An Interactive Workflow Visualization for Biomedical Processing Pipelines Integrated into the Refinery Platform

Stefan Luger*
Johannes Kepler University Linz, Austria

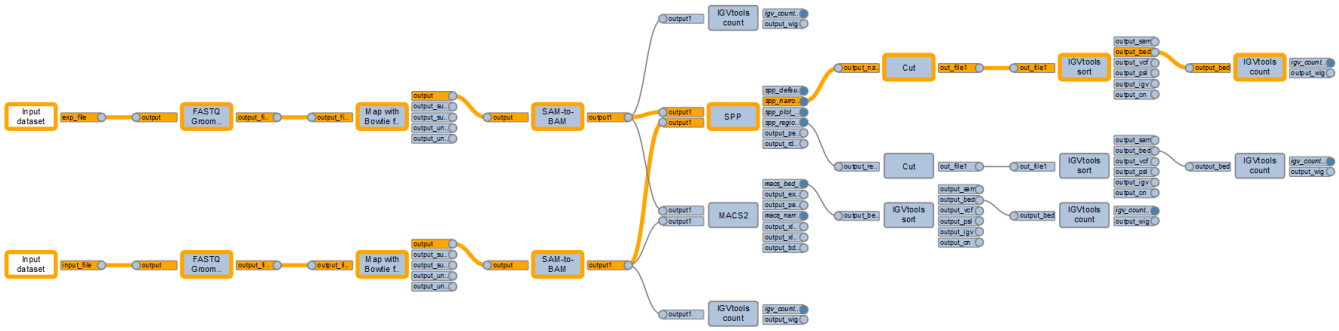

Figure 1: *This workflow visualization illustrates a biomedical processing pipeline in the domain of cancer research. The visual building blocks for this visualization comprise nodes (representing processing tools) and links (connecting these tools to each other). This visualization has its in- and output files (left and right to nodes) expanded to provide detailed information on demand. A clean layout, basic interaction techniques (e.g. drag, zoom, pan, etc.) or even more advanced features like path highlighting, let analysts examine interesting parts within the processing pipeline while keeping an overview on it.*

Abstract

In the field of biomedical research, domain experts have to manage the complexity of heterogeneous and large data processed within analysis pipelines. Their goal is to get insights through (intermediate) results conveyed by the workflows representing the pipelines. Obviously, when analyzing biomedical data (such as blood samples taken from patients), the information is stored in raw text formats and often too big to be handled. Scientific workflows consist of tools with each of them defining their own experiment execution parameters. Because of the vast amount of data being processed, simply labeling the tools of a workflow (*Nodes*) and the IO-file stream connecting nodes (*Links*) would cause too much textual information. The *Refinery Platform (Refinery)* - developed by the Park Lab in collaboration with the Hide Lab in Harvard - aims to collect, process and most importantly visualize biomedical workflows. Their former visualization was based on a Python-based generated image which does not fulfill its initial purpose any more. In this report, we present a workflow visualization based on the *Data-Driven Documents (D3.js)* JavaScript library that provides level of detail, encodes multidimensional information through compact glyph design and provides necessary interaction features to satisfy the needs of domain experts working with biomedical data. The analyst's requirements are then formalized as user tasks. These tasks are looked at concerning visual encoding and design choices as well as implementation and limitations. The workflow visualization is integrated into Refinery and showcased by the use of complex workflows (provided by the *Galaxy Project*). Cancer researchers who are working with processing pipelines on a regular basis will benefit from the visualization presented.

Keywords: Scientific Workflows, Scientific Visualization, Grid-based Graph Layout

1 Introduction

Domain experts in Genomics have to study and analyze a huge amount of data (e.g. genetic information - in this context mostly taken from patients) in order to determine a patient's risk for a specific cancer type. Conventionally this happens with the support of so called analysis pipelines where heterogeneous data (e.g. derived from tissue samples) is carried through the steps of analysis tools. Basically and very simply said, one could define a tool as follows: A tool's input files are processed by a variety of molecular biological analysis calculations. The outcome is then represented by one or more files again that are in turn the input for the next tool in the predefined pipeline. [Gehlenborg 2012; Gehlenborg 2013b; Gehlenborg 2013a]

This report focuses on the technical aspect of building an interactive workflow visualization operating on in- and output files within an analysis pipeline. In the end, the stereotypes reading this report are most likely more familiar in the domain of Computer than in Life Sciences. So we literally could define this processing step as a molecular biological data processing *Black Box* - or in the graph processing domain, we would simply use the term *Node* containing in- and output ports. *Links* connect these nodes, and in conjunction, a directed graph is shaped. To fit the behavior of an analysis pipeline, the graph governs the execution order of tools step by step as workflow. The execution of this workflow then provides insights to domain experts through analytical results. The motivation

*e-mail:stefan.luger@jku.at

for computer scientists in the fields of Bioinformatics is to develop a state of the art visualization which helps interpret the outcomes as well as the intermediate results during the workflow execution phase.

The *Refinery Platform*[1] or in short *Refinery* is a project which is currently developed by the Park Lab [2] at Harvard Medical School in collaboration with the *Hide Lab*[3] at Harvard School of Public Health that aims to collect, process as well as visualize heterogeneous data within biomedical Galaxy[4]-based workflows (see Figure 2). The workflows are manually created in Galaxy and available in the *JavaScript Object Notation (JSON)* format inside Refinery. The platform also serves as exchange hub among user groups where authorized users can share their workflows annotated with a vast amount of files of molecular biological samples including metadata.

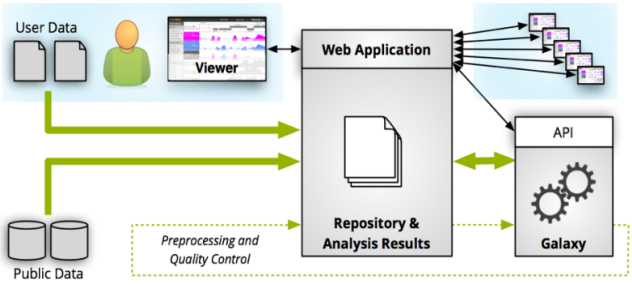

Figure 2: *Refinery - Galaxy - Workflow Visualization [Gehlenborg 2013b].*

Refinery still is in development phase and requires a new and state of the art workflow visualization, as their Python-based prototype does not satisfy the needs of their early user-base anymore. Carefully speaking, the former visualization was imported as an image and does not cover any interaction possibilities, level of detail, an appropriate layout - nor does it represent the actual workflow effectively. Figure 3 shows issues such as overlapping labels or the lack of different glyphs being used. [Gehlenborg 2012; Gehlenborg 2013b; Gehlenborg 2013a]

In this work, we propose an interactive workflow visualization integrated into the Refinery Platform operating on biomedical data stored in the JSON file format. The issues pointed out in the paragraph above are only a few among many. User-driven tasks have to be elaborated to consider these problems as well as to discover additional requirements.

1.1 User Tasks

Basically, tasks are categorized into five different groups: **T1** considers an *intuitive workflow representation*; **T2** handles *detailed information on demand*; **T3** justifies a clean *workflow layout*; **T4** is all about *interaction principles*; **T5** focuses on *examination of substructures* within the workflow. These tasks will guide the reader through the report and are referred to as capital letter **T** followed by the task identifier number.

T1: The processing pipeline (technically also referred to as *workflow*) should be represented as a directed node link diagram. Given the information on direction of links due to left-aligned node placement, analysts should have no problems interpreting the order of steps in the workflow.

T2.1: It is not desired to have every tool and its corresponding files visible permanently. In case the analyst wants to abstract the workflow on a level where only nodes and links are shown, visibility control for showing and hiding files must be given to the analyst.

T2.2: Text labels which would exceed the dimensions of their encapsulating shapes have to be cut and shown in tooltips in full length.

T2.3: Analysts do not want to deal with a possible large number of tool parameters inside the visualization. Show a complete overview of the selected node and its tool parameters outside the visualization canvas.

T3: For a clean overview, the workflow's layout should be grid-based to maintain a minimal distance between nodes and to point out the graph's topology more consistently.

T4.1: Analysts should be able to reposition nodes within the visualization canvas. Because the workflow layout is generated automatically, nodes might not be placed ideal for every possible graph topology. Therefore, the analyst should be able to compensate for it manually.

T4.2: For very large workflows, allow analysts to get detailed information on demand while always keeping an overview and resetting their view.

T4.3: Analysts want to be able to switch back and forth between the originally created layout in Galaxy and the automatically generated Refinery layout.

T5: Analysts want to focus on a particular path in the processing pipeline to get insights on involved tools and files.

Before we look at similar work in Section 2, the report's continuing structure is given. Section 3 gives a brief introduction about the project's programming environment and the integration into the Refinery Platform. Section 4 elaborates visual encoding through glyph design. In Section 5 we deliver some insights concerning the task implementation as well as limitations. To conclude, the visualization as a whole gets presented. Section 6 sums up the report and discusses future work.

2 Related Work

2.1 Scientific Workflow Visualization

The success of a good visual representation for workflows heavily depends on using the appropriate shapes in conjunction with position, size, color, value, orientation and even texture (referred to as *visual variables*, and in combination as *glyph*). We review one approach and one system which significantly focuses on visualizing scientific workflows.

Maguire et al. [Maguire et al. 2012] illustrate the necessity of generating taxonomy based glyphs to encode multidimensional information. In particular, the categorization step is automated and each dimension maps to a different glyph design. Whereas the approach of automatic categorization and generation would go beyond the scope of this work, a mapping of similar visualization elements to related visual variables is indeed of importance. Anchor elements, which provide interaction tasks, clearly should stand out among other elements to make them the obvious choice for interaction tasks.

[1] http://refinery-platform.org/.
[2] http://compbio.med.harvard.edu/
[3] http://refinery-platform.org/
[4] http://galaxyproject.org/

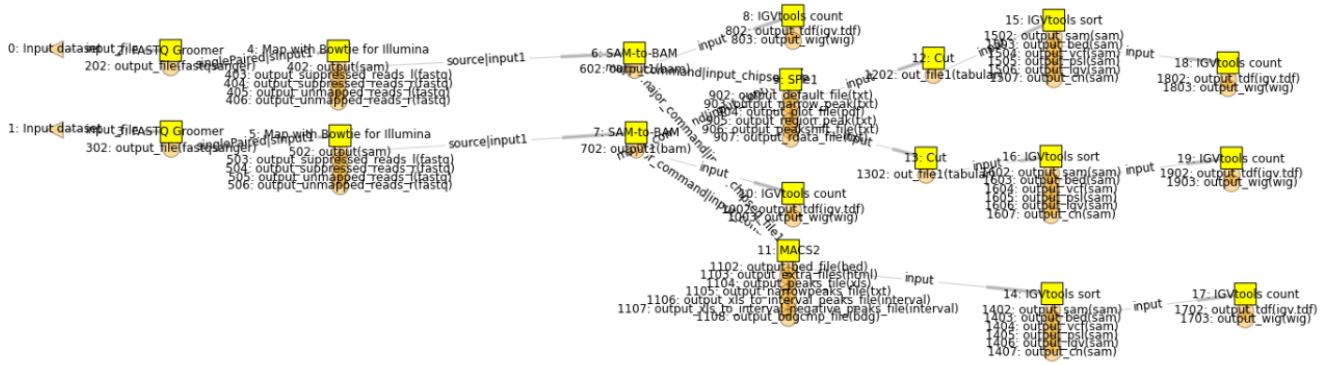

Figure 3: *Former Python-based Workflow Visualization was loaded and displayed in the format of Portable Network Graphics (PNG) [Gehlenborg 2014].*

Galaxy editor is a workflow creation framework within the Galaxy framework. Concerning our work, it provides first hints on important elements for our workflow design. In particular, elements such as nodes, links and files - which are the basic building blocks of the workflow - are labeled and shown. We want to encode a part of the information into distinct elements to hide some of the initial complexity when examining the structure of the workflow itself. Only if one is interested in a higher level of detail, one should be able to trigger additional visual elements on demand. [The Galaxy Team 2014]

2.2 Scientific Workflow Systems

In this work, a workflow visualization w.r.t. to the Galaxy format was built. Similar to Galaxy, many scientific workflow systems emerged over the past decade. Besides introducing Galaxy in Section 3.2, we want to take a brief look onto its competitors or predecessors.

In addition to the visualization of workfows, such systems like *Kepler* [Altintas et al. 2004] (created in 2002) provide accessible workflow creation tools, execution, monitoring support and record tool parameters over time for potential reruns to preserve reproducibility. Storing meta-data (provenance data) for large, complex and heterogeneous data requires a scalable and performance-oriented infrastructure and data model. Kepler builds upon nodes and links - exactly as a workflow is visualized in this work, where the former are referred to as actors and the latter as channels. Kepler also supports distributed execution over the web and integration of WSDL-defined web services.

In 2004, the first version of *Taverna* [Missier et al. 2010] was launched. Driven by the goals which led to the creation of Kepler, it is best known for its application to many sub-domains of the Life Sciences. Similar to Kepler, a workflow in Taverna consists of processors connected by data dependencies links. It is capable execution, storing provenance data and integrating web services as well. Even *R*, *Java* or sub-workflows can be interpreted. To extend Kepler's scalability, Taverna creates concurrent threads via the *Parallelise* layer. It is licensed under the *GNU Lesser General Public License (LGPL)*, hence community contributions do not come short. The creation of OnlineHPC[5] serves as example, making an accessible workflow creation editor available to the user base.

The last well know workflow management system we discuss is VisTrails [Bavoil et al. 2005]. It was developed in 2007 and so the latest among the systems introduced in this section. The key to VisTrail is its *visualization trail*, which stores provenance data about steps executed in the visualization pipeline. The loose coupled architecture separates the execution instance and the pipeline specification. Redundant operations are cached which save significant processing power when needed to run again (e.g. to compare multiple views of visualizations). On integrating VisTrails in various visualization applications, it can point out different parameter configurations over time in visualization pipelines. This approach is based on the visualization being generated via a step-by-step workflow itself and therefore is more uniform and different than the systems mentioned above. Concerning this work, the idea of a graph-based visualization pipeline and the ability to reuse cached assets can speed up the rendering process dramatically when dealing with very large amounts of links and nodes.

In Section 1 we have introduced Refinery and the goals it aims to achieve. In this section, we took a deeper look into scientific workflow systems and mentioned Galaxy, which we will explain in more detail in the next section.

3 Environment and Architecture

Now, an overview of the components surrounding the proposed visualization will be given (see Figure 4).

3.1 Refinery

Technically, Refinery is a client server system that manages repositories for imports as well as exports. The biomedical samples stored in these repositories are associated with in- and output files in the processing pipelines (workflows) imported via a Galaxy connector instance.

3.2 Galaxy

Looking at Figure 4, two logical pillars are left to discuss. Usually, workflows are created manually by domain experts. In Section 1 we have looked at the node link representation those workflows are mostly visualized and also stored as. A node represents a processing tool within the pipeline analysis. Normally, the author of a workflow makes use of differently behaving tools stored in a workflow generation framework or suite and finally assemblies them together via links - representing the file transactions alongside the pipeline. On the one hand, further in-depth processes inside a tool (we recall the molecular biological data processing *Black*

[5] http://onlinehpc.com/site/main

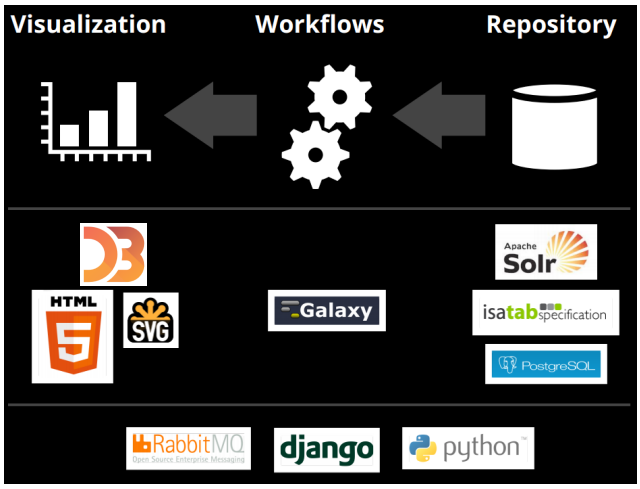

Figure 4: *The technical view on the environment is separated into three logical components: Refinery Visualization, Galaxy Workflows and Data Repositories [Gehlenborg 2013b].*

Box) are basic knowledge of domain experts in molecular biology, whereas the technical aspect of creating and integrating those tools into a common framework describes the task of computer scientists. Galaxy aims to combine those two worlds by providing a web interface for performing accessible, reproducible and transparent genomic science. It enables analysts without computer programming skills to build workflows within the editor. As Galaxy is built upon the idea of creating custom and sharing tools (referred to as *Galaxy Tool Shed*), workflow authors are able to take advantage of community created open source content. [Goecks et al. 2010; Blankenberg et al. 2001; Giardine et al. 2005]

Finally, Refinery connects to Galaxy via an API specification and integrates the created workflow data.

3.3 Repositories

The third and most right component in Figure 4 comprises repositories for raw data storage and query operations. *Apache Solr*[6] is used as full-text index and search engine. *ISA-Tab*[7] handles additional meta-data and tracking information for provenance aspects (when data changes over time). In particular, provenance data are changes on in- or output files, workflow topology or tool parameters configuration. *PostgreSQL*[8], designed for the high amount of data, manages Refinery's data storage and query tasks.

3.4 D3.js

For the workflow visualization part, Refinery most importantly provides support for *Data-Driven Documents (referred to as D3.js or in short D3)* [9] which is a web-based DOM-manipulating JavaScript library and very suitable for visualizing directed graphs over the web.

Originally it was developed by Mike Bostock in 2011 [Bostock et al. 2011]. Since then, it continuously evolved, an active user base emerged and therefore became more and more popular within the domain of Information Visualization on the web. More accurately, D3 manipulates *Scalable Vector Graphics (SVG)* elements and their attributes based on selection statements. Encapsulated into the *Hypertext Markup Language (HTML)*, combined with *Cascading Style Sheets (CSS)* and *Bootstrap* for modular design, Refinery provides the latest front end technology required for D3. Workflow data itself is exported in the JSON format via the Galaxy API, for which D3 offers a callback method to process. Additionally, one of the key aspects of D3 is the rich feature set it offers for visualization purposes. From canvas scaling, zoom and pan behavior to predefined layouts, D3 facilitates important visualization development tasks into a coherent and modular library. As layouts were mentioned, D3 maps link and node data structures automatically through referencing the node identifiers into the link data structure. On top of this, event handling supports interaction tasks on SVG elements conveniently. With all that being said, D3 is the obvious choice to fulfill the requirements specification of this practical work. Of course, other visualization libraries exist (e.g. Prefuse[10]) but they mostly are not applicable to the web environment or are not as flexible as D3 is. [Bostock 2014]

If preferred, all above components excluding Galaxy are then wrapped into a single environment containing a *Virtual Machine* via Vagrant [11] For more information about setting up Refinery please visit the Refinery documentation page[12].

The report's major focus lies on documenting the practical work done concerning visual encoding, design choices and the development process w.r.t. to best practices in the domain of Software Engineering. In the next two sections, we will first discuss the design and second the implementation process focused on the user tasks **T1 - T4** specified in Section 1.1.

4 Visual Encoding

Generally speaking, visual design choices were strongly influenced by the feedback from advisors during weekly one hour long progress meetings. Given the fact that not only the perspective of software engineers but also that of domain experts in the field of Computational Genomics were involved, shortcomings considering usability and interactivity aspects were tried to be minimized.

T1: We have introduced the characteristics of scientific workflows and their components in Section 1. In [Maguire et al. 2012] glyphs are presented as one of the most relevant elements when it comes to workflow visualization development. During the early concept design stage, a node was layouted as a simple rectangle containing a label for its tool name. An example of an input node is shown in Figure 5.

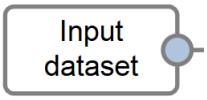

Figure 5: *Collapsed node design (rectangle) containing a name label in the center and an anchor (circle) representing the input dataset file.*

In general, a node is categorized of type *input* only then, if there are no predecessors to the node - the node actually does not represent a process by a tool itself. Moreover, it represents the main input dataset for the workflow, hence it is visualized without a filled

[6] http://lucene.apache.org/solr/
[7] http://isatab.sourceforge.net/
[8] http://www.postgresql.org/
[9] http://d3js.org/.
[10] http://prefuse.org
[11] http://www.vagrantup.com/.
[12] http://refinery-platform.readthedocs.org/

background. On the contrary, Figure 6 illustrates the simple node design for usual node types in a workflow.

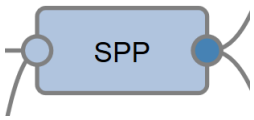

Figure 6: *Collapsed node design (rectangle) containing node label located in the center and an anchor (circles) representing in- and output files for the node positioned left and right.*

Having discussed the different node types, indeed one might argue to visualize the input node as a single file similar to the nodes containing in- and output files. With respect to a link connecting exactly two nodes, this is not easy to achieve, as the name of the input file usually matches to the input connection file. This only applies to links between nodes, which at least have one predecessor. So in this exceptional case, it could misguide the user - thinking both file names connecting both nodes are literally the same file. Therefore and because of the fact when the visualization is zoomed out - which results in smaller rectangles and makes it more difficult to interact with (e.g. dragging) - a rectangle the same size as of the other nodes was applied.

Before we further extend the node layout, *Links* are left to discuss. Links regularly connect exactly two nodes and are drawn as quadratic bezier curves. In comparison to straight lines, this results in a cleaner view of the graph as a whole. Whereas straight lines favor distinguishing connections for possible line crossings, bezier curves enhance the perception of node order and the general direction in workflows.

T2.1: On adding in- and/or output files to the node (at first within the boundaries of the node), available space became a major issue. Although in [The Galaxy Team 2014] files were encapsulated into the node too, we decided to add additional expand and collapse interaction to the node element and move the file representation outside to the left and right boundaries of a node. As a result, a collapsed node looks clean and does not require that much space as if all files were visible permanently (see Figure 5). In return, Figure 7 shows the node with its expanded file representation on the right side.

Figure 7: *Expanded node design (rectangle) showing the file in collapsed view with the anchor moved to the right of the actual file (slim rectangle).*

So far the circles positioned left or right to the node were omitted. In Figure 6, a node in its collapsed state is shown and introduces another new design: First, the background is filled lightsteelblue to better distinguish normal nodes from input nodes. Second, the circle to the right is filled steelblue, whereas the left one lightsteelblue. Circles (also referred as *anchors*), signalize in- and output files (left and right) and can be expanded through a click event on the corresponding anchor - the anchor then replicates itself and moves to each file shown (see Figure 8). The other way around, the file rectangles are removed and the original anchor is faded in again. Specific output files, which are imported back to Refinery, are demonstrated as steelblue anchors. Those files are renamed (stylized in italic) to avoid naming conflicts in the whole workflow scope.

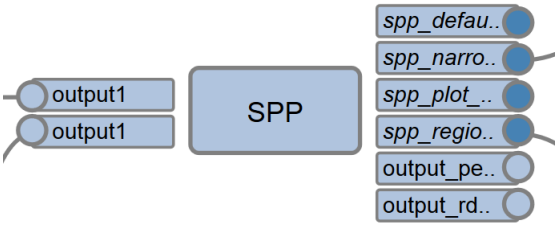

Figure 8: *Expanded node design showing opened in- and output files (slim rectangles) with their own file name label as well as dynamically generated anchors. Original anchors are hidden and disabled during the detailed view.*

T2.2: In order to help with additional on-demand information, tooltips are added. In particular, a tooltip is required either for long text occupying to much space w.r.t. to their encapsulating element dimension or for renamed output files mentioned above. Whereas Figure 9 and 10 show ordinary information about the cut text labels, in Figure 11, the original name of the output file - which is then imported back to Refinery - is listed in the third line.

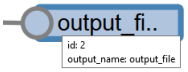

Figure 9: *Tooltip on hover interaction provides information about* `id` *and* `name` *for the input connection between the predecessor and current node.*

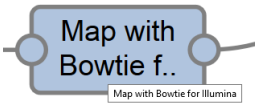

Figure 10: *Tooltip on hover interaction shows the full string of the node name.*

T2.3: Additional information about a tool in the workflow is available in the HTML table below the visualization canvas. Displaying all the information within the visualization canvas would cause overlapping with the workflow itself. Therefore, a table containing additional tool parameters (see Figure 12) is placed below the canvas. As the HTML page will grow in height, due the table occupying space, one should be able to show and hide it through a toggle behavior.

T3: Although the Galaxy-imported workflow comes with a predefined layout as shown in Figure 13, we decided to design a grid-based graph layout. The Galaxy layout occasionally arrange nodes in an unnatural way by not lining up the nodes linearly, although they seem to be positioned in one straight line. Nodes arranged to imaginary cells in the layout grid prevent these problems and further distribute the workflow uniformly in the visualization canvas. Visualization space has to be used up carefully. Especially when dealing with larger workflows, the not uniformly expanded Galaxy layout would result in a worse scaling factor applied when all nodes of a workflow should be visible on screen at once. More details on the algorithm and the implementation are explained in the corresponding task description in Section 5.

Frequently described as key to success for a good visualization is the presence of interactive tasks to be able to perform with.

TOGGLE SELECTION DETAILS »

Annotation	output_default_file	spp_default_file.txt	
	output_narrow_peak	spp_narrow_peak.txt	
	output_plot_file	spp_plot_file.pdf	
	output_region_peak	spp_region_peak.txt	
Name	SPP		
Tool ID	toolshed.g2.bx.psu.edu/repos/modencode-dcc/spp_package/modencode_peakcalling_spp/1.10.1		
Parameters	major_command	save_plot_file	True
		input_chipseq_file1	
		replace	True

Figure 12: *Upon node selection, a HTML table is generated automatically and placed below the visualization canvas. As tool parameters are of arbitrary hierarchy, nested tables will possibly get generated.*

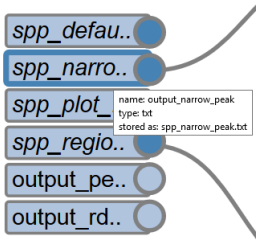

Figure 11: *Tooltip on hover interaction presents additional attribute information for attributes* `name`, `type` *and* `stored as`.

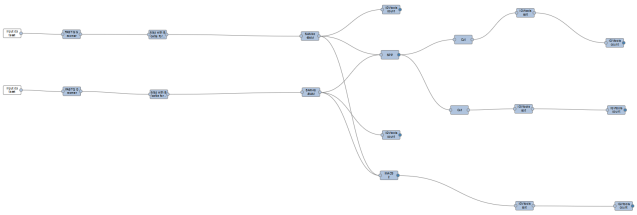

Figure 13: *Workflow Visualization based on the given Galaxy coordinates.*

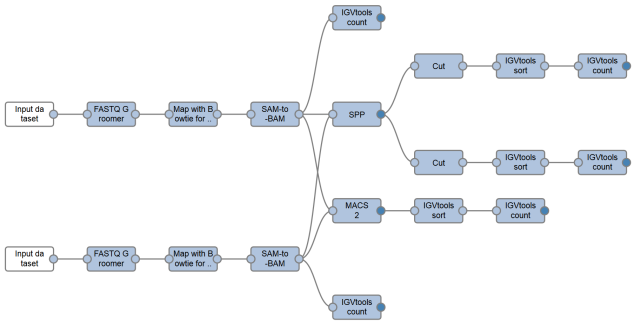

Figure 14: *Collapsed workflow visualization based on the Refinery grid layout.*

T4.1: As the layout is generated automatically, it certainly can not be optimized to consider every possible corner case for graph topologies. Analysts want to have the opportunity to manually perform changes in order to increase or decrease space between nodes. The solution to this requirement demands the ability to drag nodes. No special visual design choices to nodes have to be applied. The rectangular shape and fair size of nodes suits this task well.

T4.2: The visualization must provide general pan and zoom support. With pan functionality, one can handle very large workflows while not displaying every node on screen at once. Nevertheless, when a large workflow is loaded, the workflow should be scaled to the available window space to give an initial overview. If this behavior is not anticipated, an option to zoom in or out must be provided.

In Information Visualization, the zoom operation is categorized into two different behaviors. *Geometric zoom* is a widespread solution, where zooming only alters the scale of the canvas in focus. All elements inside the focus are then resized relatively to the current scale factor. But as a result, text labels might get distorted or in general, the SVG elements can get to small. A more sophisticated solution is described by *semantic zoom*: Elements in the visualization might not correspond to the current scale factor. This idea can be exploited to display detailed information according to a specific zoom level, while some elements preserve its size. A popular example is the implementation of *Google Maps* [13]. At a specific threshold, streets and their text labels appear in the visualization. Whereas countries grow in size, street paths and text labels maintain their stroke width.

As semantic zoom is not required for this work, geometric zoom suffices.

T4.3: If desired, one can switch between Refinery and Galaxy layout. The Galaxy layout reuses the visual assets and only differs in

[13] `https://maps.google.com/`

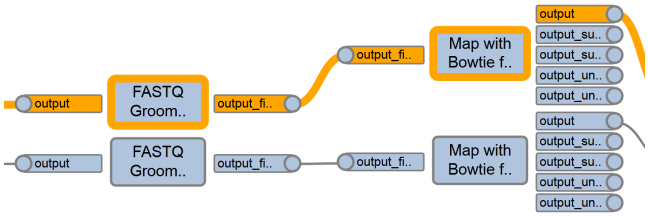

Figure 15: *The visualization provides interactivity through zooming, panning, dragging single nodes and highlighting a path via node selection. Quickly double-clicking the background screen scales the whole workflow to the canvas space. Path highlighting in detail: Selecting a node highlights the corresponding path beginning with the current node and selecting all nodes and links involved via backpropagation to the input dataset.*

terms of its node placements.

T5: For analysts, it is essential to be able to examine intermediate processing results alongside a path in the whole pipeline. Such path can be highlighted on selecting a chosen node. Every node within the path then should be highlighted with high color contrast to the underlying workflow color style. The workflow visualization itself conforms to a harmonious blue color style. Highlighting dyes the path (including all predecessor branches) orange while increasing the stroke width of link and node borders.

Having designed and discussed the visual elements, we will move forward and look into interesting implementation details, criticize limitations and discuss adaptions made in order to still satisfy the requirements.

5 Implementation

Before we discuss the implementation concept in D3, it has to be mentioned that the visualization comprises two major visual components: the D3 visualization canvas and the HTML table. Both components are tied together within a workflow HTML page - generated via the *Django* template language within an *Python* environment.

Section 3.4 already introduced D3. In order to fulfill the requirements elicit based on tasks, the general concept in D3 for creating, updating or removing visual elements is followed. D3 binds (maps) data to visual elements (e.g. a dataset containing five different values result in five bars with variable height within a barchart). In this work, this dataset (not to get confused with datasets in processing pipelines) describes the workflow imported from Galaxy. The Galaxy workflow (which Refinery stores in JSON format) can be loaded via a callback function in D3. As D3 is a DOM-manipulating language, it does in fact create or alter SVG elements through selection statements. For more details, a valuable tutorial[14] by the author of D3, Mike Bostock himself, is referenced. But in short, the following four steps describe the main concept for adding, updating or removing SVG elements in D3 [Bostock 2014]:

1. The DOM elements to append to any SVG parent (e.g. `svg` as root element) are selected via a class name.
2. The `data(object)` function then defines the container to iterate through (e.g. object of any array type).
3. The `enter()` method specifies that the SVG elements appended are indeed being added. More possibilities for removal or update tasks are defined by the `exit()` and `update()` method. The former removes all elements selected, but only if there are existing elements selected in step 1. The latter updates DOM attributes of existing elements selected in step 1.
4. Finally, `append(element)` appends the SVG element `element` specified to the SVG parent (see step 1.) in each iteration of `object`.

Using the procedure above, elements were created and grouped - assembling the final node layout presented in the illustrated figures in Section 4. All node elements can be transformed via `translate()`, `scale()` and `rotate()` and are scaled to a fixed node width and height, which again depends on the current scale factor applied through zooming and responsiveness. [Bostock 2014]

Following the course of Section 4, we will focus on tasks to point out interesting implementation details and limitations.

T1: In the JSON workflow file, each `step` field represents a node. Further fields (e.g. `name` is used as the text label for a node) are parsed and stored in an internal node link data structure within the D3 callback function. Links are extracted by the file parameters `input`, `output` and `input_connections` which provide information about connecting nodes. Guided by the D3 SVG element creation principles, nodes are grouped and appended to a SVG root container. To preserve flexibility, element dimensions and margins are scale dependent to each other. Links consist of SVG paths shaping a bezier curve which are explained in detail in [Dahlström et al. 2014].

A challenging part of this task was to fit the text labels within the node shape with fixed length. As this issue corresponds to **T2.2**, we will discuss it here. Actually, D3 does not provide HTML stylized line breaks and therefore were coded manually. Minor constraints suggested to break labels at hyphens, whitespaces or after an fixed amount of characters. Hence there are no intelligent linebreaks on syllables. Nevertheless, tooltips cover the full text string. The same procedure applies to in- and output files discussed in the next task.

T2.1: A key aspect to provide the ability to expand or collapse either side of files, is to handle click events. In Section 4 anchors were introduced as circles. These SVG circle elements provide on click events to create replicated circles as well as in- or output files represented as rectangles. File rectangles are centered in y-direction of the node. In order to define the collapse event to the file anchors which are not generated yet, D3 allows to chain select the potential SVG element to make dynamic event handling possible.

T2.3: D3 is not suitable for displaying table-like content in its canvas, hence a HTML table seemed to be the best solution in order display detailed node parameter information below the visualization canvas. Each node has its table creation event applied. After deselecting or clicking on the empty canvas, the table is deleted again. A major challenge was to overcome the parsing task containing parameters in arbitrary JSON hierarchy. We solved this issue via recursive depth-first search (DFS) followed by appending nested HTML tables to a table parent for each new layer of a parameters.

T3: For the grid layout, graph metrics are calculated before separating the nodes into columns and rows (e.g. depth and width).

Figure 14 and 16 show the workflow visualized via grid layout algorithm. Each node belongs to a specific column and row resulting in a clean layouted workflow. If any node is expanded, the column width grows accordingly, even if nodes in the same column exist and are not expanded. This implementation has a minor drawback,

[14] `http://bost.ocks.org/mike/selection/`

as the workflow is not minimized in space anymore. But in contrast to the Galaxy layout, this approach results in significantly better partitioned node placement. We present the algorithm in pseudo code in Listing 1.

```
1.Iterate over column before row.
2.On branch:
  3.Check successor nodes.
  4.Grow of workflow width (vertically) -
      add rows to provide space.
  5.Distinct even or odd case.
  6.For n or (n + 1) rows:
    a) Add n/2 rows at row index - 1
    b) Add n/2 rows at row index + 1
```

Listing 1: *Grid-based graph layout.*

Another limitation of the algorithm in its current state shows each branch occupying an own row. In detail, the layout is processed column-wise from left to right. If the current node starts a new branch, depending on the successor nodes row index, a new empty row is added after the current row index. For the case of an odd amount of successor branches (e.g. three), the second branch would succeed in the same row as the node starting the new branch. The first branch is shifted by `row index - 1`, whereas the third branch is shifted by `row index + 1`.

For the sample workflow, another issue arises at the `SAM-To-BAM` node (fourth column). As the layout algorithm processes columns before rows, `SPP` and `MACS2` are visited already and taken into account by the top `SAM-To-BAM` node. Therefore, only bottom `IGVtools count` has to be rearranged. The more branches are starting at the right side of the workflow, the more the layout will span in height.

T4.1: D3 API offers drag and drop behavior via `dragstart`. As multiple events might belong to the same SVG element, in D3, it is necessary to prevent those not being fired with `d3.event.sourceEvent.stopPropagation();`

T4.2: D3 API only provides general zoom and panning support. For the zooming behavior, the control key has to be pressed, as one should be able to still scroll the window in height. We therefore extended the basic zoom support of D3. Zooming and panning then provides a flexible working area. Nevertheless, a double click event in the background area triggers the workflow to automatically fit to the window size. Additionally the visualization takes advantage of responsive behavior via Bootstrap CSS library.

T4.3: Radio buttons located below the tool parameters table were added to allow quickly switching between both layouts. On the downside, the layout to display is set via enumeration parameter. This solution unfortunately requires a new run of the D3 script.

T5: On node selection, unified with the table creation event, a recursive graph traversal algorithm is initiated. The algorithm obeys the laws of DFS (processed from right to left). It extracts every branch and at last returns a set of nodes and links. These are then dyed orange. During development, further experiments were tested, e.g. allowing the analyst to select multiple paths. Despite being useful in some cases, it might misguide some users and therefore was removed again.

5.1 The Big Picture

Combining the features above, a powerful visualization lets domain experts use a fair variety of interaction mechanics to showcase their area of interest concerning different aspects within the workflow. Despite presenting this work in the domain of biomedical research, the user base is able to load any dataset imported from Galaxy - not related to biomedical data necessarily.

6 Conclusion and Future Work

In this work, we presented a visualization for biomedical workflows within the Refinery Platform. Usability aspects and a large set of features focusing on interactivity and compact data representation were among the main goals achieved in this work. In addition, distinctive tasks considering visual encoding and implementation including limitations were discussed.

Analogous to similar work, showing the optimal level of detail is key to success of a good workflow visualization. When dealing with smaller workflows, this problem can be solved manually through glyph design in conjunction with the appropriate usage of visual variables. For larger workflows, the approach of Maguire et al. in [Maguire et al. 2012] automates the design of glyph creation through taxonomy and categorization, but still, the workflow has to fit onto various resolutions. It seems inevitable that a variable level of detail (semantic zoom as discussed in Section 4) is the grand solution to overcome this difficulties when visualizing data in detail on demand. In order to adapt to responsive window space, geometric zoom was implemented. Concerning semantic zoom, the node and link design cover expand as well as collapse mechanics to show and hide (in)significant information within the workflow.

For future work - as the workflow visualization is part of the upcoming release by the Refinery Team - improving stability and constantly fixing and or adding minor issues is of high importance. In particular, some enhancements to the visualization were discussed: The layout algorithm still could be improved considering links crossing each other in certain areas. Altering glyphs to indicate data flow direction, assign different colors for input node links or even encapsulate more node information into tooltips might be beneficial in some cases too.

Another long-term goal of the Refinery Team is to integrate an additional layer of meta information into the visualization: the view and interaction with provenance data of workflow execution. This is important, as provenance data opens possibilities like experimenting with changes to tool parameters, files or the workflow topology (e.g. adding or removing tools). Changes are then recorded at all stages over time which, in turn, preserves reproducibility of experiments. In [Begley and Ellis 2012], it is stated that scientists tried to confirm 53 published findings related to drug development research. They proposed that only 6 preclinical studies were able to be reproduced and confirmed. This example should motivate bioinformatics and software engineers to thrive forward provenance data in applications for Life Sciences.

The workflows created and executed in Galaxy are annotated with provenance data. Just like workflows, provenance data is imported to Refinery and stored in the ISA-Tab format [Rocca-Serra et al. 2010]. As further work, the workflow visualization could be extended by the addition of a time-varying view. This would promote the idea of reproducibility preservation. Performance and scalability issues could be solved through aggregation of similar topological graph structures [Maguire et al. 2013].

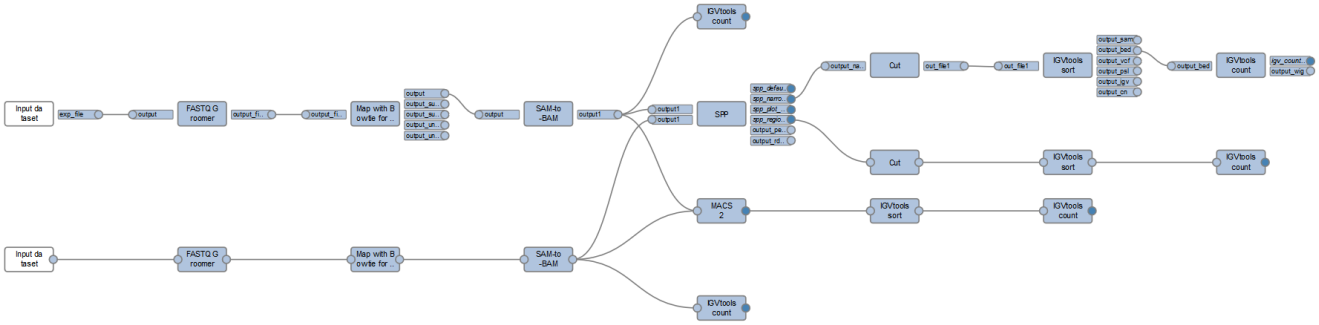

Figure 16: *This workflow visualization based on the Refinery layout partially contains expanded nodes and shows the approach of a variable width for columns inside the layout grid.*

Acknowledgements

I would like to take the opportunity to express my gratitude to Nils Gehlenborg, Marc Streit, Samuel Gratzl and Holger Stitz for their continued support during the weekly progress meetings as well as being available for questions and helping out with tips and tricks in their free time.

References

ALTINTAS, I., BERKLEY, C., JAEGER, E., JONES, M., LUDASCHER, B., AND MOCK, S. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the Conference on Scientific and Statistical Database Management*, Springer-Verlag, 423–424.

BAVOIL, L., CALLAHAN, S., SCHEIDEGGER, C., VO, H., CROSSNO, P., SILVA, C., AND FREIRE, J. 2005. VisTrails: enabling interactive multiple-view visualizations. In *Proceedings of the IEEE Conference on Visualization (VIS '05)*, IEEE Computer Society Press, 135–142.

BEGLEY, C. G., AND ELLIS, L. M. 2012. Drug development: Raise standards for preclinical cancer research. *Nature 483*, 7391, 531–533.

BLANKENBERG, D., KUSTER, G. V., CORAOR, N., ANANDA, G., LAZARUS, R., MANGAN, M., NEKRUTENKO, A., AND TAYLOR, J. 2001. *Galaxy: A Web-Based Genome Analysis Tool for Experimentalists*. John Wiley & Sons, Inc.

BOSTOCK, M., OGIEVETSKY, V., AND HEER, J. 2011. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (InfoVis' 10) 17*, 12, 2301–2309.

BOSTOCK, M., 2014. D3: Data-driven documents - api reference.

DAHLSTRÖM, E., DENGLER, P., GRASSO, A., LILLEY, C., MCCORMACK, C., SCHEPERS, D., WATT, J., FERRAIOLO, J., JUN, F., AND JACKSON, D., 2014. Scalable vector graphics (svg) 1.1 (second edition).

GEHLENBORG, N., 2012. The refinery platform for analysis and visualization of biomedical genomics data. Invited Talk held at the Center for Biomedical Informatics & Countway Library Symposium, Boston, MA, USA, 2012.

GEHLENBORG, N., 2013. Refinery platform - integrating visualization and analysis of large-scale biological data. Slides of a talk given at the Bioinformatics Open Source Conference 2013, Berlin, Germany, 2013.

GEHLENBORG, N., 2013. Visualization and analysis of large (Epi)Genomics data sets with refinery. Slides of a talk given at the Understanding Cancer Genomics Through Information Visualization Symposium, Tokyo, Japan, 2013.

GEHLENBORG, N., 2014. Refinery platform documentation.

GIARDINE, B., RIEMER, C., HARDISON, R. C., BURHANS, R., ELNITSKI, L., SHAH, P., ZHANG, Y., BLANKENBERG, D., ALBERT, I., TAYLOR, J., MILLER, W., KENT, W. J., AND NEKRUTENKO, A. 2005. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research 15*, 10, 1451–1455.

GOECKS, J., NEKRUTENKO, A., AND TAYLOR, J. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology 11*, 8, 1–13.

MAGUIRE, E., ROCCA-SERRA, P., SANSONE, S.-A., DAVIES, J., AND CHEN, M. 2012. Taxonomy-based glyph design - with a case study on visualizing workflows of biological experiments. *IEEE Transactions on Visualization and Computer Graphics (InfoVis' 12) 18*, 12, 2603–2612.

MAGUIRE, E., ROCCA-SERRA, P., SANSONE, S.-A., DAVIES, J., AND CHEN, M. 2013. Visual compression of workflow visualizations with automated detection of macro motifs. *IEEE Transactions on Visualization and Computer Graphics (InfoVis' 13) 19*, 12, 2576–2585.

MISSIER, P., SOILAND-REYES, S., OWEN, S., TAN, W., NENADIC, A., DUNLOP, I., WILLIAMS, A., OINN, T., AND GOBLE, C. 2010. Taverna, reloaded. In *Proceedings of the Conference on Scientific and Statistical Database Management*, Springer-Verlag, 471481.

ROCCA-SERRA, P., BRANDIZI, M., MAGUIRE, E., SKLYAR, N., TAYLOR, C., BEGLEY, K., FIELD, D., HARRIS, S., HIDE, W., HOFMANN, O., NEUMANN, S., STERK, P., TONG, W., AND SANSONE, S.-A. 2010. ISA software suite: supporting standards-compliant experimental annotation and enabling curation at the community level. *Bioinformatics 26*, 18, 2354–2356. PMID: 20679334.

THE GALAXY TEAM, 2014. Galaxy project web page.

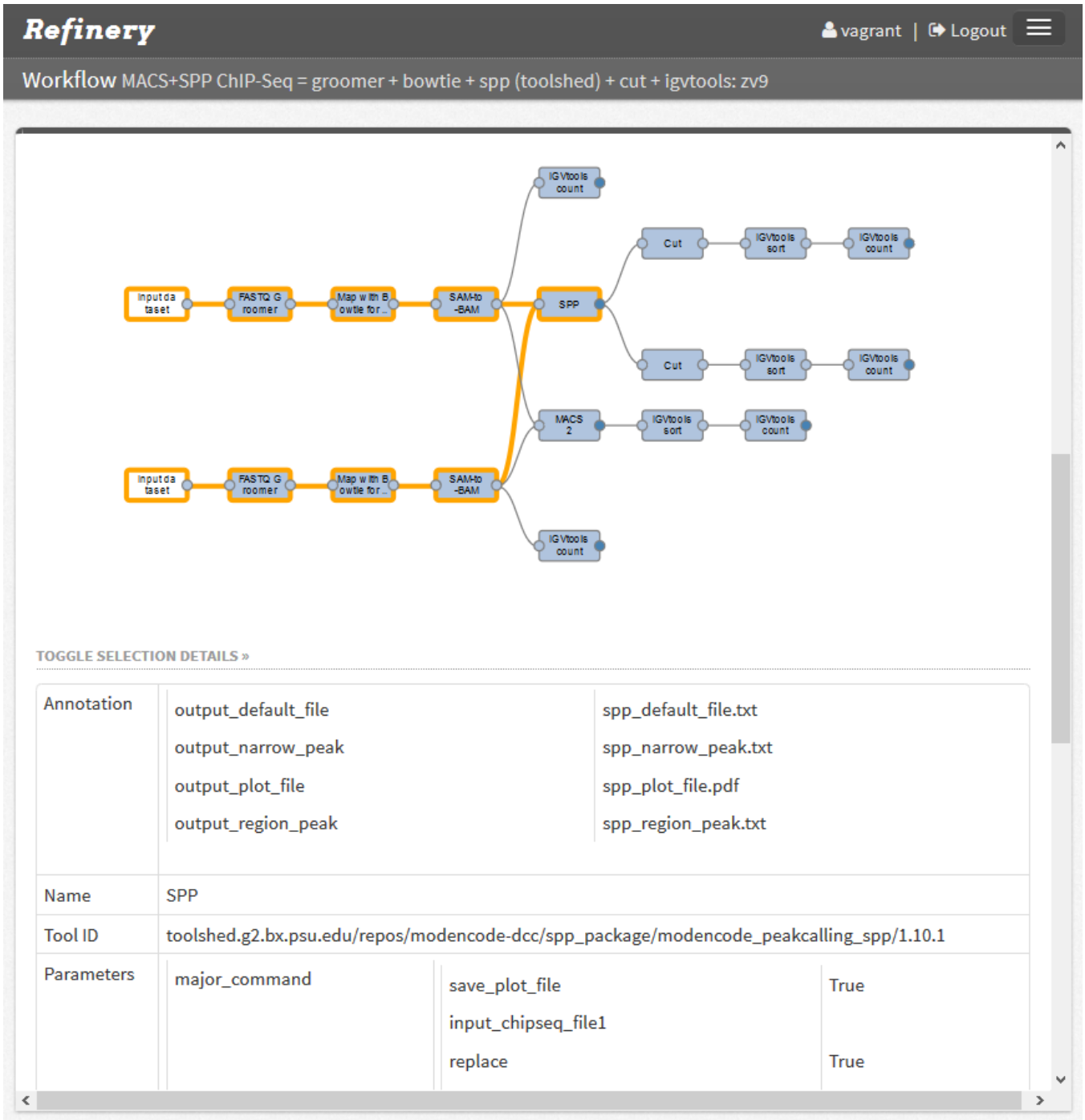

Figure 17: *All features together result in a state of the art workflow visualization providing interaction, minimalistic or detailed view - collapse and expand behavior as well as the HTML table are on-demand features and can be disabled by the user.*