B-Matrix Explainer

Matias I.B. Oddo - 2022.10.19

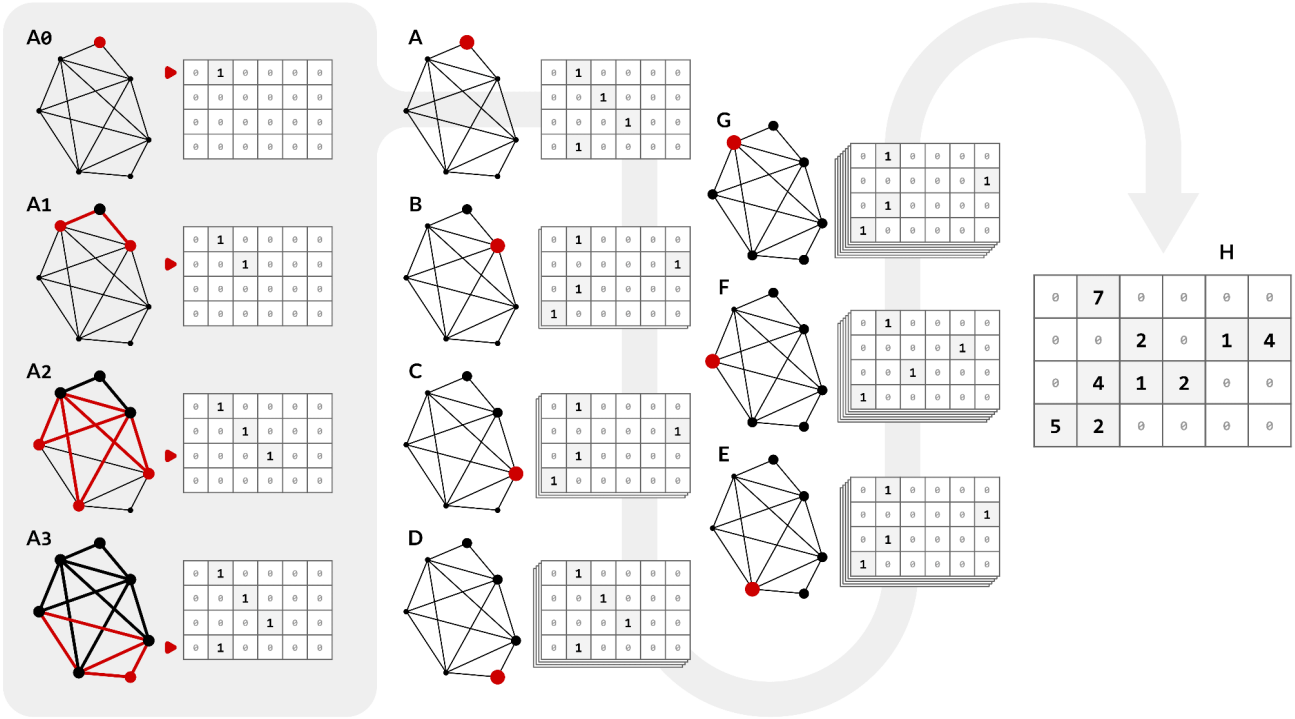

Figure 1. High-level step-by-step process of the B-Matrix algorithm for an example graph with 7 nodes and 14 edges. The final B-Matrix is shown at **H**, where rows are the number of nodes reached at a given hop and columns are the number of hops away from a starting node, and values within cells are node counts. Both axes of the B-Matrix use index zero. The algorithm starts by picking one of the graph's nodes, shown in **A** with the selected node highlighted in red. To create the bit matrix through Breadth-Search First for **A**, start at **A0** at the zeroth hop, where we only have one node at hand (the already selected starting node) so we bit-flip the cell at the zeroth row and first column. Then at first hop **A1** we move to the first row, and bit-flip at the column index of the number of nodes encountered, in this case two. Then at second hop **A2** we encounter three nodes, so we bit-flip the cell at the second row and third column. Finally at **A3** the third hop traverses the entire graph with only one node left, so we bit-flip the third row and first column. This process is repeated for every node in the graph, each node is a starting node once and gets a bit-matrix, as shown in **B**, **C**, **D**, **E**, **F**, and **G**. When all nodes have a bit-matrix, all matrices add together element-wise into an array of integers **H** (the B-Matrix) and the algorithm ends.

Abstract — This UBC CPSC 547 Information Visualization project about the B-Matrix, a graph invariant data abstraction about the structural properties of a network. Canonically visualized through a colormap heatmap idiom, this paper proposes alternative visualizations including a bar chart histogram idiom and row-normalization to enhance information fidelity and discovery. This paper has a companion GitHub repository github.com/direditedock/BMatrix_Explainer that features a comprehensive README.md visual explainer, network dataset examples, figures, and Python code for the generation of a B-Matrix and high resolution visualizations as described in this paper.

Index Terms — network analysis, network visualization, network portrait.

1 INTRODUCTION

This CPSC 547 project is about visualizing networks, and while it all worked out at the end, it had a rocky start. In the beginning I was interested in visualizing information from Wikipedia, so I made a Depth-First Search (DFS) recursive scraper for Wikimedia API to extract knowledge networks hidden in semantically rich fields within infoboxes. An infobox is a specially structured format with shared front-end and back-end content, making infoboxes both human and machine readable. My original project goal was to interlink infobox networks to fill information gaps, and then create a human-in-the-loop tool for Wikipedia editors. I started with a DFS recursive scraper, and then I added more bells and whistles such as an API solver for synonymous page redirects and an HTML regex system to isolate outgoing semantic fields such as *Influenced* from incoming ones such as *Influences* and *Influenced by*. Longstroy short, I ended up automating myself out of a job, and learned that software that gap-fills Wikipedia is not fit to be a CPSC 547 project. Infobox interlinker does work though, project progress and code can be found at `github.com/direditedock/infobox_interlinker`. Two knowledge networks came from the infobox project, one scraped from `wiki/Fortran` for a graph about programming languages (471 nodes, 1169 edges), and a much larger one from `wiki/Carl_Jung` about influential thinkers (6907 nodes, 16592 edges). With these networks at hand, I wondered what other ways are there to understand and better work with information dense networks. A paper on the topic of information-based network comparison has a "real world" network that looks similar to my Wikipedia networks,

but it also has a companion matrix visualization that I have never seen before, called a B-Matrix (Figure 1 within both [1] and [2]).

Also known as a network portrait, a B-Matrix is a graph invariant data abstraction about the structural properties of a graph [1]. It has practical application in network analysis, where portrait divergence is a metric of similarity between two networks based on B-Matrix intercomparison [2]. While in literature both B-Matrix and network portrait are used interchangeably to refer to both the matrix itself and its visual representation, this paper disambiguates B-Matrix to strictly mean the data abstraction, and network portrait to mean the visual representation. This distinction facilitates discussion of alternative visualizations of a B-Matrix.

The horizontal axis of a B-Matrix is for the number of nodes reached by the algorithm at a given step. The vertical axis is the number of hops away from a starting node that the algorithm runs through. Both the x-axis and y-axis of the B-Matrix have index-zero and start counting from the zeroth column and zeroth row in `+1` integer steps. Each cell of the B-Matrix begins as `0` and is modified by the algorithm through bit-flip from `0` to `1`. The last step of the algorithm involves adding all of the bit-flipped cells element-wise, so the B-Matrix always contains integers. Figure 1 illustrates the B-Matrix algorithm using a small graph as an explainer. Breadth-First Search (BFS) is the core algorithm for node accounting in B-Matrix creation, a sibling solution to the DFS approach deployed in infobox interlinker.

The B-Matrix of a graph has interesting properties. First, all rows add up to the total number of nodes in the graph. Second, the zeroth row is always empty except for the first column cell, which has a value equal to the total number of nodes in the graph. Third, the first row records the number of times different node counts happened at exactly one hop away from the starting node. The node degree is how many edges a node has, so the first row is effectively a record of the frequency of all node degrees in the graph. Fourth, the last row is the maximum number of hops the algorithm got through before running out of nodes. In other words, the final hop number, or the height of the matrix, is equivalent to the diameter of the graph. The graph diameter is the length of the shortest path between the two most distanced nodes. Fifth, B-Matrix construction and cell order is graph invariant, so axes labeling and interpretation is always the same. Finally, because each cell has node counts, the matrix is always made out of integers. So far in literature the B-Matrix is visualized through mapping a colormap to the range of integers in the matrix, resulting in a heatmap idiom.

This paper has two contributions. First, an alternative view through bar chart econding resulting in a histogram idiom with high-fidelity monochromatic rendering, useful for visualization of B-Matrix under challenging color displays. Second, row-normalization to reveal patterns and enhance information discovery within the hop level, a per-row transformation that is applicable to both heatmap and histogram B-Matrix views.

2 RELATED WORK

B-Matrix has been used as a comparative metric in scientific research and as part of visualizations systems in-itself since B-Matrix introduction in 2008 [1], including exploratory research such as visualizing the community structure of Brazilian musicians [10] and portrait divergence in characters, locations, and keywords said across networks built from Star Wars movie scripts [12].

Citing a preprint of [1] from 2007, [15] is the earliest practical implementation of the B-Matrix algorithm, used in urban planning research as comparative metric for network-converted street plans of different cities, and using a rainbow colormap to visualize B-Matrix node counts as cumulative probabilities. Between 2011 and 2017, Wojciech Czech contributed to B-Matrix theory by expanding the algorithm to be edge-based instead of vertex-based, and devised an optimization of the BFS approach by pre-computing node distances in some graph cases [4,5,6,7,8]. In 2010, [9] used B-Matrix alongside motif distribution to investigate network topology evolution. All B-Matrix in [9] use a reverse perceptual colormap (yellow-red-black, with higher values darker), log-transformed and globally normalized, and NaN instead of zeros. Interestingly, [9] also features a small B-Matrix explainer using a periodic ring to illustrate an edge-case where all cells have the same node count and create a more vertical B-Matrix. In geological applications, rock fractures organize as networks and exhibit natural variation in their spatial arrangements, [13] used B-Matrix and portrait divergence among other comparative tools to build a novel framework for network-based fracture analysis. Finally, Bagrow and Bollt, authors of the original B-Matrix paper, wrote a follow-up paper in 2019 using the graph invariant properties of B-Matrix to define the Network Portrait Divergence graph similarity metric. Some B-Matrix heatmaps are shown, using the default matplotlib viridis colormap with logarithmic scaling, zeros converted to NaN, and showing the zeroth row [2].

There are two visualization systems that integrate a B-Matrix. First is NetzCope, an all-in-one network explorer that shows a graph's B-Matrix with Matlab's default jet colormap, added to highlight node degree frequency and graph diameter in addition to direct graph views like node-link and adjacency matrix [3]. Second is GraphPrism, which deconstructs B-Matrix into a collection of independent heatmaps for node-specific metrics, which are connectivity (cumulative B-Matrix), transitivity, conductance, density, and neighborhood distance (Jaccard and MeetMin). The GraphPrism system juxtaposes these heatmaps, each with a reverse linear colormap of different hue, against a node-link view of the graph input. GraphPrism only shows modified heatmaps inspired by B-Matrix properties, but not a B-Matrix itself [11].

Regarding the visual explainer format, GitHub features automatic rendering of `README.md`, including image support. As an example, `github.com/tonsky/FiraCode` has an effective visual explainer through a series of infographic-style figures for Fira Code ligatures in action, which in turn inspired the visual explainer format for BMatrix_Explainer as a series of images to scroll through in the repo's landing page.

3 DATA AND TASK ABSTRACTION

The main input dataset is a graph's edgelist, a data abstraction about unique linear connections between items that collectively form a network. The B-Matrix algorithm is a data transformation that takes an edgelist and converts into an array of integers. The B-Matrix is a data abstraction about the structural properties of the input network.

Regarding visual encoding, the graph's edgelist is also the data abstraction behind the node-link idiom, which is an intuitive spatial placement of network elements, empirically found to be helpful in pattern discovery tasks when juxtaposed against the transformed data of B-Matrix heatmaps [11]. The B-Matrix data abstraction can be visually encoded in more ways than the canonical globally-normalized heatmap idiom. Because each row of the B-Matrix adds up to the total number of nodes, row-normalization is a data transformation that can enhance per-row information discovery.

All cells in a B-Matrix are integers that record frequency counts, effectively making each row a histogram [11]. So far in literature these histograms have been encoded by color intensity, but they can also be encoded by height differences in bars, effectively a classic histogram bar chart for each B-Matrix row. Bar charts have high information fidelity in high-contrast monochromatic settings, which is an effective alternative to the canonical heatmap idiom in contexts where color perception is compromised.

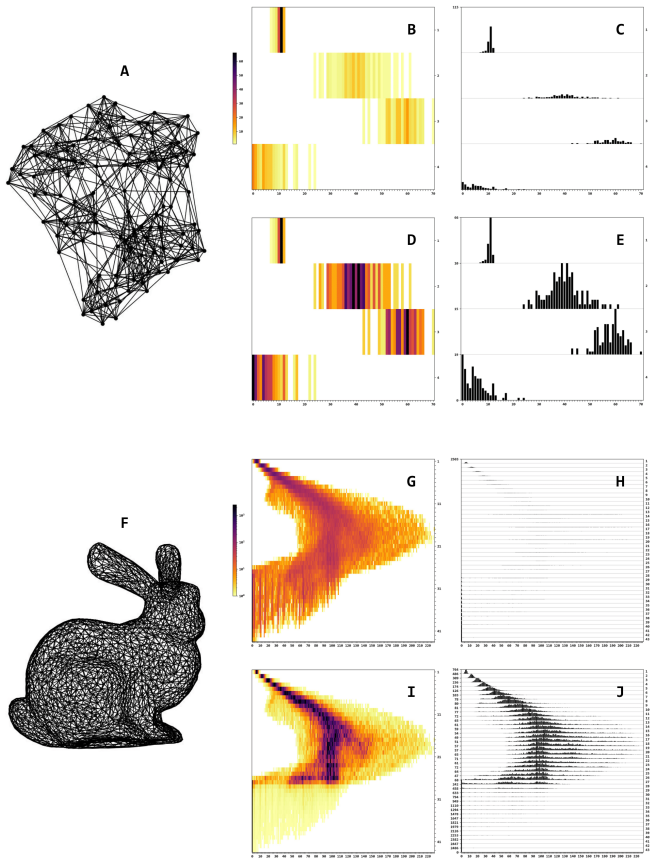

Figure 2. At **A**, node-link view of the Fall 2000 College Football Games network, which has 115 nodes and 613 edges, rendered with **`nx.draw_networkx()`** default force-directed settings, and GML file sourced from personal.umich.edu/~mejn/netdata. From the B-Matrix of **A**, heatmap **B** and histogram **C** are globally normalized while heatmap **D** and histogram **E** have row-normalization. At **F**, node-link view of the Stanford Bunny 3D mesh as an edgelist, which has 2593 nodes and 7048 edges, rendered with plot_mesh_OBJ.py found in the BMatrix_Explainer repo, and OBJ file from github.com/alecjacobson/common-3d-test-models. From the B-Matrix of **F**, heatmap **G** and histogram **H** are globally normalized, while heatmap **I** and histogram **J** have row-normalization.

4 SOLUTION AND IMPLEMENTATION

This paper presents four visualization solutions (Figure 2). First, the canonical B-Matrix heatmap idiom as found in literature, with the distinction of using a reverse perceptual colormap (low values are bright, high values are dark) for high contrast following [9] and [11], all zeros converted to NaN, and removing the zeroth row. Second, the same B-Matrix heatmap idiom but with row-normalization, where each row gets its own colormap scaling. Third, the canonical global-normalized B-Matrix presented in monochromatic bar chart histograms, where each row has a vertical axis range from zero counts to the total number of nodes. Fourth, B-Matrix as row-normalized bar chart histograms, where each row has a unique vertical axis from zero to the per-row maximum node count.

Both the heatmap and histogram idioms described here follow the data tables with two categorical key attributes (algorithm hops and counts), derived directly from the B-Matrix, where each cell is placed in a 2D alignment with these two keys (B-Matrix axes). Per-cell encoding is different, heatmaps use colormaps and histograms use rectilinear bar charts layout, but tasks supported by both are the same, chiefly information discovery, outlier and cluster detection, and information summary.

All four solutions are implemented in Python. Because B-Matrix axes are the same across solutions, they follow the same implementation considerations. To avoid axis cluttering, any axis that has more than 25 levels would have axis tick numbers every modulo-10 steps. This is expanded for each order of magnitude, levels above 250 have ticks every modulo-100 steps, levels above 2500 have ticks every modulo-1000 steps, and so on. Similarly, for the global-normalized colormap view, a log-transformed colormap is activated if the difference between the highest and lowest values in a B-Matrix is 100 or higher. Regarding axis labels, I decided against them because they can be misleading. For example, the horizontal axis can be labeled as "node counts", but this overlaps with each cell which also has node counts as frequency, so a better x-axis label would be "number of nodes encountered at given hop", which is too long. Ultimately, I think that no labels invites new users to engage with the visual explainer more deeply and arrive at the correct interpretation of axis meaning through insight.

With the B-Matrix implementation and visualization scripts complete, my goal was to find an illustrative example for the visual `README.md` of the Matrix_Explainer repo. The input graph for the explainer had to be small, else the algorithm would take too long to run through, but not too small so that key B-Matrix properties were skipped over. To find such a graph, I run my implementation over the 1252 graphs of Graph Atlas corpus as found in the networkx library through `nx.graph_atlas()` [14]. Graph number 1115 met all criteria, and it became the explainer graph (Figure 1).

5 RESULTS

The entire BMatrix_Explainer project is a GitHub repo (`github.com/direditedock/BMatrix_Explainer`) that has three main components. First, the root folder has a `README.md` visual explainer of B-Matrix data abstraction, containing a guide of how the algorithm works, B-Matrix properties, and benchmarking examples. Second, within the scripts folder, `algorithm_BMatrix.py` (adapted from `github.com/bagrow/portraits/blob/master/B_matrix.py`) takes an undirected networkx object and converts it into a numpy array B-Matrix. Third, the scripts `plot_BMatrix_colormap.py` and `plot_BMatrix_histogram.py` in the scripts folder render heatmap and histogram idioms, respectively, out of a numpy array B-Matrix.

Figure 2 shows two large networks each with the four figure exports of BMatrix_Explainer. In Figure 2, heatmap **B** is a reproduction of a published network portrait (Figure 1 in [2]), with the difference of Figure 2 here using reversed inferno instead of default viridis, and removing the zeroth row. To expand on this published figure, here views **C**, **D**, and **E** show the alternative B-Matrix visualizations presented in this paper. Note high the fidelity of **D** and **E** which have row-normalization. Also in Figure 2, note that view **I** for Stanford Bunny the 26th hop is the first row where the zeroth column has nodes accounted for, and from there onwards hops sharply decrease in presence beyond the zeroth column - a structure only revealed by row-normalization (both in colormap and bar chart encoding).

6 DISCUSSION AND FUTURE WORK

Both strengths and weaknesses of this project stem from the reliance on a Python implementation. On one hand, this project leverages the strength of scientific libraries (numpy, networkx, matplotlib) which have a global community of researchers that can pick up the contents within BMatrix_Explainer and continue to build from it within the Python scientific ecosystem. This has technical advantages, one of them being the integration of publication-ready 300dpi PNG figure exports for sharing visualization results. On the flipside, this leaves non-Python users with a learning curve before they can implement BMatrix_Explainer on their own data. For example, `algorithm_BMatrix.py` requires a networkx graph object, and while the repo does contain examples of how to convert OBJ, GML, and CSV data into an undirected graph object, this is not an exhaustive list. To extend BMatrix_Explainer beyond Python users, a browser-based JavaScript implementation could democratize implementation, but would require somewhat strict graph input to be limited to text-only (curated edgelists in CSV or TSV, for example).

Regarding future work, I image a novel system that juxtaposes a node-link or adjacency matrix view of the input graph with its

B-Matrix, with the interactive option to select cells in the row-normalized heatmap B-Matrix view to reveal selected nodes in the node-link view, for example. This can greatly enhance structure discovery and B-Matrix exploration and understanding.

7 REFERENCES

[1] Bagrow, J. P., Bollt, E. M., Skufca, J. D., & Ben-Avraham, D. (2008). Portraits of complex networks. EPL (Europhysics Letters), 81(6), 68004.

[2] Bagrow, J. P., & Bollt, E. M. (2019). An information-theoretic, all-scales approach to comparing networks. Applied Network Science, 4(1), 1-15.

[3] Barber, M. J., Streit, L., & Strogan, O. (2011). NetzCope: a tool for displaying and analyzing complex networks. In Quantum Bio-Informatics IV: From Quantum Information to Bio-Informatics (pp. 437-450).

[4] Czech, W. (2011, May). Graph descriptors from B-matrix representation. In International Workshop on Graph-Based Representations in Pattern Recognition (pp. 12-21). Springer, Berlin, Heidelberg.

[5] Czech, W. (2012). Invariants of distance k-graphs for graph embedding. Pattern Recognition Letters, 33(15), 1968-1979.

[6] Czech, W., & Łazarz, R. (2016, June). A method of analysis and visualization of structured datasets based on centrality information. In the International Conference on Artificial Intelligence and Soft Computing (pp. 429-441). Springer, Cham.

[7] Czech, W., & Yuen, D. A. (2011, August). Efficient graph comparison and visualization using gpu. In 2011 14th IEEE International Conference on Computational Science and Engineering (pp. 561-566). IEEE.

[8] Czech, W., Mielczarek, W., & Dzwinel, W. (2017). Distributed computing of distance‐based graph invariants for analysis and visualization of complex networks. Concurrency and Computation: Practice and Experience, 29(9), e4054.

[9] Gorochowski, T. E., di Bernardo, M., & Grierson, C. S. (2010). Evolving enhanced topologies for the synchronization of dynamical complex networks. Physical Review E, 81(5), 056212.

[10] Gunaratna, C., & Menezes, R. (2011, August). Using Network Science to Understand the Structure of Brazilian Popular Music. In 2011 IEEE 11th International Conference on Computer and Information Technology (pp. 395-402). IEEE.

[11] Kairam, S., MacLean, D., Savva, M., & Heer, J. (2012, May). GraphPrism: compact visualization of network structure. In Proceedings of the international working conference on advanced visual interfaces (pp. 498-505).

[12] Lafhel, M., Cherifi, H., Renoust, B., El Hassouni, M., & Mourchid, Y. (2020, December). Movie script similarity using multilayer network portrait divergence. In the International Conference on Complex Networks and Their Applications (pp. 284-295). Springer, Cham.

[13] Prabhakaran, R., Bertotti, G., Urai, J., & Smeulders, D. (2021). Investigating spatial heterogeneity within fracture networks using hierarchical clustering and graph distance metrics. Solid Earth, 12(10), 2159-2209.

[14] Read, R. C., & Wilson, R. J. (1998). An atlas of graphs (Vol. 21). Oxford: Clarendon Press.

[15] Volchenkov, D., & Blanchard, P. (2007). Comparative Study of Cities as Complex Networks. arXiv preprint arXiv:0709.4447.

8 MILESTONES

A total of 80 hours allocated to 14 weeks is 5.7 hours a week.

Week 01 - Sep 12 to Sep 16 - Started work on recursive scraper and JSON format for data export.

Week 02 - Sep 19 to Sep 23 - Scraper and JSON data export, complete scraping of programming language infoboxes.

Week 03 - Sep 26 to Sep 30 - Tentative plan is to study the network structure of programming languages according to Wikipedia. Project Pitch due September 28 at noon.

Week 04 - Oct 03 to Oct 07 - Data analysis of cardinality and properties of programming language directed graphs. Important to focus on tasks, warning about investing in a dead end project path.

Week 05 - Oct 10 to Oct 14 - No coding or data analysis, only focus on how to pivot towards a design study.

Week 06 - Oct 17 to Oct 21 - Design study based on a user (Wikipedia editor) and a specific task (improving infoboxes). Project Proposal due October 21 at noon.

Week 07 - Oct 24 to Oct 28 - Genericizing the scraper so it works with more topics.

Week 08 - Oct 31 to Nov 04 - Completed extraction of Fortran and Carl Jung dense knowledge networks.

Week 09 - Nov 07 to Nov 11 - Shifting focus to B-Matrix literature.

Week 10 - Nov 14 to Nov 18 - Project Update due November 15 at 3pm. Applying B-Matrix to dense network analysis.

Week 11 - Nov 21 to Nov 25 - BMatrix_Explainer literature and algorithm covered, GitHub repo content started.

Week 12 - Nov 28 to Dec 02 - Alternative B-Matrix visualization encodings and idioms.

Week 13 - Dec 05 to Dec 09 - Code, figures, and documentation for BMatrix_Explainer GitHub repo complete.

Week 14 - Dec 12 to Dec 16 - Final presentation and report.