Scientific Workflow Visualization

Kristian Lazo, Yeji Han
University of Hawai`i at Mānoa – Information and Computer Sciences Department

WRENCH

What we proposed

Motivation

- Scientific Workflows are key to advances in science and engineering
- They consist of task graphs with control and data dependencies
- They have complex structures and can comprise thousands of tasks
- Research and practitioners need to be able to visualize workflow structures so as to understand how workflow executions can or should behave when executed on distributed computing platforms

Objectives

- Implement a tool for visualizing workflow structures as graphs of vertices and edges in a level-by-level fashion in the browser
- Design and evaluate a heuristic for placing workflow tasks within each level so as to reduce the number of edge crossings
- Design and evaluate a randomized heuristic for further reducing the number of edge crossings

Approach

- Implement schema validations to take WfCommon workflow description JSON files as input
- Implement an in-the-browser display of arbitrary Directed Acyclic Graphs
- Implement a recursive algorithm to compute the level of each task in the workflow
- Implement a greedy heuristic for positioning each task in its level on the display
- Implement a randomized heuristic for greedily swapping tasks within a level to reduce the number of edge crossings

What Technologies are use for the project

Semantic UI: an open-source front-end JavaScript library for building user interfaces based on UI components

Gatsby: an open-source static site generator built on top of Node.js using React and GraphQL (provides over 2,500 plugins)

Cytoscape: an open-source graph visualization library that can display relational data and interactive graphs in SVG format

WfCommons: a framework for enabling scientific workflow research and development by providing foundational tools for curating, generating, and analyzing workflow instances

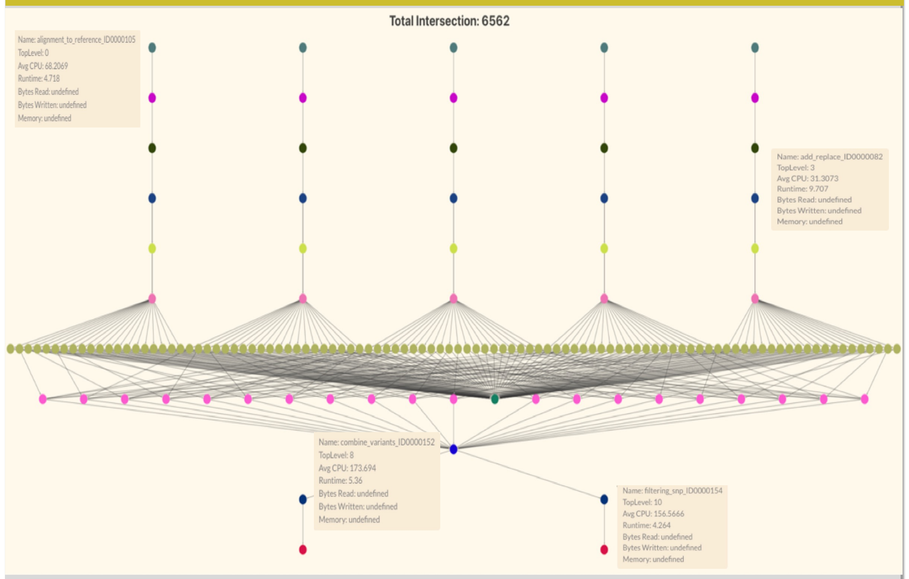
Cytoscape

WfCommons

Algorithms Developed

- **An algorithm to find task top-levels:**
  - A task's top-level is the length (in hops) of the longest path between the task and any workflow entry tasks
  - Computing the top-level of each task is necessary to arrange them level-by-level in the display
  - We implemented a bottom-up recursive algorithm, using memoization to reduce computational complexity
- **An algorithm to compute the number of edge intersections:**
  - Computing this number is necessary when implementing a heuristic for reducing it
  - Implemented an iterative **line-sweep algorithm** to compute intersections between line segments (i.e., edges that connect tasks)

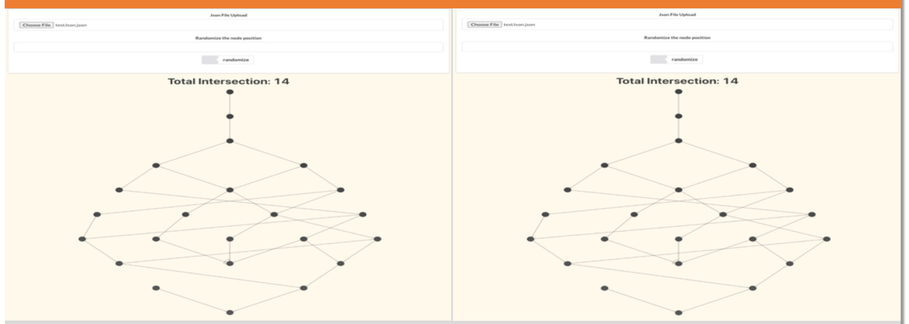
Sample Workflow Display



Using Cytoscape

- **Managing nodes and edges: Each Cytoscape vertex must be mapped to a workflow task, and edges correspond to parent-children relationships in the workflow. Each vertex in the display is movable by the user, and zoom-in/zoom-out is enabled**
- **Cytoscape Poppers: Each workflow task is described by many parameters, which are displayed on a mouseover. For better reactivity, all poppers are created initially, and then shown/hidden on mouseovers**

Task placement within each level



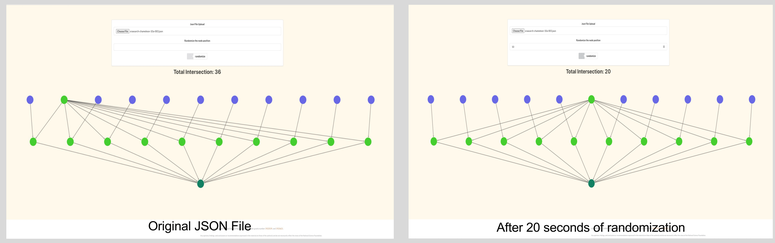
Task positions based on input JSON File

Computed task positions using heuristic

Task placement heuristic

- Heuristic used to sort all tasks in a level:
  - Given the x coordinates of tasks in the display for level n
  - For each task t in level n+1, compute x_t, the mean x coordinate of the task's parents that are in level n
  - Sort all tasks in the level by increasing x_t values
  - Arrange all tasks in the displayed level in this order, evenly spaced
- In the example above:
  - Placing workflow tasks based on the input JSON files leads to 14 edge crossings
  - Using the above heuristic reduces the number of edge crossings to 7

Further removing edge crossings

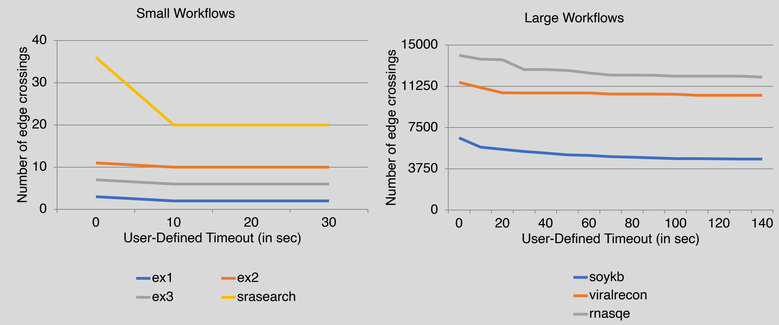


Original JSON File

After 20 seconds of randomization

Edge crossing removal heuristic

- Heuristic used to remove edge-crossings
  - While a user-set timer hasn't expired do..
    - Randomly pick two tasks in a level
    - Tentatively swap these two tasks
    - Recompute the total number of edge crossings
    - If the swap reduced this number, then apply it, otherwise try again
- In the example above:
  - Swapping two tasks in the top level of the workflow reduces the number of edge crossings from 36 to 20, arguably leading to a less cluttered display
  - This example is naive, but small enough to be displayed on this poster in a way that edge crossings can be seen without needing to zoom in



Findings

- Applying the randomized edge crossing removal heuristic to small workflows (left figure above, for 4 sample workflows) yields an initial decrease up to 10 seconds, after which no further improvement is observed
- Applying the randomized edge crossing removal heuristic to large workflows (right figure above, for 3 sample workflows) yields more pronounced decreases and benefits from relatively high user-defined timeouts

Future Work

- Optimize the implementation of the line-sweep algorithm
- Come up with ways to display workflows with extremely large levels
- Deploy the tool on the WfCommons web site

Lessons Learned

- Debugging in the context of Web development can be tedious
- Learned, painfully, the difference between shallow and deep copy
- Should have payed more attention in ICS 311 :)