, execution feedback should mainly answer two questions: First, can any result be expected at all? This may not be the case either due to the occurrence of a failure or an unacceptably long time required for computation. Second, does it make sense to wait for the result or do something else in between?

3.2 Result Feedback

This TUI involves any kind of intermediate feedback regarding the result r_P of the ongoing computation of P. We distinguish between four common classes of result feedback:

- **Structure-preserving intermediate results** $r_{\tilde{P}_i}$ are structurally equivalent to the final result r_P in the sense that the same techniques for visualization and data processing can be applied to them as surrogates while r_P is not yet available. This is typically the case for iterative and anytime algorithms. For example, the intermediate object positions after each iteration of a multidimensional scaling algorithm are structurally equivalent to the final positions. In case of the SUBSETS example, the best subset found so far has the same structure as the eventually best subset. Further examples from literature include non-negative matrix factorization [10], self-organizing maps [40], and data aggregation [17]. The structure of r_P may be multi-faceted, however, and consist of multiple parts of which only a subset is provided as feedback during computation. In the KMEANS example, r_P comprises both the cluster centers and the cluster assignments of all data points. To limit data transfer, showing only the intermediate centers during computation could be a reasonable option.
- **Aggregated information** provides a certain aspect of intermediate results $r_{\tilde{P}_i}$ without preserving the structure in full detail. Common examples are quality measures of $r_{\tilde{P}_i}$ [5], e.g., the goodness of fit for the best subset so far with respect to a certain type of model (in the SUBSETS example) or the overall stress of the current solution in case of multidimensional scaling.
- **Uncertainty** concerning the final result r_P as estimated based on the available intermediate information. An example are confidence bounds for r_P [17].
- **Provenance** includes any type of meta-information concerning simplifications made for generating $r_{\tilde{P}_i}$. Depending on the strategy to enable user involvement (see Sec. 4), this class may involve information about the considered data or the settings of complexity parameters. In this respect, provenance information is related to execution feedback about the absolute progress, but always refers to a particular intermediate result $r_{\tilde{P}_i}$.

An appropriate visual representation depends on many aspects like the type and structure of the intermediate results $r_{\tilde{P}_i}$, the update rate of $r_{\tilde{P}_i}$, the involved amount of transferred data, and the intended goal of the visualization. It should, however, ensure that the result is perceived as intermediate. One option for doing so is to explicitly represent the change of the intermediate results over time. Examples include techniques of comparative visualization [19] to represent the difference between $r_{\tilde{P}_i}$ and $r_{\tilde{P}_{i-1}}$, or line graphs to visualize the convergence of aggregated information or uncertainty over time [17].

The key benefit of intermediate result feedback for the user is to enable an earlier continuation of the analysis based on preliminary information. Moreover, result feedback supports the decision whether the ongoing computation should be cancelled. This may be the case if the current intermediate result is already good enough or if the final result is not likely to be good enough. Finally, access to intermediate results is a key requirement for result control (see Sec. 3.4).

3.3 Execution Control

This TUI involves any kind of control of the execution of the ongoing computation of P as such. The most important type of execution control is **cancellation**, i.e., an explicit or implicit request to cancel the execution prematurely. Explicit requests are issued by the user if control feedback or result feedback suggests that either intermediate results are good enough, or the final result is unlikely to be good, or no result can be expected in acceptable time at all. Implicit requests are typically triggered by updated dependencies of the algorithms like changed input data and algorithm parameters. Such requests often entail a subsequent restart of the computation, a paradigm described in the context of multi-threaded visual analysis [36].

Another type of execution control is the **prioritization** of the remaining work. While the final result r_P is not affected, the purpose is to alter the sequence of intermediate results in order to generate presumably more interesting ones earlier. In this respect, prioritization can be regarded as borderline case between execution and result control. Examples include algorithms involving spatial partitioning or hierarchical structures where users may want to process more interesting parts first [50]. As another example, algorithms processing search spaces may benefit from looking into more promising regions first.

3.4 Result Control

This TUI refers to user interaction with the ongoing computation of P in order to steer the final result r_P. This enables users to take advantage from human perception and domain knowledge [3, 25, 45], e.g., for early validation of intermediate results, guided feature selection, weighting, and for avoidance of being stuck in local extrema. In the widest sense, this TUI corresponds to the common understanding of the Visual Analytics process as defined by Keim et al. [26]. Consequently, a significant share of the Visual Analytics literature addresses this TUI, e.g., clustering [31], classification [48], regression [30], dimension reduction [14], distance functions [7], and many others.

In the context of this paper, it is helpful to distinguish between inner and outer result control. The difference is whether the steering is based on intermediate results of a single execution of P, or on final results of multiple individual executions. **Inner result control** thus refers to the ability of controlling a single ongoing computation of P before it eventually returns a final result. Typical examples are partial modifications of the computation state between two consecutive iterations of P. In the KMEANS example, users could be allowed to shift, merge, or split cluster centers between iterations.

Outer result control involves multiple consecutive executions of P that do not directly re-use previous results. It imposes no requirements on the algorithm, but relies on the visualization tool to enable the discourse between the user and the computation. As stated above, our scope of the TUI is limited to the time between the start and the end of a single computation of P. Therefore, outer result control is not relevant for this paper from the point of view of algorithm design.

4 Strategies for Achieving User Involvement

The previous section defined types of user involvement in ongoing computations. This section describes four strategies S1 – S4 to achieve user involvement for algorithms with different characteristics. We note that our focus is on the technical applicability of these strategies for enabling *any* type of user involvement, not on the discussion when specific TUI are appropriate from an application point of view. The motivation of these strategies within this paper is achieving a tighter user involvement in integrations of interactive visualization software with computational environments, such as R or MATLAB. However, their formulation does not rely on this application context, but can be regarded as a contribution to general algorithm design regarding early user involvement.

The common key idea of the four strategies is to replace the execution of an algorithm P by a series of smaller steps $\{\tilde{P}_1, \ldots, \tilde{P}_n\}$ in order to allow feedback and control between any subsequent steps $\tilde{P}_i$

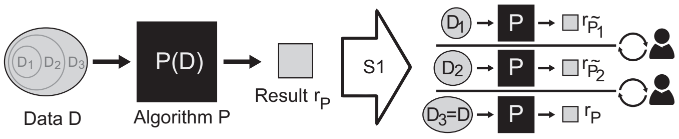

Fig. 2. S1: Data Subsetting. Additional passes of P for increasing subsets of data are computed to allow user involvement after a shorter time.

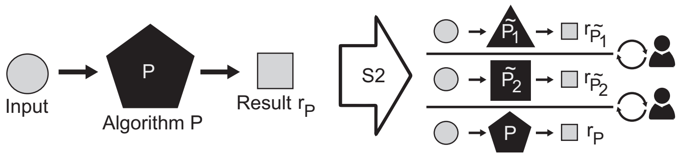

Fig. 3. S2: Complexity Selection. Additional passes of P for simplified parameters are computed to allow user involvement after a shorter time.

and $\tilde{P_{i+1}}$. The simplification of P to steps $\tilde{P}_i$ can be achieved in several ways, which can be characterized based on two orthogonal aspects: (1) The *dimension* of simplification of P can either be the *input data*, the *parameters*, or the *algorithm* itself. (2) The *approach* of simplification along these dimensions can be realized by *subdivision* of P for divisible problems, or based on *simplified extra passes* if a subdivision is not possible. The Cartesian product of these two aspects yields six combinations which are entirely covered by our four strategies: S1 computes extra passes for simplified data, S2 computes extra passes for simplified parameters or a simplified algorithm. S3 subdivides P into subproblems with respect to data or parameters, while S4 subdivides the control flow of the algorithm as such. In this sense, we argue that our set of strategies is complete in the context of our scope, i.e., enabling user involvement during the computation of operations with bounded effort.

Motivated by the structured approach of describing and comparing design patterns in software engineering [18], we characterize each strategy in terms of the name, definition, scope, examples, considerations regarding user involvement, and computational implications.

4.1 S1: Data Subsetting

Definition. Perform computations of P for increasingly larger subsets $D_i \subseteq D_{i+1} \subseteq D$ of data records or dimensions of a data table D *in additional passes* and enable user involvement after completing every pass $\tilde{P}_i = P(D_i)$ (see Fig. 2).

Scope. S1 operates solely in the data space. As a consequence, S1 is structurally applicable to any algorithm that operates on a data table or data vector D. From an application point of view, S1 requires that intermediate results provide a meaningful approximation of the final result r_P. Specifically, this is the case for algorithms inferring a global structure like clusters, trends, and aggregation.

In contrast to strategies subdividing the workload into disjoint segments (i.e., S3), the subsetting of D does not rely on reusing results between passes. As a consequence, S1 is in particular applicable in situations where algorithms cannot reasonably be subdivided for inherent structural reasons or due to constraints imposed by their programming interface. This makes S1 the most generally applicable strategy that requires little knowledge about the inner structure of P.

However, the type of P determines whether subsetting is possible and reasonable in terms of *data records* (i.e., rows of D) or in terms of *data dimensions* (i.e., columns of D). Another important consideration is the method for defining the subsets of D, which significantly affects how representative the intermediate results are. This issue is directly related to sampling, which is discussed extensively in the literature also in context of visualization [4, 13, 28, 33].

Examples. As stated above, S1 is generally suitable for algorithms inferring a global structure or information. This includes, for example, most algorithms from unsupervised statistical learning [22]. In this case, subsetting data records may enable an early detection and potentially correction of wrong assumptions or inadequate parameters, e.g., a wrong number of clusters in the KMEANS example. Dimension reduction techniques like PCA and MDS may also benefit from subsetting of data records just as most descriptive statistics like statistical moments (e.g., mean, variance, skewness), percentiles, etc.

While the purpose of subsetting in S1 is achieving early user involvement, techniques from supervised learning may already include record-based subsetting for the purpose of model validation [22]. Depending on the purpose of the model, however, applying S1 for speedup may still be applicable, e.g., when visually indicating linear trends in a scatter plot. More care must be taken with subsetting of *dimensions* in machine learning, as the selection of features is very critical for the quality and representativity of results.

User involvement. The key parameters for enabling user involvement are the number and the size of the data subsets D_i. These parameters enable a tradeoff between frequency of user involvement, quality in terms of completeness of intermediate results, and computational overhead. The typically known size of D_i relative to D enables a direct quantification of the completeness in terms of the considered data size. This information should especially be conveyed as feedback regarding result provenance and can also be seen as absolute progress. Feedback concerning the relative progress with respect to the overall time, however, requires intimate knowledge of the computational complexity of P. Especially for client-driven implementations (Sec. 5.1), this may also be a major challenge for enabling user control mechanisms at an approximately equal rate.

Computational implications. The computational overhead of S1 is the sum of all $P(D_i)$, which can be very significant. On the other hand, the execution of the $P(D_i)$ and the $P(D)$ is easily parallelizable. In so far, S1 does not necessarily incur a latency for receiving the final result. In an extreme case, all $P(D_i)$ as well as $P(D)$ are scheduled independently, loosely relying on the increasing effort for computing increasing percentages of D to arrive in order. The memory consumption, however, typically also increases with the degree of parallelization due to a multiplication of algorithm-internal structures. Regarding the storage of D_i, indexing of D should be used whenever possible to avoid data duplication and reduce data transfer.

4.2 S2: Complexity Selection

Definition. Perform computations of P for less complex parameter configurations $\tilde{P}_i$ *in additional passes* before computing P itself, and enable user involvement after completing every pass $\tilde{P}_i$ (see Fig. 3).

Scope. S2 operates in the parameter space of P. Therefore, the applicability of S2 is determined by the existence and accessibility of complexity parameters that enable a speed vs. quality tradeoff. In particular, this applies to approximation algorithms [49] and many heuristics of computationally intractable problems in operations research, but also to many algorithms in other fields like statistics (see below).

In contrast to S1 that operates in data space, the application of S2 is very dependent on P and typically requires structural knowledge of P and the effect of parameter changes in context of the specific data. This is a highly non-trivial issue in general as also shown by the growing importance of parameter space analysis as a topic in visualization literature [41]. In particular, the purpose of many complexity parameters in statistics is to adjust the suitability for particular data and a particular purpose rather than to simply trade off quality versus speed. An example is the bias vs. variance tradeoff of many types of statistical models [22], where additional complexity improves the model quality only to a certain point before degrading generalizability due to over-fitting. As a consequence, S2 should only be considered for algorithms where concluding from intermediate results $r_{\tilde{P}_i}$ to the final result r_P is meaningful.

On the other hand, S2 does not require any structural decomposability of P, as it is the case for S3 and S4. In contrast to S1 which requires vector- or tabular-oriented data, S2 is also applicable to algorithms working on non-decomposable operands like analytical functions.

Examples. In operations research, approximation algorithms for computationally intractable problems are common. They provide a solution that is provably optimal up to a constant – and often definable – factor and have provable run-time bounds [49]. For nearest neighbor search, for example, ε-approximate variants exist that enable to trade off the probability of finding the true nearest neighbor versus space and time costs especially in high-dimensional spaces (e.g., Arya et al. [2]).

Further examples of complexity parameters include the refinement

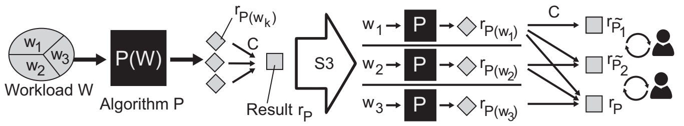

Fig. 4. S3: Divide and Combine involves (1) applying P to independent parts w_k of a workload W, and (2) recombining the results using a combination C to obtain intermediate results or the final result.

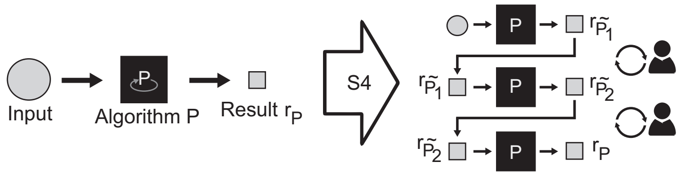

Fig. 5. S4: Dependent subdivision. The idea is to involve the user between sequentially dependent steps of algorithms P, e.g., iterations.

of subdivision schemes, the size of search radii, thresholds for stopping criteria, or even the algorithm itself as long as different algorithms yield structurally equivalent results, which includes most heuristics. Many algorithms for feature subset selection (SUBSETS), for example, differ in whether they perform a greedy or an exhaustive search.

User involvement. The key parameters for enabling user involvement are the number of approximation steps and the complexities of the steps. Unlike for S1, the degree of freedom for both parameters is determined by P. Quantitative complexity parameters like thresholds may enable a precise choice of the number of approximations and even a quantification of the completeness or precision (e.g., for estimating relative progress). Conversely, categorical parameters such as available heuristic algorithms may impose a strict limitation on the number and complexities of steps, and complexities may be hard to estimate or even order. This makes a quantification of the progress difficult or impossible in general and only enables qualitative feedback regarding progress and result provenance. While feedback regarding result provenance is essential especially for strategy S2, however, only expert users will often be able to interpret the information. As for S1, another challenge for client-driven implementations (Sec. 5.1) will typically be to enable user control mechanisms at an approximately equal rate.

Computational implications. The computational overhead of S2 is the sum of additional passes for computing the steps $\tilde{P}_i$. However, these computations are independent and thus easily parallelizable. As discussed for S1, this enables to reduce or avoid the latency for receiving the final result r_P at the cost of increasing memory consumption.

4.3 S3: Divide and Combine

Definition. Subdivide a workload W into n disjoint parts $\{w_1, ..., w_n\}$, apply P independently to each part w_k to generate *partial results* $\{r_{P(w_1)}, ..., r_{P(w_n)}\}$, and compute intermediate results or the final result based on combining some or all $r_{P(w_k)}$ (see Fig. 4).

Scope. S3 imposes two requirements on P: First, independent applications of P to parts w_k of a subdivided workload W must be possible in order to generate the partial results $r_{P(w_k)}$. Second, a meaningful combination C must exist to combine subsets of partial results to *intermediate results* $r_{\tilde{P}_i}$ that are structurally equivalent to the final result r_P. In particular, applying C to *all* partial results yields the final result r_P of P. In context of S3, a single step $\tilde{P}_i$ thus comprises the computation of a subset of partial results and the application of C to them.

The subdivision of W can be defined in terms of *data* (i.e., data space-based) or *parameters* (i.e., parameter space-based) that P is applied to. A data space-based subdivision is specifically possible in cases where applying P to a collection of elements (e.g., a set, a vector, a matrix) internally involves applying the same operation to each element. A parameter space-based subdivision is applicable to algorithms that take a specification of a domain (e.g., the extents of a search space) as a parameter, given that disjoint parts of the domain can be processed independently. It should be noted that a disjoint subdivision of the workload W does not in all cases imply a disjoint subdivision of the data or parameter space considered by each $P(w_k)$ for computation. In other words, the disjoint subdivision applies to the *output* of P rather than the *input* of P. Tolerating a certain overlap in the inputs of multiple $P(w_k)$ extends the applicability of S3 to operations that require a specified context around each processed element, e.g., for convolution or pattern search.

The characteristics of the combination C depend on the structure of the result r_P. In many cases, C is a composition of partial results in order to restore their context within W which has been lost due to the initial subdivision. Sometimes, C may also be a simple aggregation (e.g., maximum or mean). In any case, a practical requirement is that an application of C should be cheap to perform compared to computing the partial results themselves. The reason is that the intermediate results $r_{\tilde{P}_i}$ are composed of arbitrary and non-disjoint subsets of the partial results (see Fig. 4). C may thus be applied to a single partial result multiple times for generating different intermediate results.

In order to avoid confusion, we point out that S3 is related and often applicable yet not equivalent to divide and conquer (D&C) algorithms [6]. For D&C algorithms, the subdivision is an inherent property of the algorithm while S3 refers to a subdivision-based strategy in a broader sense. In particular, S3 does not require that the application of P to a single element $P(w_k)$ becomes trivial. In this respect, S3 is more related to parallelization paradigms like MapReduce [12].

Examples. An example for data space-based S3 is the sampled evaluation of a function, e.g., for progressive rendering of increasingly fine-grained function graphs. In this case, P is the evaluation of the function for a set of positions, W are the positions of all samples, w_k is a certain subset, and C restores the context (i.e., the position and order) of results of P within W. Another example is the progressive computation of aggregates, e.g., the average [17]. In this case, P may involve potentially optimized algorithms for computing the aggregate for blocks of data w_k and C further aggregates multiple $r_{P(w_k)}$ according to their cardinality.

A parameter space-based example is the computation of the autocorrelation of a time series. The parameter refers to the interval (W) of considered lags. This interval can be separated in disjoint parts and C recomposes the autocorrelation as a function of the lag-size.

User involvement. The two key characteristics of S3 are the degree of subdivision (*DIVIDE*, i.e., the number n of w_k), and the strategy to generate the intermediate results $r_{\tilde{P}_i}$ (*COMBINE*). In general, DIVIDE is the more decisive factor for execution feedback and control while COMBINE determines the frequency and quality of result feedback. An approximately uniform subdivision of W facilitates feedback regarding the relative progress as compared to S1 and S2 and also enables user control like cancellation at a roughly equal rate.

Regarding result feedback, COMBINE defines the ordering of the computations for all parts w_k, and the amount of additional completeness for each intermediate result $r_{\tilde{P}_i}$. As the number of intermediate results $r_{\tilde{P}_i}$ is independent of DIVIDE, the rates for feedback and control may be different. It is therefore a possible strategy to internally decouple the processing of COMBINE from DIVIDE (e.g., by multithreading), and to define the progress for each $\tilde{P}_i$ in terms of additional time rather than W by applying C to all already completed $r_{P(w_k)}$.

Computational implications. Increasing the degree of subdivision DIVIDE enables more fine-grained execution feedback and control without inherently inferring higher costs for obtaining the final result $r_{P(W)}$. In practice, however, each application of P may involve a certain overhead for reasons including the internal structure of P, potentially overlapping inputs of multiple $P(w_k)$, and implementation-related issues (e.g., data transfer, initialization, etc.). The latter are typically more significant for client-driven implementations (see Sec. 5.1). In addition, the overhead of S3 includes generating intermediate results from partial results and thus depends on COMBINE.

The independence of all $P(w_k)$ makes S3 suitable for performing computations in parallel. In general, parallelization is typically a key motivation for subdivision. In the context of user involvement, however, a certain degree of sequential scheduling is required in order to involve the user between independent subsets of workload parts.

4.4 S4: Dependent Subdivision

Definition. Subdivide P into sequentially dependent steps $\tilde{P}_i$ so that the result $r_{\tilde{P}_i}$ of each step is an input to the next step $\tilde{P}_{i+1}$, and is structurally equivalent to the final result r_P. Enable user involvement between steps (see Fig. 5).

Scope. S4 poses requirements regarding the decomposability of P and the structural equivalence of the result of each step $r_{\tilde{P}_i}$ to r_P. In particular, this includes iterative algorithms where the result of each iteration serves as input to the next. In addition to inherently iterative problems, multiple problems can directly be transformed to iterative problems (e.g., recursive problems [6]), or an iterative variant exists (e.g., iterative PCA [39]). While iterative algorithms are the by far most important example of S4, the sequential steps could also be defined in terms of an ordered domain that needs to be processed sequentially, e.g., progressive signal reconstruction as described below.

In contrast to S1 and S2, each step $\tilde{P}_i$ can reuse the previous result $r_{\tilde{P}_{i-1}}$ to avoid redundant computation. In contrast to S3, each step $\tilde{P}_i$ depends on the previous step $\tilde{P}_{i-1}$, i.e., it is not possible to decompose the workload into independent parts.

Examples. S4 is in particular applicable to inherently iterative algorithms. In statistical learning, prominent examples include (1) the training of regression or classification models such as neural networks, (2) dimension reduction algorithms like multi-dimensional scaling, and (3) clustering algorithms such as partitioning around medoids or the recurring example KMEANS. Other examples are force-based algorithms for graph layout [29], as well as algorithms that – potentially recursively – build hierarchical structures (e.g., decision trees), where each recursion adds to the complexity of $r_{\tilde{P}_i}$.

Concerning sequential processing of an ordered domain, consider a progressive reconstruction of a signal (e.g., a time series or an image) from a frequency-based representation, as common for displaying large JPEG images. An implementation of S4 could define $\tilde{P}_i$ as to reconstruct a certain disjoint band of increasingly higher frequencies and to add the result to the already reconstructed part of the signal.

User involvement. The key parameter for enabling user involvement is the step size, denoted by s. Varying s enables to trade off the frequency of feedback and user control against the computational overhead involved with each step. For iterative algorithms, s is typically defined in terms of iterations which enables user involvement at an approximately equal rate. Whether relative feedback can reasonably be provided depends on whether the number of steps is known in advance. As this often does not apply to convergent algorithms, a distance from a termination criterion may be provided instead.

In contrast to all other strategies, each step $\tilde{P}_i$ depends on the result of the previous step $\tilde{P}_{i-1}$ for S4. We argue that this is a requirement for permitting meaningful control of the ultimate result r_P within an ongoing computation of P, i.e., enabling inner result control (see Sec. 3.4). For S1 and S2, changing data or parameters typically requires to restart computing all intermediate results, beginning with the simplest step. For S3, obtaining a homogeneous final result r_P requires that each step $P(w_k)$ is computed in the same way for all independent workload parts w_k. For these reasons, outer result control is more appropriate when applying S1, S2, or S3. In contrast, for iterative algorithms, inner result control can be reasonable to enable domain knowledge for affecting convergence (e.g., avoiding local extrema).

Computational implications. As discussed for the degree of subdivision of S3, the step size s may have a practical effect on the computational overhead imposed by S4. Unlike for S3, however, the sequential dependence of the $\tilde{P}_i$ on each other does not permit parallelization.

5 Client-driven vs Algorithm-driven implementation of strategies

The previous sections characterized types of user involvement in ongoing computations (Sec. 3) and described four strategies to achieve user involvement for algorithms with different characteristics (Sec. 4). This section discusses possibilities of realizing the strategies when integrating interactive visualization software and computational environments. We will refer to these environments as VIS and COMP, where

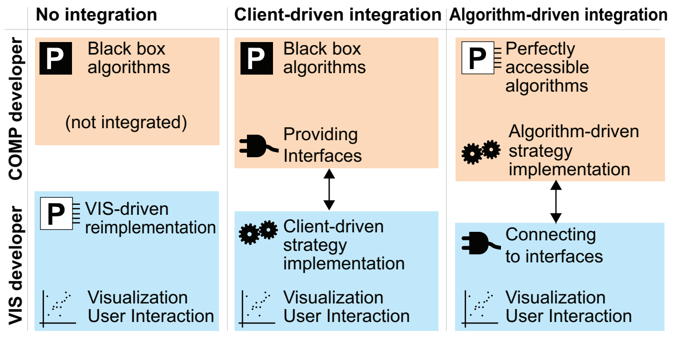
No integration
Client-driven integration
Algorithm-driven integration
COMP developer
VIS developer
Black box algorithms
(not integrated)
Black box algorithms
Providing Interfaces
Perfectly accessible algorithms
Algorithm-driven strategy implementation
VIS-driven reimplementation
Visualization User Interaction
Client-driven strategy implementation
Visualization User Interaction
Connecting to interfaces
Visualization User Interaction

Fig. 6. Reimplementing algorithms P in VIS allows to achieve user involvement but involves substantial effort for VIS (left). In client-driven integration, VIS implements strategies for achieving user involvement by connecting to existing P that provide adequate interfaces (middle). In algorithm-driven integration, VIS connects to algorithms that implement strategies and provide communication directly from within (right).

VIS can be any interactive visualization tool and COMP can be computational environments such as R or MATLAB, as well as any other external computational resource or library.

We discriminate between three responsibilities:

Algorithm refers to performing the computation of P or $\tilde{P}_i$.

Client refers to visualizing the feedback of *Algorithm* and the handling of user input, i.e., the elements for human-computer interaction.

Flow control refers to implementing the control flow and communication between *Algorithm* and *Client*. This includes the implementation of a strategy for defining and scheduling the steps $\tilde{P}_i$.

Currently, a frequent situation in Visual Analytics is that all three responsibilities, i.e., *Algorithm*, *Client* and *Flow control* are implemented as part of VIS (see Fig. 6, left column), which has disadvantages as pointed out in related work [15] and in the introduction. In context of integrating VIS and COMP, however, we assume that the role of the *Algorithm* is provided by COMP and the role of the *Client* is taken by VIS. In this case, the *Flow control* can either be a responsibility of VIS (*client-driven integration*, center column in Fig. 6), or a responsibility of COMP (*algorithm-driven integration*, right column in Fig. 6). Characterizing, comparing, and discussing these two scenarios is the purpose of this section. We establish requirements imposed by client- and algorithm-driven integration, and we derive guidelines to the design of the interface of P in favor of flexible client control.

5.1 Client-driven integration

In client-driven integration, the definition and scheduling of the $\tilde{P}_i$ is managed by VIS, while their computation is performed by COMP. Between any steps $\tilde{P}_i$ and $\tilde{P}_{i+1}$, VIS can realize user involvement, e.g., by visualizing $r_{\tilde{P}_i}$ or by adjusting the call of $\tilde{P}_{i+1}$ according to user input. This *externalization of the control flow* requires that the programming interface of P exposes all parameters that clients need to define $\tilde{P}_i$. We subsequently analyze these requirements for each strategy.

In S1, the $\tilde{P}_i$ are defined as executions of P on subsets of the input data D. Externally produced subsets can be fed to P instead of the full D, making S1 applicable for client-driven integration without additional requirements. In S2, a client-driven selection of complexity involves calling P for different parameters. This yields the following interface **R**equirement for a **C**lient-driven integration (RC) for S2:

RC 1. *Expose complexity parameters to trade off speed for quality.*

Considering the central role of most complexity parameters, this criterion is not very limiting in practice (see Sec. 6). For S3, clients need to define a part of the subdivided workload W for each call of P. To support S3, the interface of P must meet the requirement RC2:

RC 2. *Enable a precise specification of the processed workload.*

For data space-based applications of S3, a subdivision into coherent blocks is often possible on the client side. Parameter space-based applications of S3 require the possibility to fully specify boundaries via the interface, e.g., the lower and upper limit of a considered subspace.

Requirement RC2 also holds for S4, as the sequential processing of iterations is also based on a decomposition of workload. The specification can either be explicit, i.e., the number of iterations performed in $\tilde{P}_i$, or implicit, by means of a stopping criterion.

A second requirement of S4 is the ability to pass the final state of $\tilde{P}_i$ on to $\tilde{P_{i+1}}$. We identify RC3 for S4 as follows:

RC 3. *Provide access to all parts of the state as output, and accept equivalent information as input, in order to enable resuming the computation with minimal redundancy.*

The form of this state information depends on P. For statistical learning, the state is often the model itself (e.g., a regression model or cluster centers). While the model is typically the output of such P, not all interfaces support accepting a model as an input to proceed with.

It should be noted that not all strategies are appropriate for every algorithm. Given that a particular strategy is appropriate for P, algorithm developers should account for those RC that are required by this strategy in order to enable an appropriate degree of user involvement. Regarding execution feedback, the arrival of steps $\tilde{P}_i$ and their definition can be reported for aliveness and absolute progress. Regarding result feedback, the client may directly visualize intermediate results $r_{\tilde{P}_i}$ or use any output of steps $\tilde{P}_i$ to derive information such as quality metrics as a post-processing step in VIS. Regarding execution control, considering user input in between enables to call a specific $\tilde{P_{i+1}}$ for prioritization, or not at all for premature cancellation. Regarding result control, $\tilde{P_{i+1}}$ can be called for modified inputs.

Client-driven control enables several possibilities of realizing parallelization of steps $\tilde{P}_i$. For S1, S2, and S3, multiple invocations of P can be parallelized using multiple threads within COMP, multiple instances of COMP, or even different computers in network- or cloud-based environments. In practice, however, the incurred latency may exclude some options regarding responsiveness for user involvement.

As an inherent limitation of client-driven integration, user involvement is only possible between steps $\tilde{P}_i$, i.e., invocations of P. This may limit the achievable frequency of user involvement as opposed to a reimplementation of P.

5.2 Algorithm-driven integration

In algorithm-driven integration, the *Flow Control* is realized directly within the implementation of P. Specifically, P is responsible for defining simplification steps $\tilde{P}_i$ according to a particular strategy, and for communicating with the *Client* in order to enable user involvement.

Definition of the simplification steps. When defining the steps $\tilde{P}_i$, four objectives can be identified for different TUI: (1) Execution feedback should be provided as precisely as possible and at approximately equal rates. (2) Result feedback should provide good approximations of r_P as early as possible. (3) Both execution control and result control should have a minimal latency. (4) The computational overhead of user involvement should be minimal.

These objectives are partly contradicting each other. Defining the steps $\tilde{P}_i$ represents a mechanism to control this tradeoff for optimization within a given context. While the four objectives generally apply to client-driven as well as algorithm-driven integration scenarios, typically the client knows their preference in context. For algorithm-driven integration scenarios, it is thus desirable that the client has means of controlling the definition of steps $\tilde{P}_i$ via the interface of P. However, a direct definition of steps often requires an intimate knowledge about the inner structure of P. In this sense, an algorithm-driven *Flow Control* provides two key advantages over client-driven integration:

First, the implementation of P is the more appropriate place to be aware of the inner structure and any implications than the client. Ideally, the client can specify the preference of the objectives and certain constraints (e.g., a minimal frequency of feedback) while the algorithm knows *how* to realize this specification. Based on this consideration, we formulate the following **G**uideline for the design of P's interface in the context of **A**lgorithm-driven integration (*GA*):

GA 1. *Offer means for specifying preference and constraints by the client regarding desired feedback and control rates.*

A second advantage of algorithm-driven integration is that communication is not limited to the times between structurally equivalent $\tilde{P}_i$. For example, execution control can be realized after arbitrary blocks of code, allowing to check for cancellation signals often without having to generate result feedback at the same rate. This *source-code level of granularity* also allows minimizing the overhead of executing multiple steps $\tilde{P}_i$ instead of a single P, as the products of potential common initialization steps can be reused.

Communication with the *Client*. In algorithm-driven integration, the extent of supported feedback and control is entirely up to P. This makes sense, as appropriate types of user involvement are strongly algorithm-dependent. On that account, it is a key goal of this paper to encourage algorithm developers to acknowledge the degree of supported user involvement as a conscious design choice.

The exchange of information between P and VIS can be implemented in different ways. A simple feedback mechanism commonly found in command line-based computation environments is providing a textual *trace* of the ongoing computation to a console. This one-directional form of communication is usually intended to be read directly by users, not clients like VIS. As a result, parsing the trace may be difficult and highly algorithm-specific. As it is intended for console display, larger amounts of data can not be communicated reasonably.

A more flexible option is the definition of an interface by the algorithm for sending feedback to and querying control information from an unknown client during the computation. Technically, a broad set of communication techniques exists, including the registration of call back procedures, dedicated points of code insertion, registration mechanisms implementing the Observer design pattern [18], message passing and application-layer network protocols.

As a common guideline in software engineering, we argue for a separation of concerns in that algorithm implementations should need to care more about *what* to communicate and *when*, rather than about *how* and to *whom*. Details of the communication such as the number and location of clients, or issues like parsing protocols should be decoupled from the actual implementation of P in order to minimize the implementation effort of algorithm developers and to maximize the reusability of an implementation in various environments. We see two options to achieve this:

The first option is pragmatic in the sense that the algorithm developers should support the communication technique that incurs the minimal effort on their side. Translating one of the aforementioned techniques to another is typically possible and requires an Adapter [18] which can – and should – be realized outside the algorithm, for example by the client developer. In many cases, the most simple technique is providing means for registering *callback methods* in the same programming language as the algorithm implementation. This inversion of control enables a single client to insert code into P that is called at semantically meaningful positions of the control flow for exchanging feedback and control signals, e.g., between iterations. P does not need to know the client but executes callbacks as a black box. A direct use by clients may pose certain challenges, such as different languages of VIS and COMP, or requiring VIS and COMP to be executed on the same machine. However, as argued above, Adapter objects can be defined to address these challenges, e.g., by translating local calls to Remote Procedure Calls (RPC). We thus suggest the following interface guideline as a pragmatic step towards separation of concern:

GA 2. *Provide a callback interface to allow a client-side customization of the communication protocol.*

The second option of the algorithm developer is to rely on an existing communication infrastructure, which may be external or internal. External refers to libraries and middleware outside the environment of COMP, e.g., for message passing. While this typically enables more powerful communication possibilities (e.g., over a network), a disadvantage for clients could be to incur the communication infrastructure as potentially unwanted dependency. In contrast, an internal infrastructure refers to a dedicated extension of the COMP environment itself (e.g., MATLAB or the R core) that algorithms and clients can use for benefit without additional complexity or dependency. However, such extensions are typically not provided today. *We thus recommend to realize powerful and easy-to-use communication mechanisms*

Algorithm (package)	**Description**	**Operates on table**	**Exposes complexity param. (RC1)**	**Definable workload (RC2)**	**State restorability (RC3)**	**Provides communication from within**	**Comm. granularity (GA1)**	**Allows callbacks (GA2)**
tsne (tnse)	T-SNE dimension reduction	✔	✔	✔	✔	✔ (trace+GA2)	✔	✔
neuralnet (neuralnet)	Neural network	✔	✔	✔	✔	✔ (trace)	✔	✘
optim (stats)	Optimization	-	✔	✔	✔	✔ (trace)	✔	✘
sammon (MASS)	Multi-dimensional scaling	✔	✔	✔	✔	✔ (trace)	✘	✘
vegas (R2Cuba)	Monte Carlo Integration	-	✔	✔	✔	✔ (trace)	✘	✘
kmeans (cluster)	K-means clustering	✔	✔	✔	✔	✘	✘	✘
som (kohonen)	Self organizing map	✔	✔	✔	✔	✘	✘	✘
emcluster (EMCluster)	Expectation max. clustering	✔	✔	✔	✔	✘	✘	✘
rpart (rpart)	Recursive tree construction	✔	✔	✔	✔	✘	✘	✘
regsubsets (leaps)	Best subset feature selection	✔	✔	✔	✔	✘	✘	✘
biglm (biglm)	Linear model	✔	✔	✔	✔	✘	✘	✘
pam (cluster)	Partitioning around medoids	✔	✔	✘	✔	✔ (trace)	✘	✘
acf (stats)	Autocorrelation	✔	✔	✘	✘	✘	✘	✘
ksvm (kernlab)	Support vector machine	✔	✔	✘	✘	✘	✘	✘

Table 1. A survey of frequently used R algorithms regarding the fulfillment of the identified requirements and guidelines in favor of a tight integration.

between algorithms and clients as valuable future core extensions of widely-used COMP environments.

A key consideration of respective extensions refers to their level of abstraction. Low-level mechanisms do not directly support any semantics of the communicated information but rely on the end points to do so. Conversely, high-level mechanisms could directly provide support for specific types of user involvement. For execution feedback and control (e.g., cancellation), this seems rather straightforward. For result feedback and control, however, defining standardized means seems highly non-trivial, but would enable benefits like querying intermediate results of different algorithms transparently to the client. While this may be a too demanding step for current computation environments, considerations like these could be a starting point for designing new computation infrastructures, as suggested by Fekete [15].

6 Case Study "R": Applicability of Strategies

In the previous section, we identified a set of requirements and guidelines for the design of algorithmic interfaces in favor of an applicability of the proposed strategies. In this section, we investigate to which degree an exemplary computational environment fulfills the requirements of client-driven integration or even actively supports user involvement in the sense of algorithm-driven integration. Specifically, we surveyed 14 common algorithms for important problems related to multivariate analysis from the scripting environment R. The selection of R was motivated by its broad acceptance in academia and corporate research, the choice of algorithms was inspired by R's reference list of recommended packages for common topics, *CRAN Task Views*[6].

Table 1 gives an overview of the survey results. The table suggests that the large majority of the inspected algorithms supports a client-driven application of multiple strategies, while only a few of them directly provide algorithm-driven feedback. This indicates that there is currently a large potential of realizing user involvement at the hands of VIS developers, as well as potential for COMP developers to support user involvement more directly.

The data-based S1 is applicable to all algorithms operating on a data table. This applies to all our examples except for optimization (`optim`) and monte-carlo integration (`vegas`) , which take analytic functions as inputs. Also, all investigated algorithms expose some complexity parameter or method selector that influences the runtime of single steps (RC1). For example, the *pamonce* option enables algorithmic short cuts in Partitioning-around-medoids clustering (`pam`), and best-subset feature selection (`regsubsets`) offers a selector of exhaustive vs. stepwise methods. Furthermore, the majority of investigated subdividable algorithms fulfills the interface requirements of strategies S3 and S4 (RC2, RC3).

However, not all surveyed algorithms fulfill RC1 - RC3, which allows us to discuss potential interface improvements for specific real-world examples. Note that this discussion is neither an assessment of the algorithms themselves nor their specific implementations.

[6]http://cran.r-project.org/web/views

Example 1: The clustering method `pam` iteratively performs an exhaustive search of medoids, i.e., data records that exhibit a minimal sum of distances to all other records. While `pam` allows the specification of an initial set of medoids (RC3), it is not possible to subdivide iterations into separate calls (RC2). Adding a numeric parameter indicating the number of iterations to perform in each step would enable users to suggest cluster medoids in-between, in order to speed up convergence for large datasets as well as to avoid local minima.

Example 2: The iterative training of support vector machines as provided by `ksvm` does not expose the divisibility of the underlying Sequential Minimal Optimization [37]. However, the usefulness and convergence of SVMs highly depends on the choice of multiple model parameters. We suggest to enable a specification of the number of iterations and the previously trained model as input parameters of `ksvm`. This would allow early previews and cancellation of the model identification for an exploration of model parameters.

Example 3: Computing the autocorrelation function of a time series (`acf`) can be seen as a divide-and-combine approach of computing correlations between a time series T and different lags of T. While `acf` allows a specification of the longest computed lag (*lag.max*) in the sense of RC1, it lacks the counterpart *lag.min* needed for a workload specification according to RC2.

After surveying the examples regarding the client-driven applicability of strategies, we now discuss the direct support of user involvement as provided by algorithms. Several algorithms provide a uni-directional *trace* of textual feedback to a console during their execution (`pam, vegas, sammon, optim, neuralnet` and `tsne`). Most of them allow specifying different levels of verbosity, while `tsne, neuralnet` and `optim` even allow specifying the interval between messages (GA1). Apart from this trace, one example comes close to the *perfectly accessible algorithm* as outlined by algorithm-driven integration: The iterative `tsne` algorithm for dimension reduction allows clients to define a callback (GA2) that is executed instead of printing the trace at regular, client-definable intervals (GA1). This enables flexible feedback in a consistent way.

However, we found no algorithms that consider *control* signals during their execution. A possible explanation could be that R usually runs in single-threaded, stand-alone command line environments, where the receiving of concurrent control signals is practically not feasible. With callbacks at hand, however, algorithms could incorporate control by considering the return value of callbacks in their control flow. As long as measures like this have not been adopted, providing user control is possible by implementing S1-S4 on the client-side.

This case study shows that very few of the examined R implementations directly provide intermediate feedback in a consistent way, and none of them directly supports intermediate control. This confirms the necessity of external means such as client-driven strategies when integrating VIS with R for visual exploration. On the upside, all surveyed algorithms fulfill the requirements of at least one client-driven strategy. The fact that there are good as well as bad examples shows that integrability lies at the hands of the single COMP developer, even without the availability of standardized communication protocols.

7 Discussion and Future Work

This paper is intended to show individual developers in the VIS and COMP communities practical measures of supporting integrability on their end. We agree with previous work [15] that the development of algorithms that directly provide standardized communication would be highly desirable in this context, as it allows reuse and minimizes effort for the VIS community. However, agreeing on protocol standards and implementing them for existing P is tedious, and putting the full load on the shoulders of COMP developers is not reasonable. The client-driven application of S1-S4 can be seen as a practicable alternative that allows VIS developers to achieve user involvement for a large number of existing implementations. Adhering to interface requirements in favor of client-driven integration is a more manageable first request to COMP developers than providing *perfectly accessible algorithms*.

Fekete has identified two key limitations of current integrations between VIS and COMP for the purpose of exploration [15]: First, "*algorithms provided by analytical environments are not designed for exploration and make no effort in providing early results quickly to the analyst*". Our paper directly addresses this issue, as the characterized strategies and resulting guidelines pave the way for tighter integrations that support user involvement during computations. As the second issue, Fekete states that "*when data is large [...] transfer time itself exceeds the reactivity requirement*" [15]. This issue is further aggravated by the exchange of intermediate signals. However, many forms of intermediate communication are substantially smaller than the regular inputs or outputs of P, e.g., the cluster centers in KMEANS. Apart from data size, the severity of this limitation in practice depends on infrastructural aspects of the integration that are beyond the scope of this paper. Examples include *network-based* vs. *memory-based communication*, *same machine* vs. *different machine in LAN / Internet*, *stateless* vs. *workspace-based COMP*, *internal data source* vs. *tertiary database*, as well as *overheads incurred the internal data format of COMP*. As our discussion does not cover these aspects as such, we demonstrate in the following that the presented strategies and integration scenarios can work for moderately large datasets.

As an initial proof of concept, we implemented four common integration scenarios by connecting our VIS environment *Visplore* [30, 35, 36] to R and MATLAB: We integrated Visplore with (1) the c-based R-API as part of the Visplore process [24], (2) the COM interface of the MATLAB engine in a different process (3) the RServe package via TCP running on the same PC[7] as Visplore, and (4) RServe running on a different PC[8] via Gigabit LAN. Table 2 reports timings of transferring arrays of randomized double precision values from VIS to COMP. Timings for the other direction, i.e., COMP to VIS, were equivalent in this measurement. As a second experiment, we implemented the client-driven versions of S1 and S4 for the R-method `kmeans` based on the local API integration of R. The input of a 20-dimensional table of random data records is transferred to the R-workspace once, while cluster labels for each record are returned to Visplore after every step $\tilde{P}_i$. Table 3 states average timings of early result availability for varying numbers of data records (S1) as well as percentages of the full iteration count (S4), for $k = 20$ clusters. The intention of these tests is to show that data transfer can be sufficiently fast for data sizes commonly found in real world analyses. Especially in local integrations, computation times are often the more limiting factor.

Array size	R API, local	MATLAB, local	RServe, local	RServe, LAN
100 MB	0.017s	0.254s	0.476s	0.883s
1024 MB	0.171s	2.663s	5.018s	8.510s

Table 2. Timings of transferring arrays of double precision random values between Visplore and COMP environments using different integration scenarios. Measurements were averaged across 10 repetitions.

While most examples in this paper stem from the field of multivariate analysis, the discussed TUI and strategies are generalizable

Number of rows	2 iterations	5 iterations	10 iterations	20 iterations
20,000	0.105s	0.256s	0.490s	0.845s
200,000	1.594s	4.041s	8.728s	19.256s
2,000,000	19.744s	59.244s	128.209s	291.869s

Table 3. Timings of the availability of $\tilde{r}_{\tilde{P}_i}$ in Visplore when executing `kmeans` on a 20-dim. dataset of random numbers for $k = 20$ clusters in R. Visplore and R are executed on the same PC using the c-based API of R. Measurements were averaged across 10 repetitions.

to many algorithms in other disciplines. However, there are contexts where users will not consider all TUI as desirable. While result control might counteract the reproducibility of results in some cases, early result feedback might as well be unfamiliar to users that are accustomed to "seeing precise figures" [17]. In such cases, the potentially large overheads of executing additional steps $\tilde{P}_i$ might be particularly painful if these resources could have been used to execute P as a black box more quickly. Finally, approximate solutions often introduce the need for explicit encoding of incompleteness and uncertainty, which increase the complexity of drawings and may even confuse users unfamiliar with such techniques.

We see multiple directions for future work: (1) We plan to implement client-driven integration strategies for different algorithms within Visplore, in order to evaluate them in the context of real-world tasks. (2) While our discussion of realizing client-driven strategies assumed the client to be VIS, we intend to investigate implementing it as an autonomous piece of reusable middleware. (3) To enable a more general assessment of the applicability of strategies, we also intend to extend our survey to include additional COMP environments like MATLAB or Python.

8 Conclusion

In this paper we characterized possibilities of achieving a tight integration between computational environments and visualization software. We laid the ground by a structured characterization of needs for user involvement in ongoing computations. Based on this classification, we formalized and described strategies to realize these needs for algorithms of different characteristics. A detailed discussion of considerations for client-driven and algorithm-driven implementations enabled us to identify guidelines to algorithmic interfaces which we evaluated based on a survey of common algorithms of the software R.

The combination of automated analysis techniques with interactive visualization is the key idea of Visual Analytics [25]. In this sense, we see our work as contribution on multiple levels. On a theoretical level, the formalization and comparison of technical strategies to achieve user involvement is a contribution to the theoretical foundations of Visual Analytics. On a practical level, we believe that the described implementation considerations facilitate an adoption for numerous integration scenarios based on existing computation environments. On a community level, we hope that the identification of specific requirements and guidelines for client-driven and algorithm-driven implementations fosters the development of computational infrastructures which are better suited to the needs of visual exploration.

Acknowledgments

This work has been supported by the Austrian Funding Agency (FFG) within the scope of the programs COMET K1 and ICT of the Future (project nr. 840232), as well as the FWF-funded project P24597-N23 (VISAR). Thanks go to Clemens Arbesser for valuable discussions, and to Johanna Schlereth for help with the figures.

References

[1] M. Angelini and G. Santucci. Modeling Incremental Visualizations. In *Proc. of the EuroVis Workshop on Visual Analytics (EuroVA '13)*, pages 13–17. Eurographics Association, 2013.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[3] E. Bertini and D. Lalanne. Investigating and Reflecting on the Integration of Automatic Data Analysis and Visualization in Knowledge Discovery. *ACM SIGKDD Explorations Newsletter*, 11(2):9–18, 2010.

[7] Windows PC, Intel Xeon E3-1245 V2 CPU @ 3.4 Ghz, 16GB RAM

[8] Windows Notebook, Intel i7-3612QM CPU @ 2.1 Ghz, 8GB RAM

[4] E. Bertini and G. Santucci. Give Chance a Chance: Modeling Density to Enhance Scatter Plot Quality Through Random Data Sampling. *Information Visualization*, 5(2):95–110, 2006.

[5] E. Bertini, A. Tatu, and D. Keim. Quality Metrics in High-Dimensional Data Visualization: An Overview and Systematization. *IEEE Trans. on Visualization and Computer Graphics*, 17(12):2203–2212, 2011.

[6] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.

[7] E. Brown, J. Liu, C. Brodley, and R. Chang. Dis-Function: Learning Distance Functions Interactively. In *Proc. of the IEEE Conference on Visual Analytics in Science and Technology*, pages 83–92. IEEE, 2012.

[8] S. Callahan, L. Bavoil, V. Pascucci, and C. Silva. Progressive Volume Rendering of Large Unstructured Grids. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):1307–1314, 2006.

[9] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, page 181–186. ACM, 1991.

[10] J. Choo, C. Lee, C. Reddy, and H. Park. UTOPIAN: User-Driven Topic Modeling Based on Interactive Nonnegative Matrix Factorization. *IEEE Trans. on Visualization and Computer Graphics*, 19(12):1992–2001, 2013.

[11] R. Crouser and R. Chang. An Affordance-Based Framework for Human Computation and Human-Computer Collaboration. *IEEE Trans. on Visualization and Computer Graphics*, 18(12):2859–2868, Dec. 2012.

[12] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2004.

[13] A. Dix and G. Ellis. by chance: enhancing interaction with large data sets through statistical sampling. In *Proc. of the Conference on Advanced Visual Interfaces (AVI '02)*, page 167–176. ACM, 2002.

[14] A. Endert, C. Han, D. Maiti, L. House, S. Leman, and C. North. Observation-Level Interaction with Statistical Models for Visual Analytics. In *Proc. of the IEEE Conference on Visual Analytics in Science and Technology (VAST '11)*, pages 121–130. IEEE, Oct. 2011.

[15] J.-D. Fekete. Visual Analytics Infrastructures: From Data Management to Exploration. *Computer*, 46(7):22–29, 2013.

[16] D. Fisher. Incremental, Approximate Database Queries and Uncertainty for Exploratory Visualization. In *Proc. of the IEEE Symp. on Large Data Analysis and Visualization (LDAV '11)*, pages 73–80. IEEE, 2011.

[17] D. Fisher, I. Popov, S. Drucker, and mc schraefel. Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*, page 1673–1682. ACM, 2012.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[19] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual Comparison for Information Visualization. *Information Visualization*, 10(4):289–309, 2011.

[20] H. Griethe and H. Schumann. The Visualization of Uncertain Data: Methods and Problems. In *Proc. of Conference Simulation and Visualization (SimVis '06)*, page 143–156, 2006.

[21] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large Complex Data: Divide and Recombine (D&R) with RHIPE. *Stat*, 1(1):53–67, 2012.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, volume 1. Springer, 2nd edition, 2009.

[23] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive Data Analysis: The CONTROL Project. *Computer*, 32(8):51–59, 1999.

[24] J. Kehrer, R. N. Boubela, P. Filzmoser, and H. Piringer. A Generic Model for the Integration of Interactive Visualization and Statistical Computing using R. In *Poster Proc. of the IEEE Conference on Visual Analytics Science and Technology (VAST '12)*, page 233–234. IEEE, 2012.

[25] D. A. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering The Information Age - Solving Problems with Visual Analytics*. Eurographics, 2010.

[26] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. Visual Analytics: Scope and Challenges. In *Visual Data Mining*, number 4404 in Lecture Notes in Comp. Science, pages 76–90. Springer, 2008.

[27] D. A. Keim, F. Mansmann, and J. Thomas. Visual Analytics: How Much Visualization and How Much Analytics. *SigKDD Explorations*, 2009.

[28] Z. Liu, B. Jiang, and J. Heer. imMens: Real-Time Visual Querying of Big Data. *Computer Graphics Forum*, 32(3pt4):421–430, 2013.

[29] M. J. McGuffin and I. Jurisica. Interaction Techniques for Selecting and Manipulating Subgraphs in Network Visualizations. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):937–944, 2009.

[30] T. Mühlbacher and H. Piringer. A Partition-Based Framework for Building and Validating Regression Models. *IEEE Trans. on Visualization and Computer Graphics (VAST '13)*, 19(12):1962–1971, 2013.

[31] E. J. Nam, Y. Han, K. Mueller, A. Zelenyuk, and D. Imre. ClusterSculptor: A Visual Analytics Tool for High-Dimensional Data. In *Proc. of the IEEE Symp. on Visual Analytics in Science and Technology (VAST '07)*, pages 75–82, Oct. 2007.

[32] J. Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[33] F. Olken and D. Rotem. Random Sampling from Database Files: A Survey. In *Statistical and Scientific Database Management*, number 420 in Lecture Notes in Comp. Science, pages 92–111. Springer, 1990.

[34] C. Olston and J. D. Mackinlay. Visualizing Data with Bounded Uncertainty. In *Proc. of the IEEE Symp. on Information Visualization (InfoVis'02)*, page 37–. IEEE, 2002.

[35] H. Piringer, W. Berger, and J. Krasser. HyperMoVal: Interactive Visual Validation of Regression Models for Real-Time Simulation. *Computer Graphics Forum (EuroVis '10)*, 29(3):983–992, 2010.

[36] H. Piringer, C. Tominski, P. Muigg, and W. Berger. A Multi-Threading Architecture to Support Interactive Visual Exploration. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):1113–1120, 2009.

[37] J. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Advances in Kernel Methods - Support Vector Learning*, 1998.

[38] O. Rübel, S. V. E. Keränen, M. Biggin, D. W. Knowles, G. H. Weber, H. Hagen, B. Hamann, and E. W. Bethel. Linking Advanced Visualization and MATLAB for the Analysis of 3D Gene Expression Data. In *Visualization in Medicine and Life Sciences II*, Mathematics and Visualization, pages 265–283. Springer, 2012.

[39] S. Roweis. EM Algorithms for PCA and SPCA. In *Proc. of the Conference on Advances in Neural Information Processing Systems*, page 626–632. MIT Press, 1998.

[40] T. Schreck, J. Bernard, T. Tekusova, and J. Kohlhammer. Visual Cluster Analysis of Trajectory Data with Interactive Kohonen Maps. In *Proc. of the IEEE Symp. on Visual Analytics in Science and Technology (VAST '08)*, pages 3–10. IEEE, 2008.

[41] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual Parameter Space Analysis: A Conceptual Framework. *IEEE Trans. on Visualization and Computer Graphics (cond. accepted for InfoVis)*, 2014.

[42] B. Shneiderman. Dynamic Queries for Visual Information Seeking. *IEEE Software*, 11(6):70–77, 1994.

[43] M. Streit and O. Bimber. Visual Analytics: Seeking the Unknown. *Computer*, 46(7):20–21, 2013. Guest Editors' Introduction.

[44] D. Temple Lang and D. F. Swayne. GGobi Meets R: An Extensible Environment for Interactive Dynamic Data Visualization. In *Proc. of the 2nd International Workshop on Distributed Statistical Computing*, 2001.

[45] J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE, 2005.

[46] C. Turkay, F. Jeanquartier, A. Holzinger, and H. Hauser. On Computationally-Enhanced Visual Analysis of Heterogeneous Data and its Application in Biomedical Informatics. In A. Holzinger and I. Jurisica, editors, *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics*, number 8401 in Lecture Notes in Computer Science, pages 117–140. Springer Berlin Heidelberg, Jan. 2014.

[47] S. Urbanek. No Need to Talk to Strangers - Cooperation of Interactive Software with R as Moderator. In *Proc. of the Symp. of the Interface of Computing Science and Statistics*, 2002.

[48] S. van den Elzen and J. van Wijk. BaobabView: Interactive Construction and Analysis of Decision Trees. In *Proc. of the IEEE Symp. on Visual Analytics Science and Technology (VAST '11)*, pages 151–160. IEEE, 2011.

[49] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[50] M. Williams and T. Munzner. Steerable, Progressive Multidimensional Scaling. In *IEEE Symp. on Information Visualization, 2004. INFOVIS 2004*, pages 57–64, 2004.

[51] P. C. Wong, H. Foote, D. Adams, W. Cowley, and J. Thomas. Dynamic Visualization of Transient Data Streams. In *Proc. of the IEEE Symp. on Information Visualization (InfoVis '03)*, pages 97–104. IEEE, 2003.

[52] J. S. Yi, Y. a. Kang, J. Stasko, and J. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '07)*, 13(6):1224–1231, 2007.