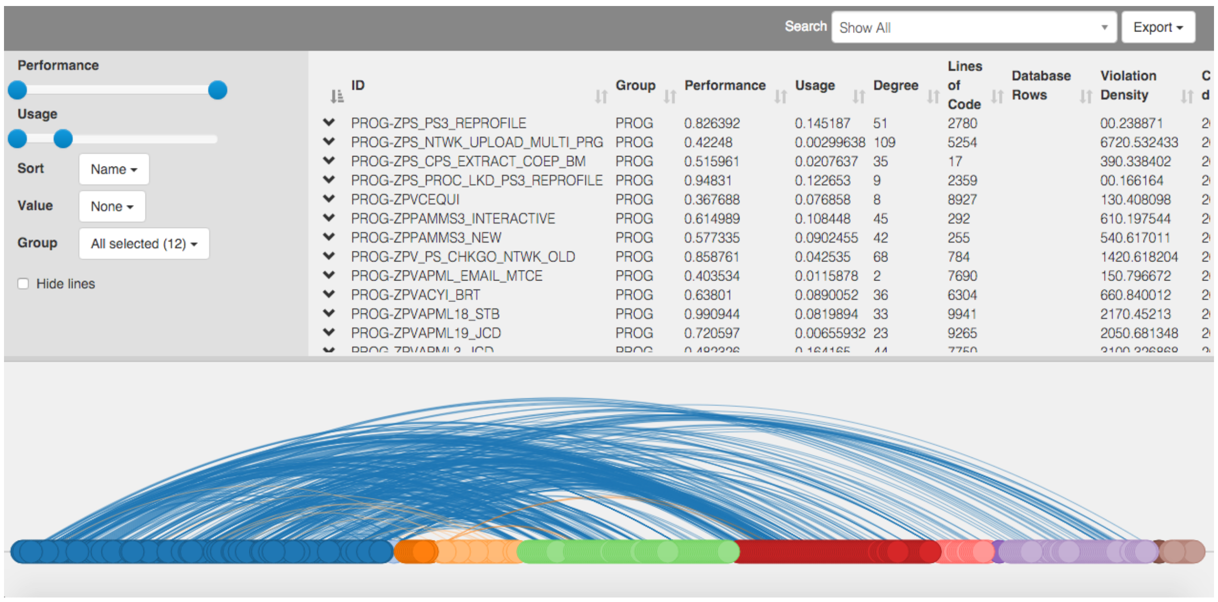
SAPVis: An interactive system explorer

Vaden Masrani



ID	Group	Performance	Usage	Degree	Lines of Code	Database Rows	Violation Density	C d
PROG-ZPS_PS3_REPROFILE	PROG	0.826392	0.145187	51	2780		00.238871	2
PROG-ZPS_NTWK_UPLOAD_MULTI_PRG	PROG	0.42248	0.00299638	109	5254		6720.532433	2
PROG-ZPS_CPS_EXTRACT_COEP_BM	PROG	0.515961	0.0207637	35	17		390.338402	2
PROG-ZPS_PROC_LKD_PS3_REPROFILE	PROG	0.94831	0.122653	9	2359		00.166164	2
PROG-ZPVCEQUI	PROG	0.367688	0.076858	8	8927		130.408098	2
PROG-ZPPAMMS3_INTERACTIVE	PROG	0.614989	0.108448	45	292		610.197544	2
PROG-ZPPAMMS3_NEW	PROG	0.577335	0.0902455	42	255		540.617011	2
PROG-ZPV_PS_CHKGO_NTWK_OLD	PROG	0.858761	0.042535	68	784		1420.618204	2
PROG-ZPVAPML_EMAIL_MTCE	PROG	0.403534	0.0115878	2	7690		150.796672	2
PROG-ZPVACYI_BRT	PROG	0.63801	0.0890052	36	6304		660.840012	2
PROG-ZPVAPML18_STB	PROG	0.990944	0.0819894	33	9941		2170.45213	2
PROG-ZPVAPML19_JCD	PROG	0.720597	0.00655932	23	9265		2050.681348	2

Fig. 1. SAPVis arc layout showing a filtered selection of a SAP domain

Abstract—In this work we present SAPVis, an interactive network visualization tool which helps SAP developers track dependency graphs in their network in order to guide testing and find inefficiencies within their system. By using an non-static arc diagram layout coupled with user manipulation tools (filter, sort, zoom, pan, and resize by attribute) we show how our visualization can help developers view and compare subgraph network topologies, find duplicate code, find outliers, and understand the impact of making changes to a component within the system.

Index Terms—SAP, Arc Diagram, Network

1 Introduction

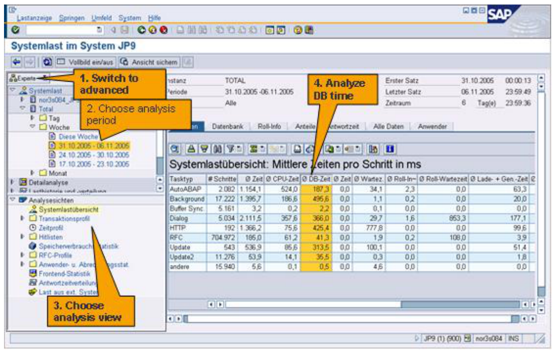
SAP (*Systems, Applications & Products in Data Processing*) is software used by large multinational corporations to manage their daily operations. SAP is used by over 290 000 companies in 190 countries and controls 24% of the global enterprise software market. Companies use SAP to manage inventory, sales, expenses, payroll, and everything else organizations need to keep track of in order to run efficiently. The base SAP system is robust enough to meet most business requirements but many companies nonetheless customize the application to deal with their specific requirements. These customizations could include millions of lines of code over thousands of objects built and enhanced over time. These custom components consist of a large interconnected network of objects which can become obsolete and slow. Because these system are often managed by changing staff they often contain many inefficiencies, including duplicate or unused code, expired users, redundant data tables and ineffective or outdated test suites. A tool that can help technicians quickly locate these inefficiencies would help companies improve their SAP system and save them money.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.
Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/

SAP currently offers numerous tools for developers and system technicians. Many times, however, the output from these tools are presented as a complicated table of values without any visual aids. For example, the *ST03 Workload Monitor* as seen in Figure 2 shows performance statistics of all programs run by the company [10]. This tool allows the user to view statistics pertaining to a particular artifact, where here we define artifact as any SAP-specific object, including users, program, data table, transaction code, function group, and more as seen in table 1.

One thing the Workload Monitor does not do, however, is display *dependency graphs*. *Dependency graphs* are a series of caller-callee (parent/child) relationships which show artifact A depending on artifact B. One example of a dependency graph is shown in Figure 3, where we see IPC_SAP.h depending on ace/IPC_SAP.i, ace/Flag_Manip.h, ace/post.h, and ace/pre.h, which in turn depend on other programs. Developers have many uses for dependency graphs. First, they are a way to track the effect of a change in a particular area of the system. Upon modifying a program, developers can run tests only over those artifacts in the dependency graph of the program being changed rather than testing the entire set of programs in the system. It's common for SAP systems to contain over 90000 artifacts while some of the dependency graphs within those system can contain fewer than a hundred nodes. Therefore testing only over the dependency graph saves time and resources. A developer also can get a quick visual estimate of code similarity between two artifacts by comparing multiple dependency graphs. Too often, because the SAP system is central to a company's operation, developers will make a copy of a productive program to make changes to it rather than modify the program itself. Unfortunately, this results in several programs that more or less do the same thing but cause maintenance issues when a bug is found in the original. Examples of these are programs like ZBUD, ZBUDNEW, ZBUDOLD and so one. An artifact can have multiple parents and multiple children, or they can have no parents or children at all, which we call "orphaned". Therefore dependency graphs also serve as a quick visual indicator of the relative importance of a particular artifact. A high degree indicates importance in the system while orphaned nodes may be able to be archived or deleted.

We envision SAPVis to be one tool among many used by SAP developers tasked with improving the efficiency of their system. Specifically, we want to quickly find underperforming, unused or duplicate artifacts that can be modified to make the SAP system more responsive and stable. In addition, we want to provide as quick way for technicians trace the effects of changes to an area of their network. The dataset and tasks are described in detail in section 3. We will accomplish this by displaying the network in an interactive arc diagram as described in sections 4 and 5. We begin below with a brief overview of common techniques used to display network data.



Instanz	TOTAL		Erster Satz	31.10.2005 00:00:13
Periode	31.10.2005 -06.11.2005		Letzter Satz	06.11.2005 23:59:49
	Alle		Zeitraum	6 Tag(e) 23:59:36

Systemlastübersicht: Mittlere Zeiten pro Schritt in ms

Tasktyp	# Schritte	Ø Zeit	Ø CPU-Zeit	Ø DB-Zeit	Ø Zeit	Ø Wartez.	Ø Roll-In-	Ø Roll-Wartezeit	Ø Lade- + Gen.-Zeit
AutoABAP	2.082	1.154,1	524,0	187,3	0,0	34,1	2,3	0,0	63,3
Background	17.222	1.396,7	186,6	495,6	0,0	1,1	0,2	0,0	20,0
Buffer Sync.	5.161	3,2	0,2	2,2	0,0	0,1	0,0	0,0	0,0
Dialog	5.034	2.111,5	357,6	366,0	0,0	29,7	1,6	853,3	177,1
HTTP	192	1.366,2	75,6	425,4	0,0	777,8	0,0	0,0	99,6
RFC	704.972	185,0	61,2	41,3	0,0	1,9	0,2	108,0	3,9
Update	543	536,9	85,6	313,5	0,0	100,1	0,0	0,0	51,4
Update2	11.276	63,9	14,1	35,5	0,0	0,3	0,0	0,0	1,8
andere	15.940	5,6	0,1	0,5	0,0	4,6	0,0	0,0	0,0

Fig. 2. ST03 Workload Monitor, the code monitoring tool currently offered by SAP

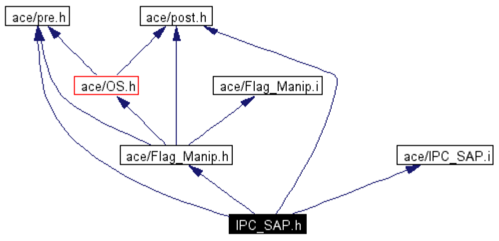


Fig. 3. A small dependency graph showing the relationship between seven programs. Here we see IPC_SAP.h depending on four other programs, ace/IPC_SAP.i, ace/Flag_Manip.h, ace/post.h, and ace/pre.h.

2 Related Work

There have been many proposed network visualization idioms. The most common and most intuitive is a node-link (NL) diagram, where each item is represented by a point in 2D space and lines are drawn to show connections between items. Data attributes can be encoded by the size, shape and colour of the nodes and edges. The NL idiom works well with small, sparse networks but suffers from readability and performance issues with large and highly connected networks. Computing the positions of the nodes can be a challenge, particularly with large networks which are prone to producing "hairballs". Force directed layout algorithms are the most common algorithms for computing node layouts, but they are notoriously brittle on large networks, and are non-deterministic which can cause confusion with the user. Alternative layouts include Chord Diagrams which lay the nodes on the circumference of the circle or arc diagrams which lay the nodes along one dimension [7]. Computing the position of nodes in 1D is inexpensive compared with the force directed layout algorithms and arc diagrams avoid the hairball problem, but they can still suffer from occlusion and edge crossing issues, especially as the size of the network grows.

A common alternative to the node-link diagram is an adjacency matrix (AM), where networks are displayed in a symmetric 2D matrix with the rows and columns each representing the nodes, and an entry at position M(i,j) representing an edge between node i and j. The entries can be filled with a boolean, an edge weight, or a colour to encode an edge attribute. Adjacency matrices can handle large and dense matrices but their major weakness is unfamiliarity; most users need to be taught how to read adjacency matrices and the steep learning curve can be a hurdle, especially for non technical user [8].

There has been work in trying to overcome the limitations of both views by combining adjacency matrices and node-link views. For example, because tracing paths can be more challenging in adjacency matrices than in NL diagrams, MatLink [3] overlays the matrix views with an arc diagram along the row and column headers to facilitate path tracing. MatrixExplorer [2] provides two synchronized views (matrix and NL) faceted together along with manipulation tools - filtering, clustering, reordering, sorting, zoom and pan, annotation - to allow the users to explore the data. NodeTrix [4] shows multiple adjacency matrices linked together by first showing the network as a NL diagram and allowing the users to select dense regions of the network to be shown as an adjacency matrix.

Another approach in visualizing large networks is to aggregate the data in such a way as to provide the user with a meaningful abstractions of the entire network. This gives the user metadata on the network as a whole rather than displaying the exact network topology. PivotGraph [13] does this by performing a roll-up operation to combine similar nodes and display them on an grid with two categorical attributes along the axes. Instead of displaying every node and arc in the network, aggregate nodes and arcs display the relationship between categorical attributes in the network as a whole. The Honeycomb [12] system also aggregates nodes, but instead uses a predefined hierarchy to aggregate cells of an adjacency matrix in order to display social networks with millions of connections. Elzen and Wikjs DOSA system [11] aggregates user selections and displays meta-information, such as number of nodes and edges with the selection, in a high-level overview display alongside the original network. The original network is displayed as a scatterplot with the x and y axis encoding two of its attributes. Filtering and selection tools are provides to the user to allow them to select which areas of the network they wish to aggregate.

Nagel and Duval offer a visual survey of arc diagram techniques and list five primary characteristics that distinguish arc diagrams from one another. They are: 1. Node distance 2. Node seriation 3. Arc directionality 4. Arc weight and 5. Degree of connections [9]. Characteristics 1 and 2 pertain to the layout of the nodes. In our case, distance between nodes holds no meaning, what Nagel and Duval call *Layout*, but the nodes are serialized based on both type and a user selected attribute. Variants to the *Layout* choice include *Data*, where distance between nodes maps to a value in the data, and *Metadata*, where the distance is based on a value of the diagram (eg. degree). Criteria 3 and 4 pertain to the characteristics of the arcs. In our design, arcs are neither directed nor weighted, but they are coloured based on the parent node, meaning the type of the parent can be discerned by looking at the arc. Adding arc weights and directionality is an area of future work. Finally, Nagel and Duval's fifth criteria specifies whether an arc connects one node or groups of node. Our design has opted for showing connections between single nodes, but integrating the pivot graph roll-up technique would be a possible extension of this work.

Surprisingly, all the arc diagrams listed by Nagel and Duval were static and could not be manipulated by the user. In contrast, our system allows the user to filter, zoom, select, resize, and resort the nodes in order to explore their network. It is important to preserve network topology in the context of understanding an SAP system so this work will not aggregate nodes like was done in the PivotGraph and Honeycomb systems. To handle the scale issues that come with a NL view, we will instead allow the user to reduce their dataset by attribute values in order to find outliers and extremum. We have opted to use a node-link view over an adjacency matrix because we want to minimize the learning curve of the tool and facilitate easy path exploration.

3 Data and Task Abstractions

The data comes from CodeExcellence, a company which offers tools to help companies monitor the quality of their code. The data and tasks are described below.

3.1 Data Description

It is common for a highly customized SAP system to contain over 90,000 artifacts which are divided into various domains that represent different subdivisions within a business. For example, a domain could represent all users and programs in the finance department within the company. Our system will display one domain, which are typically approx. 3000 nodes and 15000 edges. Table 1 lists the 13 primary artifacts which can exist in a domain. Each artifact has a performance and usage score which come internal monitoring within the SAP framework, as well as creation dates, database size (where applicable) and a list of dependencies.

The dataset consists of item and link data where each item has categorical, ordinal and quantitative attributes. The main categorical attributes are types (eg. USER, PROG, TCOD), domain (eg. "PS - Project System") and id (eg."PROG-ZZ_CONVERT_SDCC_DATA"). The ordinal attribute is the creation date. The main quantitative attributes are performance, usage, size and degree. The performance and usage attributes are scores between 0 and 1, the size by number of lines of code, and the degree is an integer which measures the number of incoming and outgoing edges.

Link data shows the relationships between various artifacts, where a link between $artifact\_i$ and $artifact\_j$ means $artifact\_i$ calls or has access to $artifact\_i$ in the SAP system. This can be a user who has permissions to modify a particular program, or a program which uses a particular data table. The links are coloured based on the parents type and a node can have multiple or zero links. Circular links are possible (where $artifact\_i$ and $artifact\_j$ which calls $artifact\_i$) but are currently not displayed, as one arc will be occluded behind the other.

3.2 Task Description

It is important to identify those artifacts which have many links as these are an integral part of the SAP domain and therefore should be tested often. Similarly, it is important for SAP technicians to identify orphans, or artifacts with no parents or children, as these are not communicating with any other part of the system. Orphans are likely duplicate or unused code which can be deleted. Besides looking for artifacts with zero links, a technician can compare dependency graph topologies in order to estimate whether two artifacts are duplicates, or near duplicates of each other. Of course the technician would have to verify this by looking at the code directly, but the topological view allows the technician to quickly scan for potential duplicates. A technician might also want to verify their changes have no unintended consequences by running tests over all artifacts in the system that are dependent on the modified code. The dependency graph allows the user to quickly identify the IDs of those artifacts they want to test. Finally, the technician may want to find bottlenecks in their system by selecting artifacts which meet a particular criteria, for example low performance and high usage. These artifacts would then be flagged as needing to be rewritten.

In the language of visualization, we are creating a tool that allows users analyze and search within their network. The analyze task lets user consume their data in order to discover outliers (programs that aren't called, users with no permissions, tables that aren't used), find similarities in network topologies which may be a source of duplicate and therefore redundant code and find extremum (eg. programs with high usage and low performance). The tool should also support the four types of search, "Lookup", "Locate", "Browse", "Explore". The user may want to use the search bar or table to look up a particular node in order to view the attributes associated with that node. By allowing the user to sort the nodes based on attributes we introduce "locate", which in this context is locating the position of a particular node along the axis after the nodes have been sorted by an attribute. If the user does not know the identity of the node(s) they are looking for, they may choose to "browse" instead by exploring the parents or children of a node, or sorting by an attribute and looking at the low or high range of each type. Finally, by making use of the zoom, pan, sort and filter tools, they can explore the subsets of the network without having a specific target in mind.

4 Solution

After surveying various network layout idioms, we settled on an 1D arc layout over the more conventional 2D force directed layout or adjacency matrix approach. Initially we tried a 2D layout, trying the a number of parameters with the Barnes-Hut [1], Kamada and Kawai [6], and ForceAtlas2 [5] algorithms on datasets with 1500 nodes and 10000 edges. None avoided the "hairball" problem and all took upwards of five minutes to stabilize. As well, spacial position in force-directed layouts contain no inherent meaning (except relationally with clustering) and so we would not have been able to encode any quantitative information via the position channel nor would we have been able to offer the user sort functionality. Adjacency matrices would solve the performance issue and are able to be sorted based on quantitative attributes but most users are unfamiliar with this idiom and therefore encounter a steep learning curve when learning to read the matrix. We recognizing our tool is one among many a SAP developer may use and a steep learning curve would reduce the adoption rate of our tool.

For these reasons we decided on an arc layout approach. This should be an intuitive idiom for most new users and we have the position, luminance, saturation and area channels with which to encode information. The arc diagram is faceted with a standard table view, and a filter pane on the left side, as seen in image 1. The arc diagram is fully interactive; the user can zoom pan, and the immediate parents and children (one level of the dependency graph) is displayed upon hovering over a node. A user can select a node which displays its full dependency graph by lightening the saturation of all nodes and arcs in the network not associated with the selected node. The table updates with the attributes associated with that dependency graph. This can be seen in figure 4 which shows the result of a user selecting the node TABL-COBK. The associated information for the dependency graph is shown in the table (not shown) and the user choose to export this chain to a csv/xml/json for further testing.

The nodes are grouped by type along the axis and the type in encoded through the hue channel. Within each group, nodes can be sorted and resized by the user, who selects a quantitative attribute from the dropdown selectors on the filter pane. The nodes are initially sorted alphabetically and are unsized. The colour of each arc encodes the type of the parent node (eg. blue arcs are from blue PROG artifacts, orange arcs are from orange TABL artifacts). Nodes are permitted to overlap initially and then spread out along the axis as the user zooms in. By making use of semantic zooming, which keeps the size of each node fixed while reposition the node and arcs along the axis, we are able to display thousands of nodes.

Complementing the table and arc diagram is a filter pane which allows the user to reduce the dataset by selecting performance and usage ranges with the sliders and chose which groups to display. They can also decide to resort the nodes alphabetically or by degree, performance, usage, creation date or by lines of code. Entire groups can be repositioned by clicking and dragging the group name within the dropdown selector. As well, the user can resize each node by performance, usage, or lines of code (LOC). These tools allow the user to browse through subsets of the data without having a specific target in order to find artifacts that may be compromising the performance of their system. Should the user wish to locate a specific artifact, a search bar is included in the upper right corner. Envisioned use cases for these selections are discussed in Section 6 and the full what-why-how analysis is shown table 2.

What: Data	Node/Link data; quantitative, ordinal and categorical attributes .
Why: Tasks	Find extremum, find outliers, compare network topology, find similar items.
How: Encode	Arc diagram layout, hue encodes type, opacity encodes focus, size encodes user-selected quantitative attribute.
How: Facet	Arc diagram faceted with a table and control panel.
How: Reduce	Filtering.
How: Manipulate	Zoom, pan, select, sort.
Scale: Manipulate	approx. 3000 nodes, approx. 15000 links.

Table 2. What-Why-How framework

5 Implementation

This visualization was implemented using the libraries D3, jQuery, Bootstrap, DataTables.js, and underscore.js. The third party packages bootstrap-multiselect.js, bootstrap-slider.js, chosen.js, react-rubaxa-sortable.js and tinycolor.js were used for the select, search and slider widgets. A designer was consulted at the beginning stages to help with layout and colours, while all coding was done by the paper's author.

The majority of the work was spent implementing the arc diagram using D3, which proved to be quite challenging. There were no examples to start from as all the arc diagrams were static and only displayed a few hundred nodes at most. The biggest hurdle was handling scale; D3 is designed to manipulate SVG elements, but SVG requires the browser maintain an object model for each element. This leaves a heavy memory footprint which becomes noticeable after a few thousand elements are drawn. In our case, we were drawing over 18000 elements (arcs and nodes) which made the performance so poor the system was unusable. The solution was to switch to HTML5's canvas element, which draws the pixels on the page without having an in-memory representation of the object just drawn. This means the canvas element performs a lot better when many items are drawn, but animation is more difficult because one cannot use D3's inbuilt tweens to animate as they rely on an in-memory DOM object transition been states.

The other main challenge with implementing the arc diagram was having to calculate the pixel position of each node after the user changed any of the filter parameters. This was difficult because one had to consider both the zoom level, the group position, the sort order, and the three possible changes of state (a node being added, changing position, or being removed.) It turned out that following a rough MVC pattern was the best way to have the three components interact with one another. Here the the data was handled by a "model" (DataHandler.js), the index.html acted as the controller by reading the user input and sending the filtered data to two "views" (arc_diagram.js and DataTables.js).

6 Results

Below we present number of use cases demonstrating how SAPVis answers the questions posed in the introduction. Overall we feel SAPVis accomplishes most of its targets although more work is required to improve the overall user experience and reduce cognitive load. This includes adding basic functionality that weren't permitted by time constraints, like a color legend, the ability to select multiple nodes at a time, and the have the names of all the nodes in a dependency graph displayed upon selection. The discussion that follows the use cases below will address ways to improve the user experience as well as suggest additional functionality and modifications for future iterations.

6.1 Use Case 1: Smarter testing

A SAP technician has just renamed a column in the COBK table and wants to make the requisite changes in the programs which use the table. For testing, he wants a list of the transaction codes which call the programs that use COBK. He begins by entering "COBK" in the search bar. The dependency graph is displayed but cluttered amongst all the other nodes. In order to improve readability he hides the background arcs and all the groups except "PROG", "TCOD", and "TABL"and moves "TCOD" the top of the list by dragging from the dropdown

dropdown menu. He chooses to sort by degree in order to reduce edge crossings and is left with the view seen in figure 4. He uses the zoom and pan to get a rough idea of how many transaction codes and programs rely on this table, and then exports the chain as a CSV. He uses CSV in a script which runs test on the given IDs to assure his changes are error free.

6.2 Use Case 2: Browse poor or unused code

Best Buy asks a technician to improve the performance of their SAP system, as it is taking too long to scan items and customers are getting upset with long lines at checkout. The technician doesn't know which programs are causing the bottleneck so she decides to make a list of programs that might need to be rewritten. Along with the ID of each program, she wants to save its dependency graph so she knows what should be tested if that program is modified. She begins by sorting all the nodes by degree and resizing the nodes by performance, knowing that leftmost nodes with small radii are poor performers that are used by many entities throughout the system. She selects a few of the smallest nodes as seen in Figure 5, confirms their low performance by looking at the attributes in the table, and exports their dependency graphs. Next, she uses the sort dropdown menu to sort by usage and adjusts the performance filter to select only the bottom 10%. To narrow the search further, she uses the second slider to select only those items with a usage score above 90%. She is left with four low performing but highly used programs. She selects each and exports.

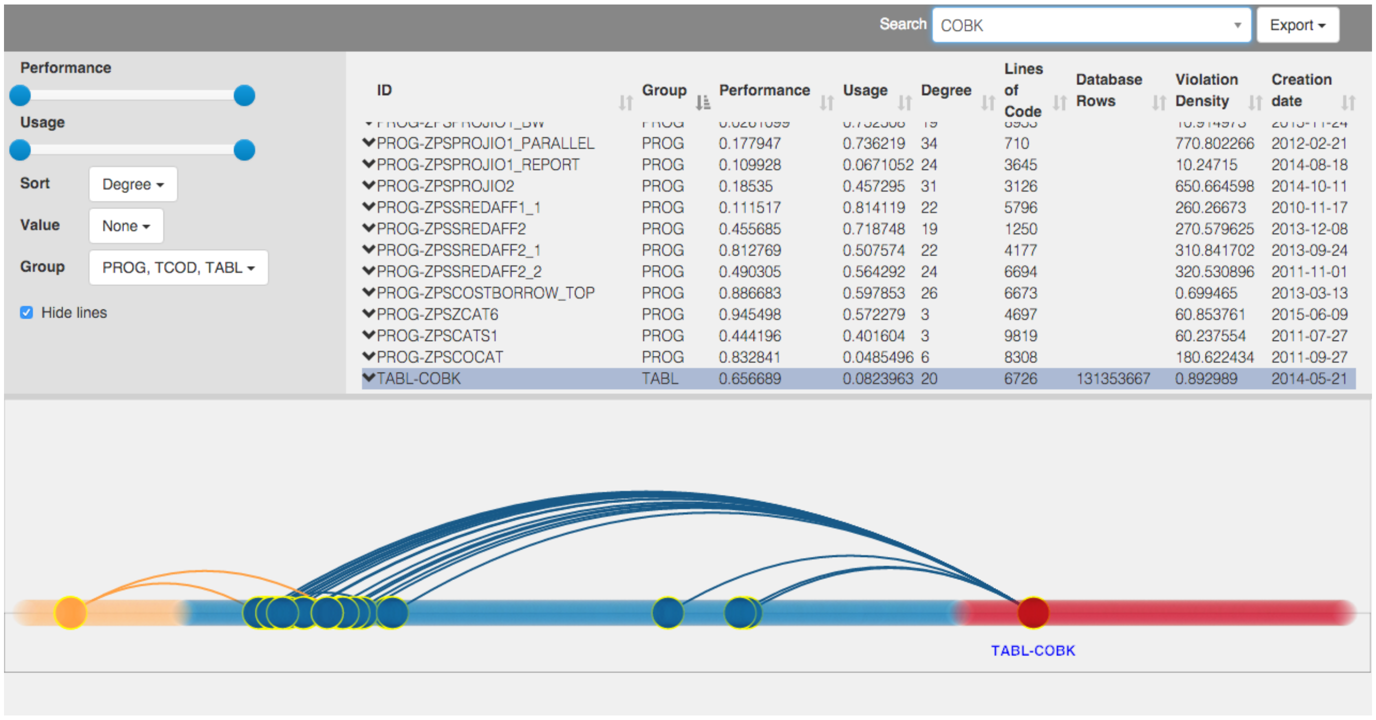
She also decides to make a separate list of code that may longer be used. Proceeding in much the same way as before, she sorts by degree but this time looks at the rightmost side of each group, noting all those orphaned programs with no incoming or outgoing edges. These are artifacts that are not communicating with the rest of the system and can likely be deleted. She again uses the sliders except this time lowers the usage slider to only view those nodes with a usage score beneath 10% and documents those nodes which remain.

6.3 Use Case 3: Find duplicate code

While scanning through the table, a technician notices multiple programs with similar names, as seen in Figure 6. The technician also notices the similar line counts and low usage scores for items 0110 - 0140 and suspects a previous developer duplicated a program rather than updated it, thereby leaving a redundant copy of an old program. To confirm his intuition, he selects each item from the table in order to compare their network topologies. He notices they are identical with the exception of one struct difference between them. The technician later confirms this by looking at the code directly and sees the old versions are no longer used. By comparing the highlighted networks, the technician was able to quickly estimate the degree of code similarity between two programs and find redundancy in their system.

7 Discussion and Future Work

We consider SAPVis successful although incomplete at the time of writing. Before putting the tool in front of users, there are a few additions that need to be made in order to consider the prototype com-



Sort: Degree
Value: None
Group: PROG, TCOD, TABL
☑ Hide lines

ID	Group	Performance	Usage	Degree	Lines of Code	Database Rows	Violation Density	Creation date
PROG-ZPSPROJIO1_PARALLEL	PROG	0.177947	0.736219	34	710		770.802266	2012-02-21
PROG-ZPSPROJIO1_REPORT	PROG	0.109928	0.0671052	24	3645		10.24715	2014-08-18
PROG-ZPSPROJIO2	PROG	0.18535	0.457295	31	3126		650.664598	2014-10-11
PROG-ZPSSREDAFF1_1	PROG	0.111517	0.814119	22	5796		260.26673	2010-11-17
PROG-ZPSSREDAFF2	PROG	0.455685	0.718748	19	1250		270.579625	2013-12-08
PROG-ZPSSREDAFF2_1	PROG	0.812769	0.507574	22	4177		310.841702	2013-09-24
PROG-ZPSSREDAFF2_2	PROG	0.490305	0.564292	24	6694		320.530896	2011-11-01
PROG-ZPSCOSTBORROW_TOP	PROG	0.886683	0.597853	26	6673		0.699465	2013-03-13
PROG-ZPSZCAT6	PROG	0.945498	0.572279	3	4697		60.853761	2015-06-09
PROG-ZPSCATS1	PROG	0.444196	0.401604	3	9819		60.237554	2011-07-27
PROG-ZPSCOCAT	PROG	0.832841	0.0485496	6	8308		180.622434	2011-09-27
TABL-COBK	TABL	0.656689	0.0823963	20	6726	131353667	0.892989	2014-05-21

Fig. 4. Here we see a dependency graph for TABL-COBK which shows the table (red) being called by more than a dozen programs (blue) which are in themselves called from two transaction codes (orange). Here the use has filtered the selection by group, hiding all groups that are not PROG, TCOD or TABL. The user has not decide to value the nodes by any attribute and therefore they are all set to a default radius. The attributes for each node are displayed in the table with the selected node highlighted.

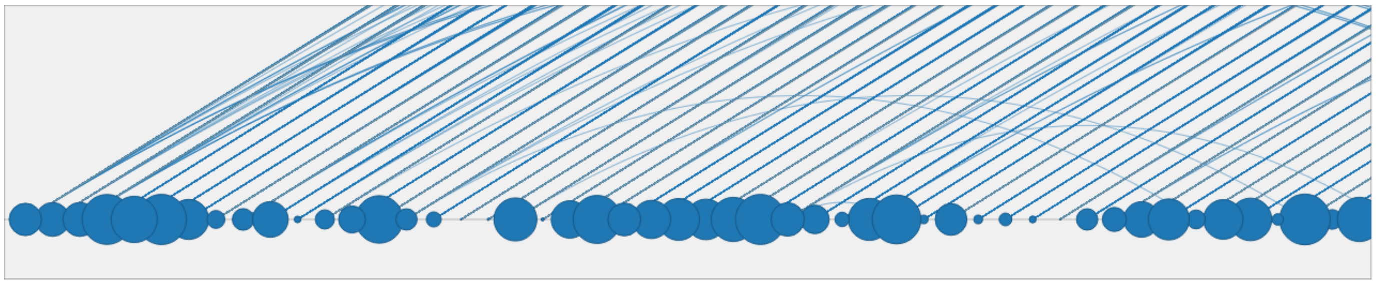


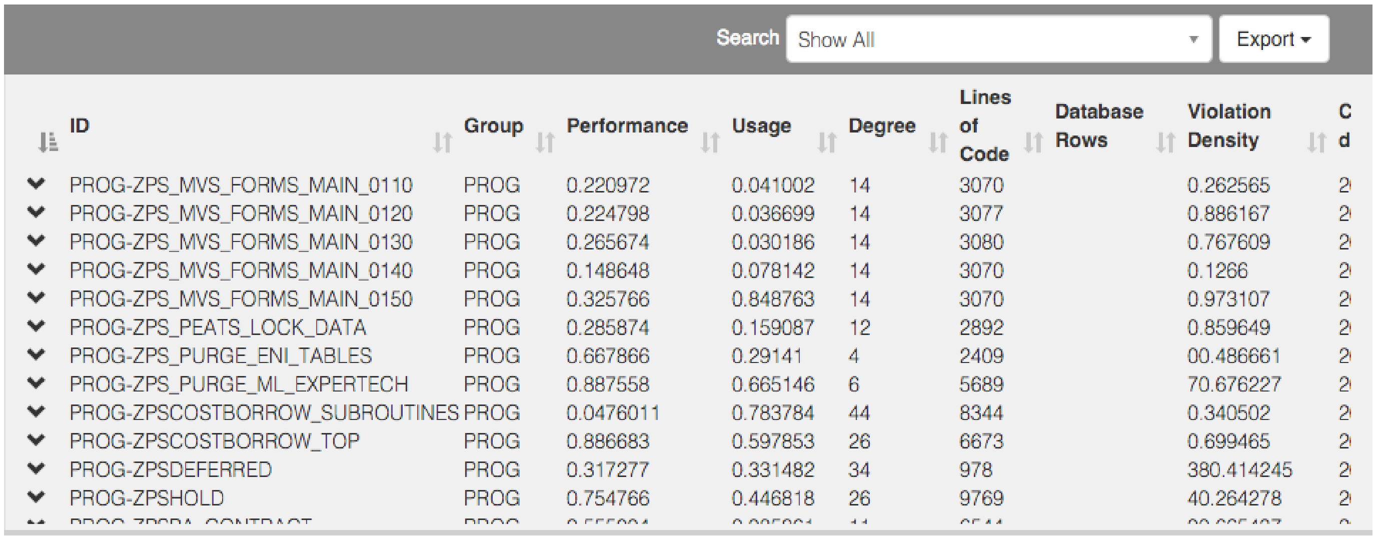
Fig. 5. The arc diagram sorted by degree and sized by value. Here we see the user has sorted the arc diagram in descending order of degree. The have also sized by performance in order to find the low performing programs which are called by many other programs in the system. In this way the user can find potential bottlenecks in their system.

plete. First, a legend so the types can be quickly distinguished, rather than having the user mouse over each color to learn the type. Also, the nodes should have their titles appear underneath once the user has zoomed in sufficiently to prevent the cognitive load associated with remembering which node has which ID. The other feature that needs to be implemented is the ability to select and export multiple nodes at once. Currently the user can either export the results of a filter or a dependency graph but cannot export a set of hand selected nodes. As well, dependency graphs are *directed* but our current system is showing undirected edges. The colour of the arc indicates the type of the parent node, but there is no way to determine the parent between arcs of the same type short of looking it up in the table.

All of these additions are necessary before considering the prototype complete because currently the cognitive strain on the user is too great; besides mousing over a node or referring to the table, there is no visual reminder of the IDs of nodes within a dependency graph. Besides these deficiencies, overall we feel SAPVis is successful. Unlike 2D force directed layouts,it is performant, allows on-the-fly filtering and grouping, and is able to handle the scale of one SAP domain, approximately 3000 nodes and 15000 arcs. As well it is intuitive and users are not faced with a steep learning curve as in the case of adjacency matrices.

SAPVas was designed to work alongside other debugging tools so future extensions may include adding additional views to the dashboard. One view we proposed initially (but decided to delay until receiving user feedback) was a collapsible tree view which would display the selected dependency graph. This would allow the user to explore large subgraphs and hide sections they are not interested in. As well, because only one subgraph would be displayed at a time, we would be able to avoid the hairball issue that results from displaying multiple interconnected subgraphs simultaneously. We would also like to improve the search bar to search through attributes as well as IDs.

Once we make the improvements to finish the prototype, we plan to get user feedback from domain experts in order to see how SAPVis works in conjunction with their current work flow. One major weakness of this study was lack of access to a SAP technician. Although the data provider was a domain expert in the sense of understand the SAP system, we did not have the opportunity to monitor the actual work patterns of SAP technicians. Understanding which tools they used and in which situations would have been very helpful in tailoring this tool



	ID	Group	Performance	Usage	Degree	Lines of Code	Database Rows	Violation Density	C d
	PROG-ZPS_MVS_FORMS_MAIN_0110	PROG	0.220972	0.041002	14	3070		0.262565	2
	PROG-ZPS_MVS_FORMS_MAIN_0120	PROG	0.224798	0.036699	14	3077		0.886167	2
	PROG-ZPS_MVS_FORMS_MAIN_0130	PROG	0.265674	0.030186	14	3080		0.767609	2
	PROG-ZPS_MVS_FORMS_MAIN_0140	PROG	0.148648	0.078142	14	3070		0.1266	2
	PROG-ZPS_MVS_FORMS_MAIN_0150	PROG	0.325766	0.848763	14	3070		0.973107	2
	PROG-ZPS_PEATS_LOCK_DATA	PROG	0.285874	0.159087	12	2892		0.859649	2
	PROG-ZPS_PURGE_ENI_TABLES	PROG	0.667866	0.29141	4	2409		00.486661	2
	PROG-ZPS_PURGE_ML_EXPERTECH	PROG	0.887558	0.665146	6	5689		70.676227	2
	PROG-ZPSCOSTBORROW_SUBROUTINES	PROG	0.0476011	0.783784	44	8344		0.340502	2
	PROG-ZPSCOSTBORROW_TOP	PROG	0.886683	0.597853	26	6673		0.699465	2
	PROG-ZPSDEFERRED	PROG	0.317277	0.331482	34	978		380.414245	2
	PROG-ZPSHOLD	PROG	0.754766	0.446818	26	9769		40.264278	2

Fig. 6. A snapshot of the table showing potentially duplicate code.

to suit their needs. User feedback will help fill this gap.

Besides technical details such as the use of canvas instead of SVG elements or how to use D3's enter-update-exit design pattern, the main takeaway lesson was to focus on rapid iteration at this early stage over code robustness. It is more important to get a working prototype into the hands of users than it is to make sure the code is sound, as some features may turn out to be solving the wrong problems. This means that, although the performance is less than optimal (we admit), the focus as this stage is completing the last of the prototype functionality in order to collect user feedback.

8 Conclusion

In this work we presented SAPVis, a tool to help SAP technicians visualize their systems by displaying relationships between SAP artifacts on an interactive arc diagram. We particularly focus on displaying *dependency graphs* which show the chain of dependencies emanating from a user selected artifact. This is useful as it allows developers to search for unintended consequences of changes to an artifact and provides a quick visual estimate of code similarity between artifacts. As well, we provide the user with manipulation tools that allow them to filter, sort and resize the nodes along the axis based on attributes to find extrema within their system. Although there has been work showing the various ways static arc diagrams can be modified, and how they can be used to augment other visualizations, to the best of our knowledge) this is the first research to show how a dynamic and interactive arc diagrams might be used.

Acknowledgments

The author wishes to thank Tamara Munzner for her guidance and CodeExcellence for their data.

References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. pages 446–449, Dec 1986.
[2] N. Henry and J. Fekete. Matrix explorer: a dual-representation system to explore social networks. *IEEE Trans. on Visualization and Computer Graphics*, (5):677–684, Sept 2006.
[3] N. Henry and J.-D. Fekete. Matlink: Enhanced matrix visualization for analyzing social networks. In *Human-Computer Interaction INTERACT 2007*, pages 288–302. Springer Berlin Heidelberg, 2007.
[4] N. Henry, J.-D. Fekete, and M. J. Mcguffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. on Visualization and Computer Graphics*, 13:1302–1309, 2007.
[5] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLoS ONE*, page e98679, June 2014.
[6] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, pages 7 – 15, 1989.
[7] M. J. McGuffin. Simple algorithms for network visualization: A tutorial. *Tsinghua Science and Technology*, pages 383–398, Aug 2012.
[8] T. Munzner. *Visualization Analysis and Design*. A K Peters, 2014.
[9] T. Nagel and E. Duval. A Visual Survey of Arc Diagrams. Oct 2013.
[10] SAP. MS Windows NT kernel description, 2015.
[11] S. van den Elzen and J. J. van Wijk. Multivariate network exploration and presentation: From detail to overview via selections and aggregations. *IEEE Trans. on Visualization and Computer Graphics*, pages 2310–2319, 2014.
[12] F. van Ham, H.-J. Schulz, and J. Dimicco. Honeycomb: Visual analysis of large scale social networks. pages 429–442. 2009.
[13] M. Wattenberg. Visual exploration of multivariate graphs. In *ACM CHI 2006 Conference on Human Factors in Computing Systems*, pages 811–819. ACM Press, 2006.

Name	Code	Description
User	USER	Stores read/write/edit permission‘’. One user can have access to multiple programs and TCODES
Program	PROG	A SAP program. Eg. Calculate vacation pay
Transaction Code	TCOD	The executable to run program(s). One transaction code can activate multiple programs. Users are expected to interact with the system through TCODES.
Database Table	TABL	A table in the database. Eg. table of all customer names and their contact information.
View	VIEW	A compiled view of the database that could include several tables. For example, a database view could return all outstanding payments to international vendors.
Function	FUNC	One function can be used in multiple programs. A function can be called by other functions. Eg. CALCULATE_GST.
Metadata	TTAB	Meta data about database tables. Table definition buffers.
Structure	STRU	A data structure used by one or more programs.
Sap Table	TTYP	A table stored in working memory within a SAP system instead of in the database.
Includes	INCL	Code fragments that can be used in other programs. Includes promote modularity and reusability in complex systems.
Method	METH	Methods of classes
Function Group	FUGR	Function Groups organize functions into logical units.

Table 1. List of artifacts within a SAP system