Process Book – Data visualization 2017

Mattia Martinelli Niccolo Sacchi Dario Pavllo

Contents

1 Introduction

Buying from huge e-commerce websites such as *Amazon* has many advantages, but paradoxically, users are often confused by the vast variety of products that are offered. Users may have a rough idea about the characteristics of the product they want to buy, but they often undergo the same process of comparing similar products. We aim to remove this redundancy and aid them in their purchases, suggesting the best or most popular products that correspond to their search. For instance, comparing smartphones or laptops may be difficult due to the wide price range and the required technical knowledge. With our platform, users can easily search in a graphical way the most popular products that match their needs.

1.1 Idea and related work

Amazon's website already offers a sophisticated search system, which allows users to select category, price range and some technical characteristics of the product they want to buy. But would it not be nice to query among similar articles without explicitly providing to the website such features? For instance, product description pages already contain relevant information about related articles, such as *similar items*, *items bought together* and *items that customers buy after viewing this item*. These links suggest alternatives to the customer; however, they only show *neighbouring* offers and not the whole overview of offers. To overcome this limitation, we have decided to develop a graphical and interactive visualization that shows such relations among multiple articles.

On our platform, products are shown as a network (Figure 1), where it is clear to see which are the items that users usually end up buying. Imagine a user who wishes to buy some professional studio headphones and has a rough idea of their characteristics. By typing some keywords, he/she can highlight relevant products and follow the highlighted path towards an optimal product. Of course, defining which articles may be *optimal* is not a trivial task; however, if many clients buy the same product after reviewing a set of other products, then it is very likely that the former is more appealing.

There may also be groups of products in which it is not possible to identify an optimal one, i.e. users are often uncertain in choosing one of them. Each one of these groups represent a cluster of *competing products*, which allows users to find alternatives to their search.

In conclusion, we aim to exploit product relations produced by previous users' queries to help new users in their search, in a way that is not (directly) available on Amazon.

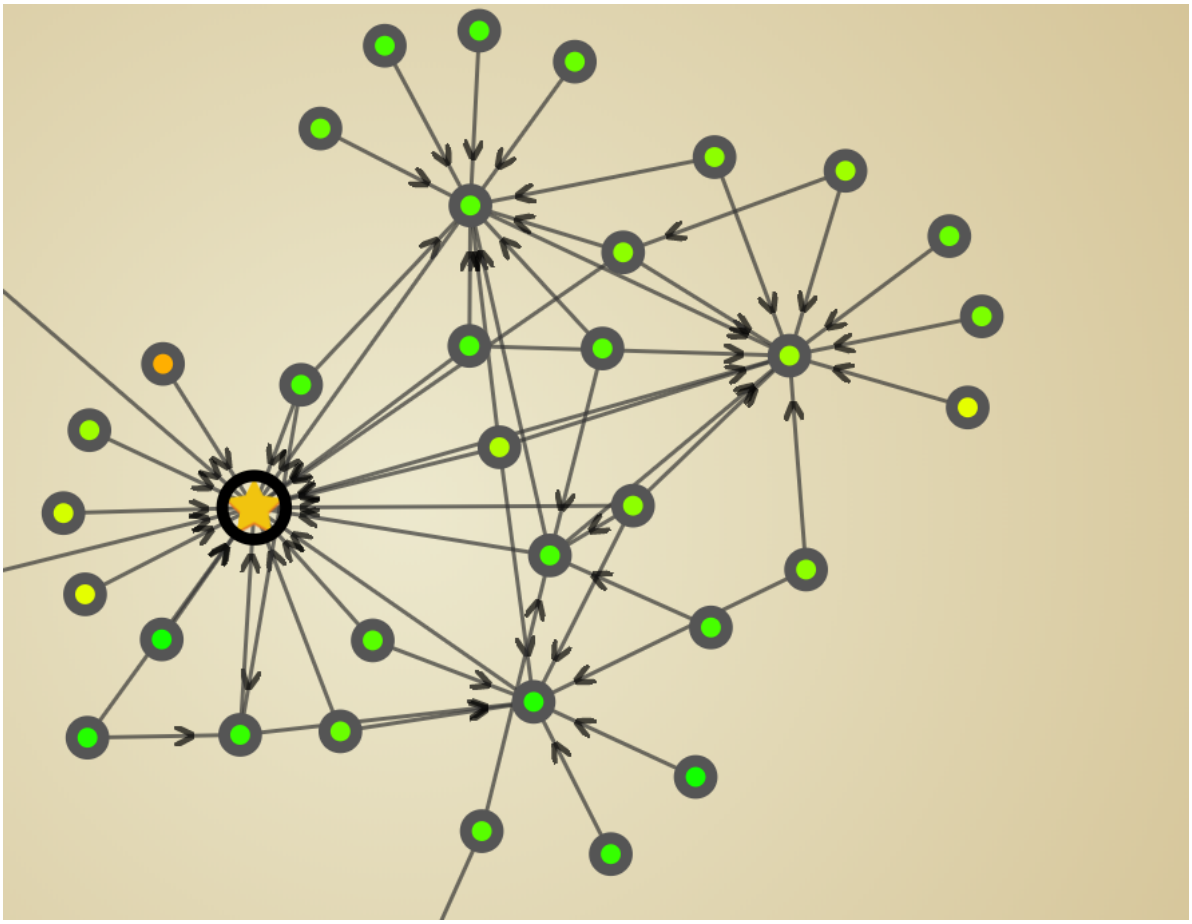

Figure 1: Subgraph of the *Headphones* category. As can be seen, some nodes (products) are *attractors*, in the sense that users end up buying those after visualizing a large number of other products.

1.2 Assumptions

At this point, it could be argued that graph edges provide meaningful relations. We supposed that when users navigate through Amazon's offers, they already have an idea of the product that they want to buy. It can be a technical characteristic, a brand or merely a budget. Our dataset should therefore provide relations of products that must be similar according to some *human-made* criteria, which would be difficult to extrapolate with an algorithm or a machine learning model.

1.3 Target audience and use cases

With our platform we want to address both customers and vendors. Customers may take advantage of previous people's choices to decide what product is worth buying, or to get an idea about alternative items. On the other hand, vendors may explore the visualization to search competitors to their products and see the characteristics of the most popular items in category.

2 Dataset

We have been provided with a dataset of Amazon products that contains the following relations among the articles: "bought together", "also viewed", and/or "buy after viewing". Section 3 explains how we have exploited such relations to create our network. The dataset contains products from every category on Amazon, but we have decided to work exclusively on selected ones. We have focused only on categories where it is possible to compare products in an objective manner, and where the purchase choice does not merely depend on users' personal preferences. In the latter case, examples are clothes, music and books. Finally, we have built our platform upon the following macro-categories: *Electronics*, *Cell Phones and Accessories*, *Automotive*, *Tools and Home Improvement*, and *Musical Instruments*.

2.1 Description

The dataset consists of two JSON files:

- *metadata*: contains information related to the products, such as their unique ID (*ASIN*), category, description, *sales rank*, brand, price, and relations with other items. The size of the file is 9.81 GB (uncompressed).
- *reviews*: contains ratings and reviews associated to each product, as well as the helpfulness of each review. The size of the file is approximately 87 GB (uncompressed).

2.2 Preprocessing

Due to the large size of the files, we decided not to include individual reviews. We created a single dataset where, for each product, we extracted the average rating, number of reviews, and their *helpfulness* score. These fields have been then merged with *metadata*.

Additionally, for the purposes of the visualization, we further reduced the size of the dataset. Some categories contain several thousands of elements, but a real-time interactive visualization is able to handle at most a few hundred elements (and the bandwidth requirements must as well be considered). Instead of randomly sampling the graph, we came up with a cleverer approach. Initially, we label the connected component of the graph, then we remove the smallest ones until we reach a target graph size (300 to 500 nodes per category). This approach, while achieving the goal, ensures that the most relevant relations are preserved in the reduced graph (that is, we remove only products that have few relations with other nodes).

It is worth noting that our visualization does not rely on server-side scripts or databases (we use GitHub pages). For this reason, we preprocessed all the data in local and exported a set of JSON files. Each (sub)category (or, equivalently, each graph) corresponds to a specific JSON file of 200-300 kB that is loaded dynamically (and asynchronously) during the category selection. The file `categories.json` contains the index of the categories (tree and URL of every dataset).

3 The graph

The dataset has been transformed into a directed graph, where nodes represent products and edges represent *competitions* between products.

3.1 Structure

The graph structure follows precise rules. An edge from product A to product B is added if clients **buy** B after viewing A (*buy after viewing* relation), but such edge is removed if A and B are frequently bought together (*bought together* relation). The former means direct competition, i.e. an article has been preferred over another, while the latter means no competition, i.e. the two articles are complementary (e.g. a cellphone and a cover). In our context, a directed edge from A to B means "B is preferred over A", whereas an undirected edge (or, equivalently, two opposite directed edges) means "A is competing with B". It is easy to extend this definition to groups of competing products, that is, max-cliques. If some groups are totally interconnected, we can assume that they are in direct competition and that one is not necessarily better than the other. We explicitly show them in our visualization.

In previous experiments, we tried to build the graph by adding edges between products that are viewed together (*also viewed* relation). This relation does not imply that any of the products has been actually bought, and it produces a graph that is too dense to give meaningful results.

3.2 Insights

By visually inspecting the graph, it is possible to identify some common structures, which are depicted in Figure 2.

- **Accumulators:** these are popular products that have many incoming edges. To put this into numbers, we can associate a fan-in metric (or in-degree) with every node, and, expectedly, products with a high fan-in tend to be preferred over those with a lower one.
- **Max-cliques:** groups of products that are totally interconnected. In many cases, these products are also accumulators. Cliques represent products that are in direct competition with each other (and it is not really clear which one "wins"). Note that these competition relations might even comprehend products of the same brand.

We have inspected such figures to insure that they have a meaningful representation, both manually and by means of specific algorithms. We could confirm that accumulators represent popular products and cliques represent products with very similar characteristics.

4 Sketch design

The section shows our original design idea and how we have devised the visualization. Simple sketches that we have made prior to start the implementation are also proposed.

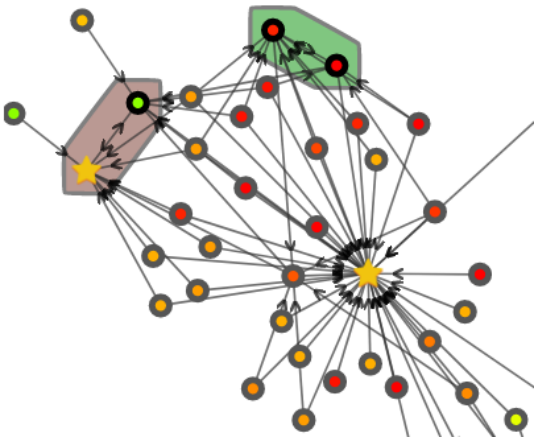

Figure 2: Portion of the graph of a category. The stars represent accumulators, whereas the coloured halos represent cliques.

4.1 Preface

Before visualizing the graph, the user must choose a category. We decided to keep categories independent for the following reasons:

- The visualization is cleaner and faster to process. If items of multiple categories were mixed, the visualization would result slower and more difficult to interpret.
- Items in cliques usually belong only to a single category. Indeed, it would be meaningless to cluster similar products if such products belong to different categories.
- Users are usually interested in one specific category, and its choice represents a pre-filtering mechanism.

4.2 Navigation process

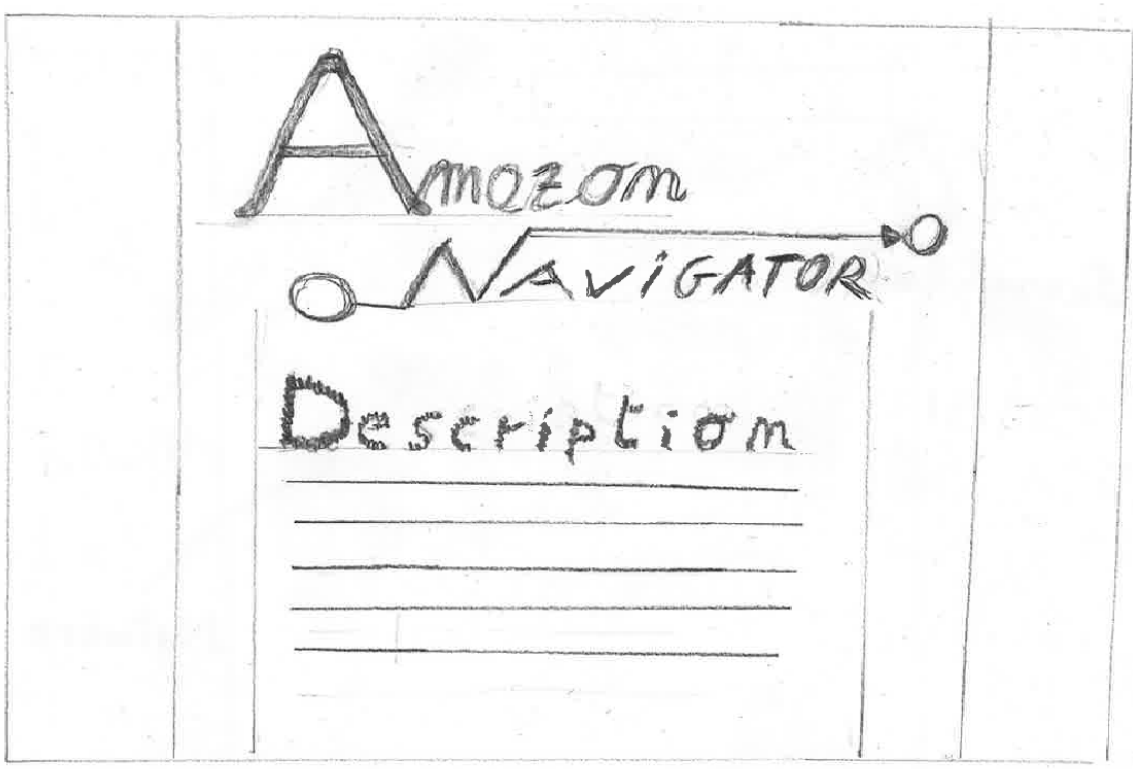

Figure 3: Introductory screen of the webpage.

In summary, the visualization guides the user from their idea of product to the final result. In particular, the process should be as follows:

1. As an introduction, the user is presented with a brief and intuitive description of the project (Figure 3). Then, a quick tutorial will be also included to explain how to use the platform.

2. The user is presented with a selector that navigates through the Amazon category tree (Figure 4), and allows him/her to select a category (e.g. headphones, mobile phones, laptops, etc.). It will also be possible to search for a category according to some keywords.

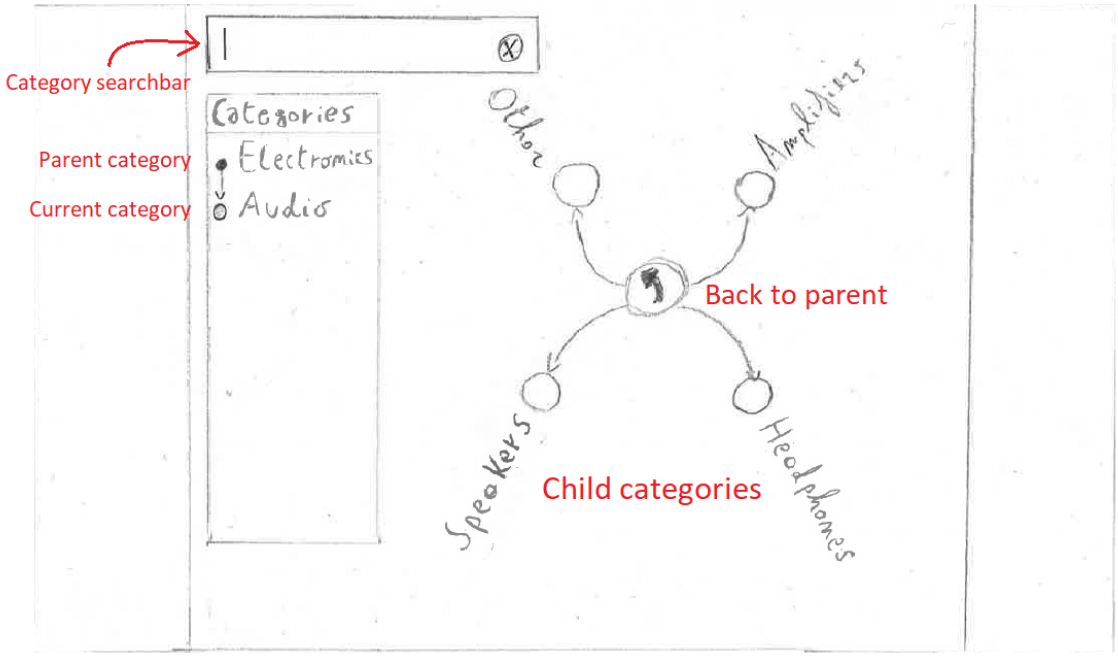

Figure 4: Category navigation.

3. Once a category is selected, the graph view appears (Figure 5). Initially, the full graph is shown, so that the user can get a sense of its topology (sparseness, attractors, cliques, etc.).

Figure 5: The whole graph is shown when a category is selected.

4. The user can query a few keywords and set a price range, which will cause the graph to highlight only relevant parts and collapse the rest (Figure 6). At this point, the user can inspect the paths between products, as well as the *attractors*. The visualization will also provide some recommendations (automatic paths) and display the characteristics of the products, along with their differences.

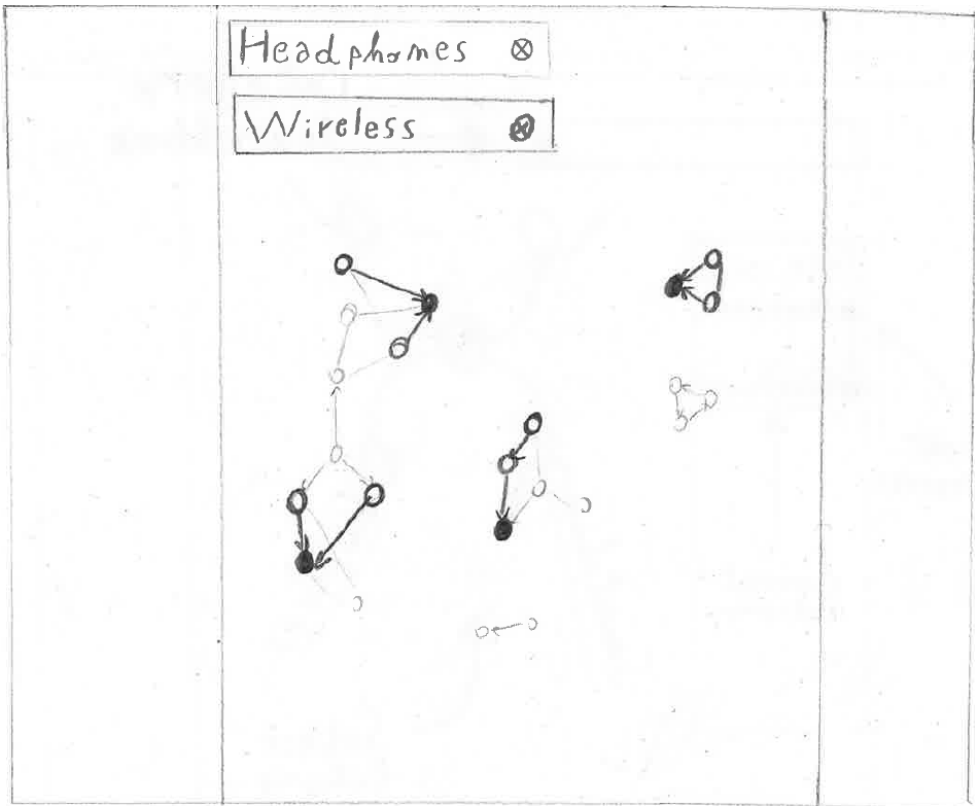

Figure 6: Example of products filtered according to keyword.

5. The visualization provides some recommendations, showing title, picture and a brief summary of their characteristics (Figure 7). Such recommendations are *attractors*, that is, nodes with the highest incoming edges. Therefore, they are considered the most popular products in category.

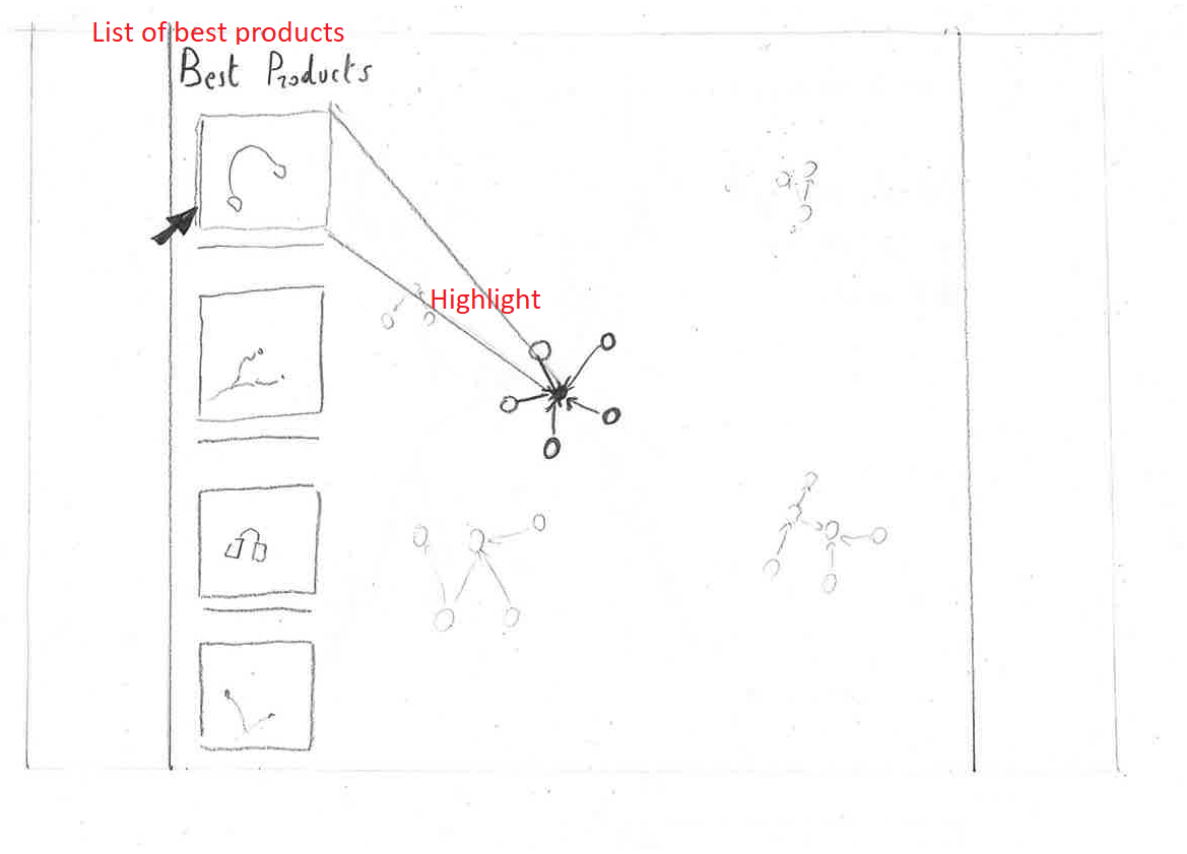

Figure 7: Best products are directly shown to users, but they are also highlighted on the graph.

5 Implementation

5.1 Graph algorithms

A question easily comes to mind: how do provide recommendations to users? More specifically, how do we decide, for every product, the path that leads to the best product? What constitutes a "best product"? According to our idea, products with many incoming edges (fan-in metric) are regarded as superior by people. However, using a simple threshold does not produce satisfying results, as its choice is crucial (and could be different for each category). Instead, we use a graph visit algorithm that works as follows:

1. We sort the nodes (products) by fan-in in decreasing order.

2. We start with the node that has the highest fan-in and we mark it as "best product". Afterwards, we run a BFS (breadth-first search) graph visit from that node, following all edges in reverse order. All visited nodes are labelled so that they point towards the best product. Additionally, the BFS gives us the shortest paths to the latter.

3. We move onto the next node in the list, and, if it has not been visited, it is marked as "best product" and expanded in the same way as before.

4. The algorithm ends once all the nodes in the graph have been labeled.

This approach is very efficient as it runs in linear time (except for the sorting), and produces very reasonable results. Indeed, we observed that the paths comprehends products with similar characteristics, and a certain path could intuitively be interpreted as the "average path that people follow from the first product they visit to the one that they buy".

5.2 Visualization

The graph must show to users the most relevant products in an intuitive manner. As a crucial requirement, users should find the information of interest "at first glance", without needing to interpret the graph. However, making intuitive a graph is not a simple task. In order to achieve the goal we undertook the following steps:

1. *Product Graph choice.* We initially opted for a collapsible graph (d3 force layout with bundling, as depicted in Figure 8), which would allow the user to "hide" all the competing products into one node and expand it only if the user is really interested in analyzing them.

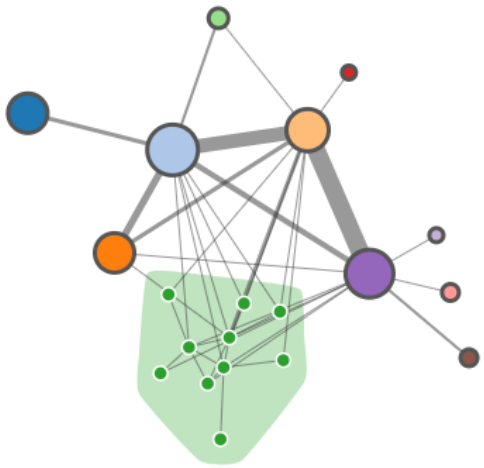

Figure 8: D3 force layout with bundling.

2. *Product Graph: code cleaning and improvements.* After diving in the code of that example we realized that there were several problems. Changes were needed in order to satisfy all our requirements since the code was not versatile, inefficient (e.g. the graph was initially recomputed every time a node was collapsed/expanded) and it used d3 v2. We needed both a structure that would allow us to easily update the graph (exclude/include nodes) and to maintain the code easily. Therefore, we restructured most of the code implementing classes (which improved versatility and manageability), removed a couple of features we did not need and improved the efficiency. Then, we converted the code to d3 v4 (it was not an easy task, as we discovered that d3 v4 is not much retro-compatible) so as to obtain something more versatile and efficient. However, we understood that collapsing and expanding a node would have implied non-trivial changes to our graph algorithm, i.e. finding and highlighting the path to the best product would have been

difficult according to the expansion state of a group. Moreover, the cliques are small and few and the time spent in implementing ad-hoc graph algorithms to keep collapsible nodes would not be worth much. So we decided to indicate the cliques with polygon hulls (d3.polygonHull) so as to keep indicating them. Finally we added arrows to edges, since the direction of the edge is of major importance in our algorithm. We ended up with a graph where cliques were well distinguishable but it was still not very intuitive as the user would not see products but only dots connected by edges.

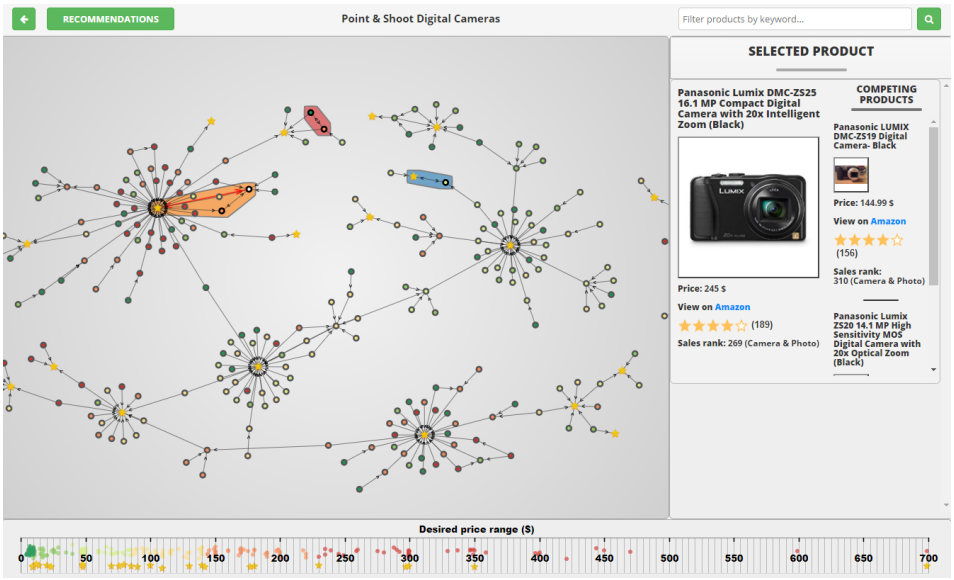

Figure 9: Products graph UI.

3. *Product Graph: add interactivity.* Once we display all the products of a category as a graph the next step is to choose an intuitive and easy way to explore and/or select only the parts of the graph in which the user is interested. To accomplish the task we added several features.

   - When the user *mouseover*s a node, we show the photo, link, and other information related to the product. Moreover, we highlight the path to the best product, i.e. a star.
   - We add a search box to filter the graph using a keyword. This search box has auto-completion and it suggests the words that have been extracted from the title of the products, so that the user does not end up in selecting an empty graph. In particular, we show not only the products related to that keyword but also all the reachable nodes and the direct parents. This way, we both show all the better products and also the fan-in (also referred to as in-degree) of the filtered products (note: the fan-in is the metric we choose for selecting the best product between competing products, i.e. products belonging to the same clique).
   - We show in a hamburger menu the list of computed best products of the whole graph. Those products are also indicated by stars on the graph so to show that the best products are the ones with many incoming edges.

- The nodes are colored according to the price of the corresponding product. In particular, we adopted a 6-class RdYlGn (red-yellow-green) diverging color pattern (obtained from ColorBrewer) that is mapped to the products through a quantile distribution, as the distribution of prices is not uniform. To do this, we sort all the product prices, split them into six bins and then give a color to each bin. However, outliers, e.g. products with very high prices, end up in having the same color of products with a very different price. To reduce this problem we add a brushable price bar with which the user can focus on the products whose price belongs to a specific interval. Notice that we keep all the products (but we grey out irrelevant ones), since filtering out the products that do not belong to the interval could also filter out the best products, e.g. the user could select the price interval 50-70 while the best product costs 40$.

4. *Category Graph.* Allows the user to select a category and have an idea of the number of products. We implemented an expansible radial tree accompanied by a list/timeline (on the left) of explored categories. The result is shown in Figure 10.

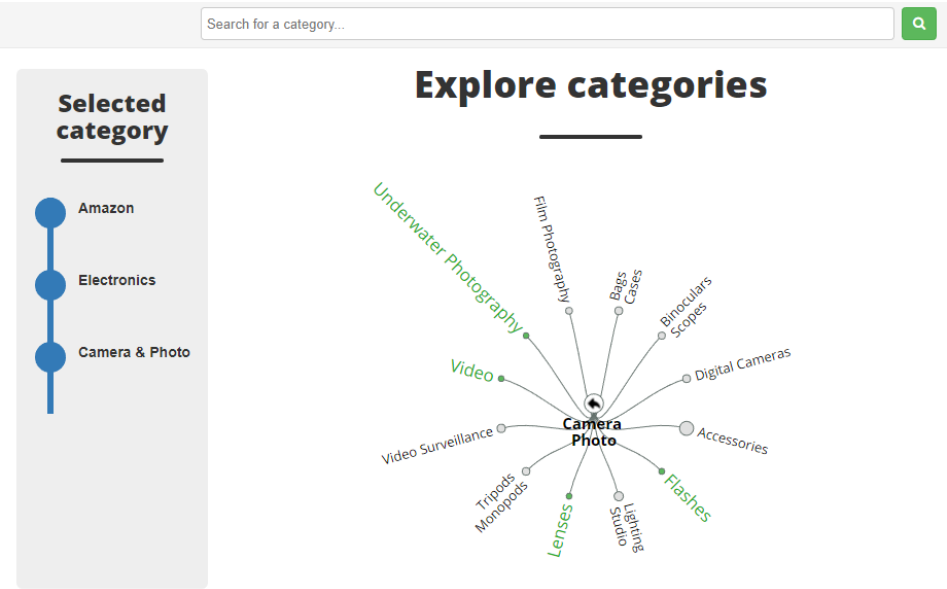

Figure 10: Category selection UI.

5. *Merging.* Once we implemented both visualizations, we merged them into a single easy-to-use visualization.

6 Conclusion

We now conclude the process book self-evaluating our visualization and proposing further improvements.

6.1 Evaluation

Thanks to our visualization, we could clearly see that the Amazon product graph has a very structured topology, consisting of accumulators, connected components that are sparsely interconnected (i.e. clusters), and cliques. The visualization allowed us to observe many behavioural patterns that users adopt when choosing the products to buy. For instance, we initially thought that users would (almost) always buy the cheapest product in a group, but we actually observed that clusters tend to have the same price range (and there are consistent price variations among different clusters).

In some categories, we observed that interpreting the graph is cumbersome because of the high number of overlapping edges, despite the graph being sparse. We tackled this issue by adding the possibility of filtering the products, but we decided anyway to keep the full visualization to give the user a "feel" of the graph.

6.2 Further improvements

Our visualization has some room for improvement. The following points could be addressed:

- Performance: the force layout has a high computational cost ($\mathcal{O}(n^2)$) by definition, and this limits the number of nodes that can be displayed at the same time (a few hundreds at most). Nonetheless, we believe that some clever tricks could increase this limit. Another option would be to pre-compute the node positions using an offline algorithm, although this would limit interactivity.

- Design: although the graph meets the functional requirements, its design could be improved. Apart from stylistic characteristics (which are subjective), some dense areas of the graph are not easy to interpret. In some cases, the high number of edges requires zooming in order to "untangle" the visualization. We thought about possible solutions, and all of them involve some kind of trade-off that would reduce the information visible on the graph. Perhaps, some hierarchical subdivision of the graph would render it more clear.

- More data: a server-side component could provide the user with the full graph, instead of a small sample of it. Search queries could be processed on the server, and the result would be returned to the user.

- Guided tutorial: it could be useful to show the user some tooltips in the graph visualization, describing how certain patterns (e.g. cliques) should be interpreted.

Additional notes

Peer evaluation

- Preparation: we have always shared all the necessary information by means of online communication channels, and we have never let a team member to be not up-to-date with the development status.
- Contribution : every member of the team has put great effort into keeping a constant and uniform pace during the development of the project.
- Respect: we have always encouraged different ideas, even though sometimes it might have been tough to reach a consensual agreement.
- Flexibility: Overall, no team member has never complained about criticisms or divergences. We have always tried to ensure consensual agreement, and we have never imposed a solution over another.

Work disclaimer

In the initial project proposal form, we stated that our Data visualization project was jointed with our Applied data analysis project. Although we have deployed the same dataset in both courses and we reused parts of the data processing pipeline, the goals (and the development) of the two projects have significantly diverged. In this course we have focused merely on the visualization of the graph, whereas in the other one we have exploited it for a deep analysis (machine learning and statistics). In addition, for this course we needed some different and more suitable preprocessing of the data for the visualization.

Furthermore, the Amazon dataset is not public and is available only under request. For this reason, we did not upload it under a GitHub repository (so as to avoid copyright issues), but made it available under a different link.

The website is partially based on Beautiful Jekyll, a template generator.