Art Orchard

Darius Bopp*
MIT

Diana Nguyen†
MIT

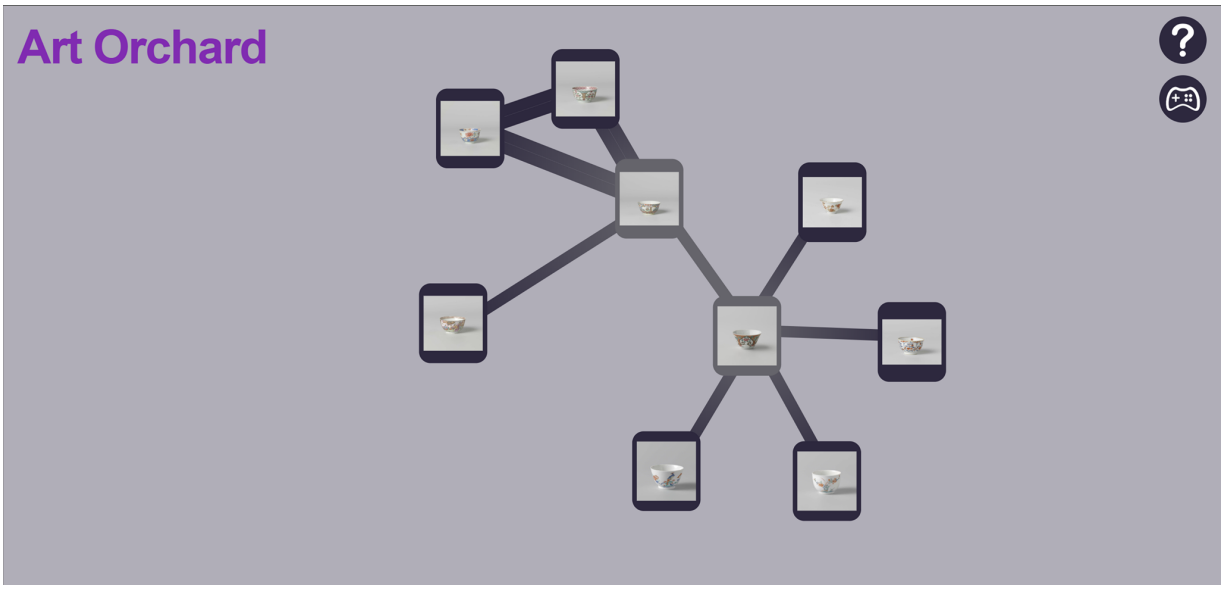

Figure 1: Image of the Art Orchard website

1 Introduction

Art museums can be intimidating to walk through. Visitors are surrounded by hundreds of beautiful and unique artworks laid out in groups of cultures or mediums. The divide between groupings of art makes it difficult to imagine how art could potentially be connected by human nature.

How is artwork connected through cultures and mediums? How are artists influenced throughout history? What inspirations did artists have? These are all questions that Art Orchard strives to answer. More specifically, Art Orchard aims to show connections between art from the Metropolitan Museum of Art (MET) and the Rijksmuseum, using a graph structure based on a similarity algorithm. To further motivate exploration of similar artwork, Art Orchard provides a WikiRace format to find pairs of similar artwork.

2 Related Work

We were inspired to complete this project due to previous externships in collaboration with Microsoft, the MET, and the Rijksmuseum. Previous work on visualizing and interacting with artwork from those museums include Gen Studio [1] and Mosaic [2]. Gen Studio used data from the MET museum's collection so that users could interact with a Generative Adversarial Network that created new artwork based on features from currently existing artwork. Mosaic allows users to search through the MET and Rijksmuseum collections to find similar artworks filtered by culture and medium.

Previous work was done in collaboration with Mark Hamilton, who contributed to both websites. Hamilton graciously provided his similarity algorithm [4] used in Mosaic for use in Art Orchard. The similarity algorithm is unique in that it allows for filtering by culture and medium, in addition to finding "similar looking" art.

3 Methods

Art Orchard was created using cola.js [3] with D3 to create the graph of the art similarities. The similarity algorithm has an API in which matches are provided given an image url. In order to populate the graph, an initial artwork is chosen along with 5 of its nearest neighbors. For each neighboring node, its neighbors are also retrieved to get hints about how many neighbors would come out of expanding the node. This outer layer of nodes is hidden so that the user experience is instant: when a user clicks on a node, that node's neighbors appear right away.

We also have two sliding sidebars to display information to users. The first one, identified with a question mark, is an information tab, explaining the controls of the visualization, and the source of the data and underlying algorithm. The second tab, identified with a game controller, is the game tab, which explains the goal of the game portion of the website, along with displaying the start and end artwork. The tab also includes a button to start a new game, which involves getting a new start and end artwork. Game pairs were generated with an offline version of the data with a simple graph exploration algorithm.

4 Results

We managed to fully build out the website, albeit with some limitations. As with any software, there are some bugs. Sometimes

*e-mail: dbopp@mit.edu
†e-mail: nguyend@mit.edu

edges get doubled up when exploring the graph. When trying to move nodes on the graph, most of the graph moves fluidly with the movement. However, the original root node is stuck in place, limiting it so the only way to move the entire graph is to move the root node itself.

The biggest limitation of the current implementation is the API that is queried for the data. We originally had issues with CORS policy, and set up a middle-man server with heroku to handle the requests. Despite this, sometimes the API responds with a 503 Error, claiming it is overloaded. This results in nodes not having the neighbors they should, and cascades into nodes appearing purple, but not actually doing anything when clicked. This also means that sometimes the game portion of our website is impossible to complete. Since we are not running the API, we are not able to directly affect the server hosting it. To try and mitigate the errors, we self-limit requests to about 8 requests every 10 seconds. This self limiting also causes purple nodes to not have their neighbors yet, but increases the number of purple nodes that will eventually get their neighbors.

5 Discussion

We feel like we have succeeded in what we set out to do. Despite networking issues, the website enables users to explore a graph of art based on their similarity, and grow the graph to get interesting results. Users can interact with the graph-space in nearly any expected way (pan, zoom, move) and extra information is easily found.

We also recognize that our current system is very feature limiting. Users are not able to select their own starting artwork, and users are unable to specify expansion along specific queries, such as culture, medium, or time. These limitations come from some technical and design restrictions. To make clicking nodes real-time reactive, their neighbors must be pre-fetched. This makes it hard to give users the freedom to choose their own query type. Since the data set is very large, we opted to have a narrow finite set of options, since we did not have a reasonable way to open up the whole space to the user.

An interesting question when making similarity graphs is where you place the cutoff for whether or not an edge is drawn between two nodes. We arbitrarily chose to connect the top 5 results from the API, in an effort to not let the graph get too cluttered. However, this adds false structure to the graph. You may see cliques of some art, and wonder if there is a reason other similar art are not included in that clique, when it is just because of arbitrary cutoffs. A better system may be to query for a large number of artworks, but create a cutoff based on the similarity metric instead of just taking the top few results.

6 Future Work

Art Orchard can be improved in two ways: its performance and its features. The results section discussed the limitations of the similarity API, which was the bottleneck in the visualization's performance. Although we are not in charge of maintaining the API, work could be done in collaboration with the API owners to improve its performance. We could also create our own API and manage it ourselves, however that is out of the scope of this project. One other alternative is to fully precompute all similarities to avoid waiting on web requests. This was attempted, but the dataset proved to be too large (3GB) to handle on our machines.

Many additional features could be added to improve the scope of exploration on the Art Orchard website. The similarity API allows for filtering results by culture and medium, however we were not able to take full advantage of that due to time constraints. A way that we could use that feature is to add in filtering by cultures and mediums, and show a graph where connections are all different from the original culture/medium. This would allow users to explore more unique connections between artworks. Another extension could be to add in a search option to allow for exploration of the full MET and Rijksmuseum dataset. Users could search for specific objects and explore similar artworks for those selected objects.

References

[1] *Gen Studio*. `https://gen.studio/`.

[2] *mosAIc*. `https://microsoft.github.io/art/`.

[3] *Online Graph Exploration - Incremental exploration of a large graph*. `https://ialab.it.monash.edu/webcola/examples/browsemovies.html`.

[4] M. Hamilton, W. T. Freeman, S. Fu, and M. Lu. *Fast, Conditional, and Interpretable Nearest Neighbors for Semantic Alignment*, vol. 108. PMLR, 2020.