**DestinyNET: A Javascript Library for the Visualization of Network Exploration**

Morgan Tenney

![Network Diagram](image)

**Fig. 1. An example of three-step exploration on the social connections between online username for the game “Destiny,” prioritizing players that had the most interactions with their peers**

**Abstract** - Understanding the structure of a network graph isn’t limited to analyzing them in their entirety. A lot of insight can be gleaned from the initial construction and exploration of the network, and how its paths change when different traits are prioritized. In this paper, we will describe the DestinyNET library, which seeks to enable easy and flexible exploration of any network. DestinyNET provides an easy way to ‘plug in’ an API and visualize the expansion of the graph, either manually or by following the given priority function. The user merely has to provide a function to retrieve neighbors for a given node, a priority function for deciding which node to explore next, and a starting node. With this, DestinyNET can provide a visualization of the graph as it grows and the relative priorities of the unexplored nodes, and allow the user the freedom to manipulate the graph and explore further.
1 - Introduction
Software for network visualization is well-explored and highly sophisticated. None of these programs, however, allow the user to experiment with the actual exploration and discovery of the network. How would the graph expand if the user employed a breadth-first search algorithm as opposed to a depth-first search algorithm? What would happen if they prioritized further exploration on different features of their dataset?
DestinyNET\textsuperscript{1} is a network crawler visualization that seeks to allow the user to answer any of these questions. The user can ‘plug in’ their own API, specify different prioritization methods, and witness firsthand the different paths the network growth follows.
DestinyNET was initially built using an API that pulled live data from the internet, and fully supports asynchronous data fetching; the user can use data fetched live as easily as data read from their local machine.

2 - Related Work
There is little to no work that occupies the role DestinyNET fulfills. Most network crawling visualizations are specifically crafted for the network they are exploring. Kirby Banman’s project “Semantic Web Crawling Visualization”\textsuperscript{2} comes close, since it explores different network crawling algorithms. However, it is built around a web crawler, and cannot be easily used to visualize any network. DestinyNET is flexible enough to be applied to any network.

3 - Methods
DestinyNET is effectively a wrapper for D3’s force-directed graph visualization. Rather than spending time on crafting the D3 visualization themselves, the user only has to create an API for retrieving the network data and plug it into the network crawler. The user provides DestinyNET with three things:

1. Neighbor Retrieval Function: a function which, when given a node, returns an array of all neighbors of that node
2. Priority Comparison Function: a function which, given two nodes, returns which node should have higher priority for further exploration
3. Start Node: the node object that the crawler should begin its search from
The user can then specify the number of iterations for the crawler to explore further. Ultimately, the user only has to write approximately three lines of code outside of their custom API in order to have a successful visualization.
What follows is a more specific description of how to set up DestinyNET\textsuperscript{3}:

1. Download the project, and write your custom API. If your custom API is in a separate Javascript file, ensure that it is included in index.html.

\textsuperscript{1} If you’re curious about the name, it is based off of the first network the tool was used to visualize; the social connections between usernames in the online video game “Destiny”
\textsuperscript{2} “Semantic Web Crawling Visualization” can be found on Github: https://github.com/kdbanman/visualcrawl
\textsuperscript{3} These instructions can also be found on the project Github page: https://github.com/Morgan10E/DestinyNet
Fig. 2. This is the structure DestinyNET expects for all of its node objects.

2. Modify main.js to use your API. The code should look of the form:

```javascript
var crawler = new NetworkCrawler(
    neighborRetrievalFunction,
    priorityComparisonFunction
);
crawler.setStartNode(startNode);
crawler.run(numberIterations);
```

Where

neighborRetrievalFunction and priorityComparisonFunction are user-defined functions that tap into the custom API written by the user. neighborRetrievalFunction must accept two parameters: first, the node object whose neighbors are being retrieved; second, a callback function that is called with the array of neighbors as its parameter - this is how the network crawler receives the request.

priorityComparisonFunction must accept two parameters, which we will call nodeA and nodeB. priorityComparisonFunction returns true if nodeA should have higher priority for exploration than nodeB.

3. Set the start node. Give the crawler a node object to use as a starting point.
4. Run the crawler. Give the number of times you want it to explore the next highest priority node, or `run(0)` to only explore manually.
5. Host the web page (ie. python -m SimpleHTTPServer) to view the visualization.

Now that you have the visualization open in a web page, you can watch and interact with the network crawl. The visualization has two primary components: the network graph and the priority queue. While in the visualization, there are a few interactions available.

Fig. 3. A blue node has already been explored. A green node is currently being explored, creating all the connections between neighbors. Hovering over a node in the graph or the queue highlights it in both the graph and the priority queue.
DestinyNET has shown is that there is more opportunity in the area of making compelling visualizations with no prior knowledge of the data being displayed.

5 - Future Work
The next steps for DestinyNET are focused on improving the aesthetic of the visualization. First, making some changes to the force-directed graph so that the graph better represents the data and connections, since as of now the nodes become far too spread out during the exploration phase. In addition, the color choices in the visualization were meant to make changes and interactions obvious, rather than aesthetically pleasing.

The priority queue will also have some additional work. Ideally, it should be made more clear when items have been newly added to the priority queue as opposed to simply increasing in rank.

Providing the user with easier ways of customizing the appearance of the network graph or priority queue is also a primary goal. For example, similarly to how the optional data object allows the user to display information about a node, there could be an optional appearance object that allows the user to specify the node’s appearance in certain states. The user could also define custom force graph functions if they wanted different behaviors - the current default is to assume that a higher value on the link between nodes implies that the links should be closer, but that isn’t the correct approach for all network graphs.

The focus of DestinyNET remains allowing the user to experiment with exploring any network easily. The future development of DestinyNET will prioritizes that further that goal.

3 - Results
Using DestinyNET, users were able to get a network crawler up and running from scratch within thirty minutes.
It must be acknowledged that the actual visualization itself has some shortcomings. The visualization is not particularly unique or innovative - it is the tool and library built around it that makes it easy to explore and experiment that is unique and innovative.

4 - Discussion
The beauty of programs such as Tableau is the flexibility - the ability to give it any dataset and discover the nature of the data through the visualizations, without needing any specialized software.