Supplemental Material for "Preserving Command Line Workflow for a Package Management System using ASCII DAG Visualization"

Katherine E. Isaacs and Todd Gamblin

Fig. 1: Dependency Graph Characteristics for Tool Blocks

Tool	# Nodes	# Edges	Layers	Nodes per Layer	Question	Question Layers
GraphViz	11	22	4	1-2-5-3	paths	1 & 3
`git`-like	11	22	4	1-2-5-3	paths	1 & 4
`graphterm`	11	22	5	1-1-2-4-3	paths	1 & 4
GraphViz	17	27	6	1-1-4-2-6-3	dependencies	5
`git`-like	17	26	7	1-2-4-2-1-4-3	dependencies	4
`graphterm`	18	26	5	1-4-6-5-2	dependencies	4
GraphViz	22	45	7	1-1-1-4-6-6-3	paths	1 & 5
`git`-like	22	45	7	1-1-1-4-6-6-3	paths	1 & 5
`graphterm`	22	45	7	1-1-1-4-6-6-3	paths	1 & 5
GraphViz	34	62	7	1-2-4-5-5-11-5-1	dependencies	8
					paths	1 & 7
`git`-like	33	60	7	1-1-4-5-5-11-5-1	dependencies	6
					paths	1 & 7
`graphterm`	34	61	7	1-2-4-5-5-11-5-1	dependencies	6
					paths	1 & 7

1 Experimental Objects used in Study

We provide more details about the experimental objects used in the study. Fig. 1 appends information about the number of layers per graph, the division of nodes in each layer, and the layers on which the target nodes for each question resided for the Tool block graphs. In some cases we were able to choose graphs that had similar major dependencies, resulting in highly similar graph structures.

2 Dependency Visualization Features in Github Repositories

We performed a online search for existing methods for visualizing dependencies using the search string "site:github.com visualize dependencies." We used Google search in an incognito tab of a newly installed Google Chrome with no signed in accounts. We manually retrieved links from the 49 returned pages. Some links were duplicated between results pages. We added additional links found from examining the total ones (procedure described below), resulting in a total of 521 links examined.

We explored each link to determine if it had a dependency visualization feature. If the returned link was not a project page (e.g., it was an issue, feature request, or project file) and did not mention visualization, we navigated to the main project page of the returned link. If the returned link referred to a project we already analyzed, we skipped it. We also skipped projects that were general visualization tools or were not targeted at a computing domain. For example, we skipped all projects where the target of the visualization was biology-related. We included machine learning network visualizations but not sentence structure dependencies in natural language processing or scene graph overlays in computer vision. We skipped links that were blog posts or personal websites. We manually inspected stand-alone code snippets ('gists'). If the returned link was a discussion thread (e.g., for an issue or feature request) and mentioned a possible outside-project visualization, we followed those links. For projects that were lists of links to other projects, we searched for promising links using the strings 'visual' and 'dependenc.' Of the 521 links, we found 483 unique projects.

From the main page of any project, we assessed the graph visualization features in the following manner. If we could discern enough information from the link returned by the search (sometimes the manual or wiki) or the README, we associated the found features with the project. If the graph visualization procedure was not fully explained (e.g., an image was shown but the libraries to generate it were not described or text contained something like 'visualizes with GraphViz'), we performed a directed search of the code to determine what the partial explanation meant. If no visualization procedure was mentioned in the README, we either read the entirety of the source code (for small code bases) or directed our search using Github to search the repository for the terms: (graph, network, tree, visual, view, plot, diagram, layout, svg, png, pdf, html, dot, gexf, graphml, dagre, d3, indent, ascii).

For the keyword search, we used all terms even after finding a visualization so that no keyword or visualization would get preference simply due to order of the search. We manually inspected the snippets returned in the first five pages of results for each term if they existed and searched further on promising leads. We limited the inspection to five pages *a priori* with the rationale that most users would not look further. Some projects had large numbers of results due to html documentation, repeated use of png or svg assets, or alternate meanings of the search terms (e.g., 'Visual Studio' in every code file turning up for 'visual'). Uses of GraphViz that were documentation-only (e.g., requirements of the documentation tool or library such as Doxygen, not to visualize dependencies) were not counted.

Some of the links found through examining the project links led to external sites. For those we looked at the features, gallery, and

- *Katherine E. Isaacs is with the University of Arizona E-mail: kisaacs@cs.arizona.edu.*
- *Todd Gamblin is with Lawrence Livermore National Laboratory E-mail: tgamblin@llnl.gov*

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

documentation pages for evidence of dependency visualization. We did not however read the entire documentation. Often in those cases we were unable to discern exactly how the visualization was accomplished, so data about what libraries or tools were used was not recorded. In our summaries, we consider the libraries used to be 'unknown' for these projects.

We found dependency visualization features in 224 of the 483 projects. Some projects had multiple visualization options, such as outputting a `dot` file to be rendered, rendering a png, and having a web application. Of the 224 projects, 108 had a visualization related to GraphViz through either outputting a `dot` format file (71 projects) or the use of a GraphViz-based rendering or layout algorithm to generate an image file, PDF, HTML file, or application (52 projects). Eighty-three of the projects enabled an HTML viewer for their visualization. In addition to the GraphViz-based ones, 47 used d3js as a central tool (e.g., force-directed layout, tree, Sankey diagram), 11 used dagre, five used ngraph, and three used visjs or networkx. A complete list of tools can be seen in Fig. 3. The most common form of visualization was a layered node-link diagram (e.g., hierarchical layout, `dot`, Sugiyama) with 129 projects. Fifty projects offered a force-directed layout and 34 showed a tree, 22 of which were drawn with ASCII.

Fig. 2 shows the number of instances of each view we categorized. Fig. 3 shows the number of instances of each tool or library being used to create the visualization. Fig. 4 shows the number of instances of each format returned by the visualization features that we found. Web applications are uniformly categorized as 'html.'

Included in these supplemental materials is a CSV listing all of the links and their categorization, ordered by Google search ranking.

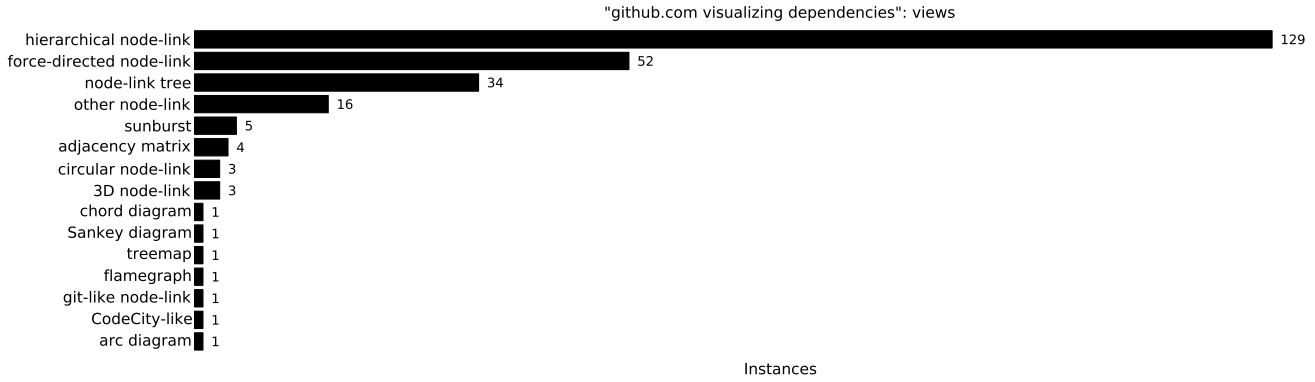

Fig. 2: Types of visualizations used by Github repositories.

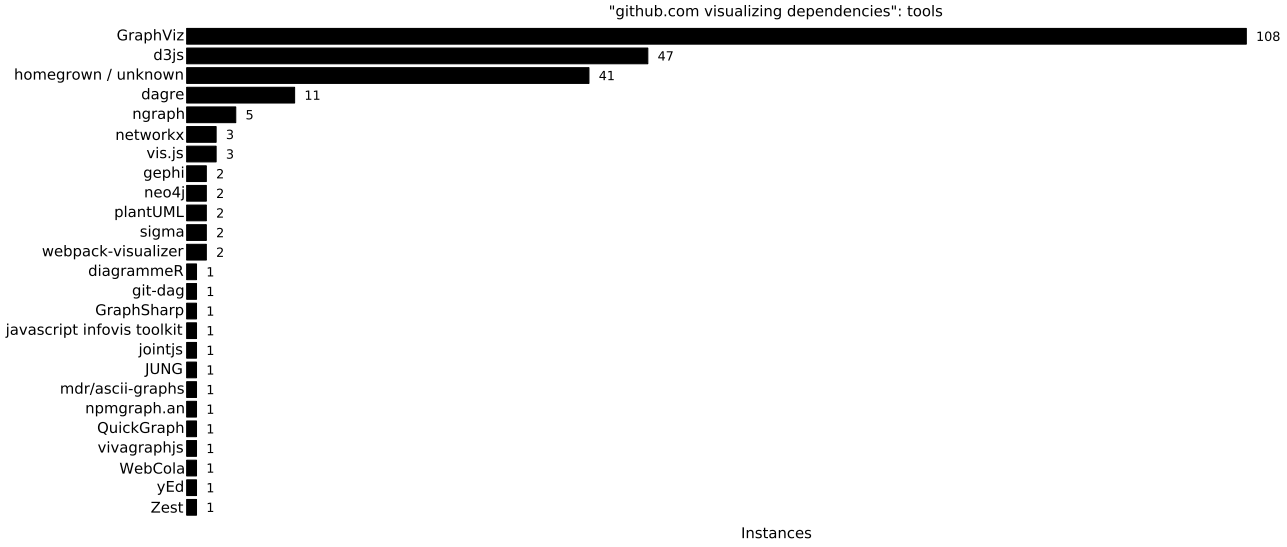

Fig. 3: Tools and libraries used for visualization features in Github repositories.

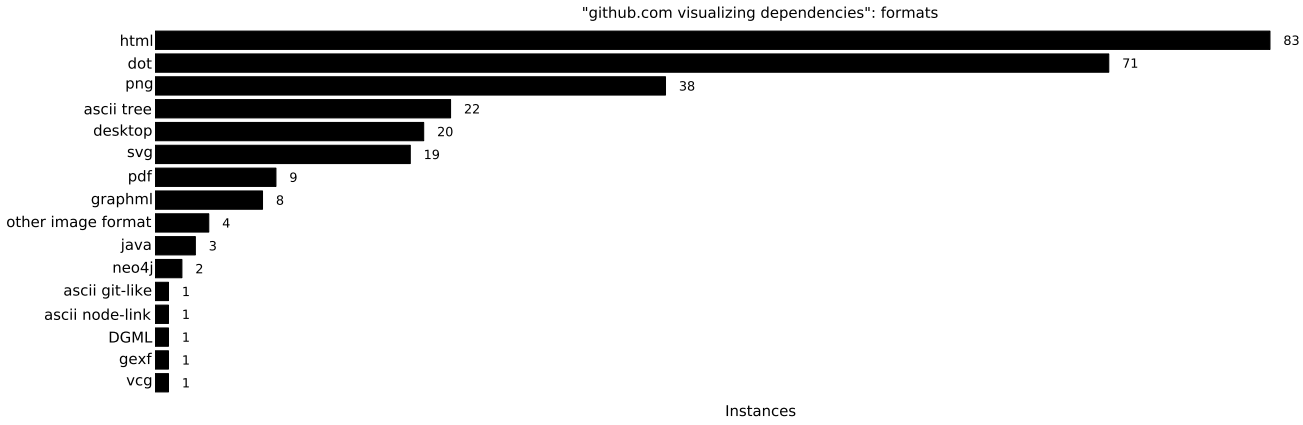

Fig. 4: File formats of the visualization features in Github repositories.