Redesigning Interactive Educational Modules for Combinatorial Scientific Computing

M. Ali Rostami[1][0000−0001−6154−4464] and H. Martin Bücker[1,2][0000−0002−5210−0789]

[1] Institute for Computer Science, Friedrich Schiller University Jena, 07737 Jena
[2] Michael Stifel Center Jena for Data-driven and Simulation Science, 07737 Jena

Abstract. Combinatorial scientific computing refers to the field of using combinatorial algorithms to solve problems in computational science and data science. Teaching even elementary topics from this area is difficult because it involves bridging the gap between scientific computing and graph theory. Furthermore, it is often necessary to understand not only the methodologies from mathematics and computer science, but also from different scientific domains from which the underlying problems arise. To enrich the learning process in combinatorial scientific computing, we designed and implemented a set of interactive educational modules called EXPLAIN. The central idea behind EXPLAIN is its focus on describing the equivalence of a problem in terms of scientific computing and graph theory. That is, in EXPLAIN, the scientific computing problem and its graph theoretical representation are treated as two sides of the same coin. The process of solving a problem is interactively explored by visualizing transformations on an object from scientific computing, simultaneously, with the corresponding transformations on a suitably defined graph. We describe the redesign of the EXPLAIN software with an emphasis on integrating a domain-specific scripting language and a hierarchical visualization for recursively defined problems.

Keywords: Combinatorial scientific computing · Sparse matrices · Graph models · EXPLAIN · Recursive nested dissection.

1 Teaching Combinatorial Scientific Computing

The term *Combinatorial Scientific Computing* (CSC) refers to an emerging interdisciplinary scientific discipline that is concerned with the development and use of combinatorial techniques for the solution of problems arising from computational and data science. A typical approach in CSC consists of first modeling a problem from scientific computing using graph theory, then solving the combinatorial problem, and finally analyzing the results. If the analysis shows that the results differ substantially from reality, this approach is refined in an cyclic fashion, starting again from the modeling phase. The reader is referred to [13] and [16] for more details on CSC. Compared to the more traditional disciplines of scientific computing and graph theory, the CSC discipline is young. Therefore, there is currently no textbook that supports students attending a course

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

in CSC. Teaching in this area is also complicated by the fact that the fields of scientific computing and graph theory are typically not meaningfully interwoven through any curricular activities. Moreover, the problems addressed in computational and data science frequently have their roots in various different scientific disciplines requiring the students a comparatively high degree of skills in an interdisciplinary way of thinking.

To enrich, promote, and support teaching and learning effectiveness in the CSC domain, we designed, implemented, and used in the classroom a software called EXPLoring Algorithms INteractively (EXPLAIN). This web-based software is a collection of interactive educational modules for different problems arising from scientific computing that have strong connections to graph theory. More precisely, the current version of EXPLAIN includes the following modules: fill-in of Cholesky factorization [14], different variants of sparse matrix compression [5, 4], parallel sparse matrix-vector multiplication [17], and one-way dissection ordering [11]. The primary goal of EXPLAIN is to illustrate the intimate connection between scientific computing and graph theory. The way to accomplish this aim is based on displaying a problem instance in terms of two equivalent representations simultaneously. A representation using an object from scientific computing is displayed immediately adjacent to an equivalent representation using a graph model. EXPLAIN allows students to explore the solution process of a CSC problem interactively. This process corresponds to a sequence of transformations applied to both these representations at the same time.

Although there are a lot of software tools for teaching graph-theoretical topics and graph algorithms, there is currently no other software than EXPLAIN with a focus on the CSC domain. However, we shortly mention some work that is indirectly related to that area. The software packages Gato/CATBox [19] and CABRI-Graph [6] have a focus on animation of graph algorithms. The growing field of data science and network science is a rich source of tools for visualizing, exploring, and analyzing graphs [2, 1]. However, none of these graph visualization tools addresses aspects of scientific computing. On the other hand, software tools that teach scientific computing do not involve any aspects from graph theory; see for instance the interactive Java applets and the NCM software associated with the textbooks [12] and [15], respectively.

The novel contribution of the present article is the redesign of EXPLAIN in an attempt to address the issue of increasing the student engagement. More precisely, in the previous version of EXPLAIN, the algorithm, i.e., the transformation that is applied in each step of the solution process, was prescribed by EXPLAIN. The redesigned version now introduces a domain-specific scripting language enabling the student to develop a new algorithm or modify an existing algorithm. To illustrate the redesign we consider the extension of the module addressing the problem of finding a one-way dissection ordering [11].

The outline of this article is as follows. After sketching the well-known concept of a nested dissection ordering in Section 2, we start the presentation of the redesign in Section 3. Here, we introduce a new simple scripting language used to interactively develop and modify an algorithm within EXPLAIN. As a side

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

effect, this language also allows to add the new feature of algorithm animation. In Section 4, we extend our previous work on the one-way dissection ordering [11] to a recursive approach. Finally, in Section 5, we describe the extension to larger problem sizes and summarize our findings in Section 6.

2 Nested Dissection Ordering

Runtime and storage requirements of direct methods for the solution of systems of linear equations with large sparse coefficient matrices are heavily influenced by the ordering of rows and columns [9]. An ordering with favorable characteristics is a dissection ordering. Let A be a sparse symmetric positive definite coefficient matrix and let P denote a permutation matrix, then finding a dissection ordering is given as the following problem:

Problem 1 (One-Way Dissection Ordering). Given a sparse symmetric positive definite matrix A, find a symmetric permutation

$$P^T AP = \begin{bmatrix} A_1 & 0 & B_1^T \\ 0 & A_2 & B_2^T \\ B_1 & B_2 & C \end{bmatrix} \tag{1}$$

of A such that the size of the last main diagonal (quadratic) block, C, is minimized while the sizes of the two remaining diagonal (quadratic) blocks, A_1 and A_2, are balanced.

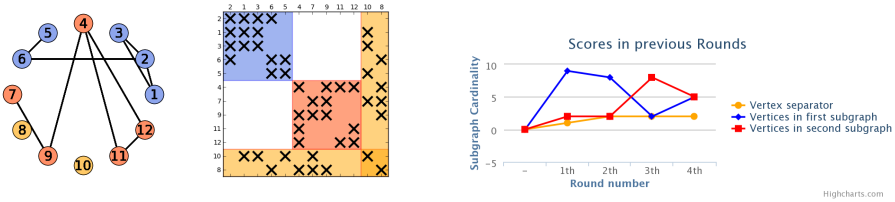
The connection of this scientific computing problem to an equivalent graph problem is illustrated by an interactive educational module in EXPLAIN [11]. Let V be a set of vertices and E be a set of edges and let $G = (V, E)$ denote an undirected graph that is associated with the matrix A in the following sense. The adjacency matrix of G has the same nonzero pattern as A. The graph problem that is equivalent to Problem 1 is then as follows:

Problem 2 (Small Vertex Separator). Given the graph G associated with a sparse matrix A, find a disjoint decomposition of the vertices $V = V_1 \cup V_2 \cup S$ with a vertex separator S such that the size of the vertex separator, $|S|$, is minimized while the sizes of the two remaining components, $|V_1|$ and $|V_2|$, are balanced.

In EXPLAIN, a student can determine a vertex separator S by interactively selecting a sequence of vertices from G. Whenever a vertex is selected the layout of the graph and the layout of the matrix are changed simultaneously as follows: (i) The edges incident to that vertex are eliminated from the graph. (ii) The row and column corresponding to that vertex are moved to the last position in the matrix. (iii) The color of that vertex and of its corresponding row and column are changed to the color denoting membership to S.

In an EXPLAIN module, the process of interactively solving a problem instance, such as determining a one-way dissection ordering, is referred to as a *round*. Thus, in each round of this module, the determined ordering produces

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29



(a) (b)

Fig. 1. (a) Graph and matrix view. (b) Score diagram. Both figures taken from [11].

three sets of vertices, S, V_1 and V_2, that correspond to the matrix blocks C, A_1 and A_2, respectively. The size of these entities characterize the quality of the ordering. To encourage improved participation, EXPLAIN integrates elements of gamification by allowing the student to opt for a sequence of rounds, attempting to improve the quality of an ordering. The history of these rounds is displayed in a *score diagram*. Two excerpts of EXPLAIN's visualization of a problem instance are depicted in Fig. 1. This figure shows the situation when round 4 is completed. The set V_1 and its corresponding block A_1 are printed in blue, while V_2 and A_2 are drawn in red. The color orange is used for the vertex separator S that corresponds to the block C. The score diagram shows that in the first three rounds, the balancing was not achieved. It also indicates that the size of the vertex separator is similar in all rounds.

It is straightforward to apply one-way dissection recursively as follows:

Problem 3 (Nested Dissection Ordering). Given a sparse symmetric positive definite matrix A, find a symmetric permutation $P^T AP$ of A by applying a one-way dissection ordering recursively to the blocks A_1 and A_2 of the matrix (1) such that

- the sum of the sizes of the last diagonal blocks that occur at each of L levels of the recursion is minimized
- while the sizes of all remaining diagonal blocks that occur at these levels of the recursion are balanced.

The corresponding graph problem is then formulated as follows:

Problem 4. Given the graph G associated with a sparse matrix A, find a disjoint decomposition of the vertices V by finding a small vertex separator recursively for the components V_1 and V_2 of the graph such that

- the sum of the sizes of the vertex separators that occur at each of the L levels is minimized
- while the sizes of all the remaining components that occur at these levels of the recursion are balanced.

The module described in [11] illustrates one-way dissection, i.e., the connection between Problems 1 and 2. However, it is not capable of illustrating nested dissection, i.e., the connection between Problems 3 and 4.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

3 Redesigning EXPLAIN

When using the interactive educational modules in classroom students asked how, given a certain CSC problem, they can modify an algorithm that is implemented in the module by their own ideas. The previous implementation of EXPLAIN was not designed to allow for interactive modifications of the algorithm. However, students are better integrated into the learning process if they are themselves more profoundly involved in the algorithm design. In particular, a true motivation for a student is to browse and edit the actual algorithm in a pseudo-code form. That is, on the one hand, we do want to engage the students in implementing an algorithm. On the other hand, we do not want to involve students in the details of any programming language which would prevent them from concentrating on the specific CSC topic proposed by the corresponding module. Our new design therefore consists of three elements: a new scripting language, an interactive online editor, and an algorithm animation.

The new scripting language is designed to be user-friendly and is based on the existing JavaScript code in the previous version of EXPLAIN. In an abstract [18], we briefly discuss a preliminary version of this scripting language and give more detail here. In this scripting language, any algorithm is a single (finite) loop over a set of steps. This assumption corresponds to EXPLAIN's concept of a round which solves a CSC problem by clicking on the vertices in the graph view step by step. Further assumptions are as follows: (i) Program variables are divided into three classes: global predefined, global user-defined, and local user-defined variables. Examples of global predefined variables are those variables that store information of the current matrix and graph. (ii) The algorithm consists of a set of simple mathematical operations on a graph and a matrix as well as some predefined visualization functionalities. Table 1 shows a subset of these operations. (iii) The vertices are unique integer values starting from 0. (iv) Arrays are similar to JavaScript arrays.

Table 1. Subset of operations supported by the new scripting language.

Function Interface	**Operation**
colorVertices(vs, cs)	Color vertices (vs) with colors (cs)
getColors(vs)	Extract colors of vertices (vs)
colorColumn(col, c)	Color column (col) with color (c)
colorRow(row, c)	Color row (row) with color (c)
neighbors(v)	Extract neighbors of vertex (v)
updateMatrix(ord)	Update matrix with ordering (ord)
drawSubMat(c0, c1, r0, r1, c)	Color submatrix (r0:r1,c0:c1) with color (c)

For this language, we provide an online editor that consists of two sections, denoted by "Globals" and "Code" in Fig. 2. These sections are used to define global variables and implement a step of the algorithm, respectively. Given a global array `ordering`, the algorithm specified in the Code section loops over the

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

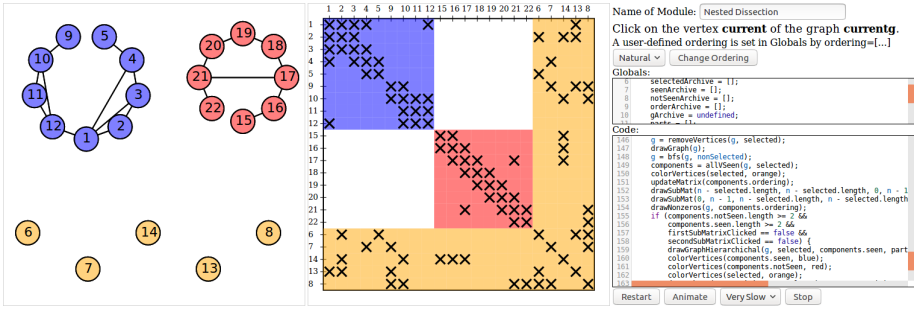


Fig. 2. The online editor is visualized next to the graph and matrix view.

values of this array. The array `ordering` can be either selected from a predefined list of orderings or defined interactively in the Globals section by the student or the teacher.

Finally, EXPLAIN supports an animation of the algorithm. That is, the steps of the algorithm are executed one after another, while the current active statement is highlighted and the matrix as well as the graph view change accordingly. For instance, if the active statement carries out a coloring, the colors of the corresponding objects in the matrix and graph view are updated. The student can choose the speed of the algorithm animation. It is also possible to stop this animation at any time or to restart the animation.

To illustrate the use of the scripting language, consider Fig. 3. Here, a part of the nested dissection algorithm is implemented. More precisely, it shows a single step of the algorithm acting on a graph `g`. From the set of all selected vertices `selected`, all incident edges are removed and the updated graph is drawn in the graph view. A breadth-first search of the graph `g`, implemented in the function `bfs`, followed by the function `allVSeen` returns the components of `g`. Then, the matrix view is updated using the function `drawSubMat` and the nonzeros are drawn in the matrix view. If the determined components are large enough the algorithm switches to a hierarchical view by the function `drawGraphHierarchichal`. Finally, the components are colored in the graph view and the corresponding matrix blocks are also colored with the same color.

4 Hierarchical Layout of Graph and Matrix View

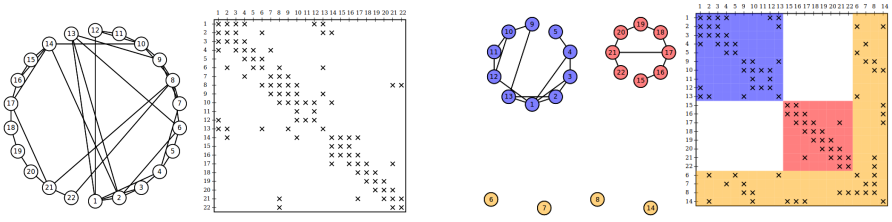
The JavaScript library D3 (Data-Driven Documents) [3] constitutes not only an important building block for the design of the scripting language introduced in the preceding section, but is also useful to implement visualizations of hierarchical structures. Recall from Section 2 that the previous implementation of EXPLAIN is capable of illustrating the connection between the graph and matrix representations of a one-way dissection ordering, cf. Fig. 1, but fails to do so for a nested dissection ordering. To illustrate the recursive application of the dissection idea, the size of the graph/matrix needs to be somewhat larger.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

```
var n = numOfVertices(g);
g = removeEdges(g, selected);
drawGraph(g);
g = bfs(g, nonSelected);
components = allVSeen(g, selected);
colorVertices(selected, orange);
drawSubMat(n - selected.length, n - selected.length, 0, n - 1, orange);
drawSubMat(0, n - 1, n - selected.length, n - selected.length, orange);
updateMatrix(components.order);
drawNonzeros(g, components.order);
var seen = components.seen.length;
var nseen = components.notSeen.length;
if(seen < 2 && nseen < 2) return;
drawGraphHierarchichal(g, selected, components.seen, components.notSeen);
colorVertices(components.seen, blue);
colorVertices(components.notSeen, red);
colorVertices(selected, orange);
drawSubMat(0, seen - 1, 0, seen - 1, blue);
drawSubMat(seen, seen + nseen - 1, seen, seen + nseen - 1, red);
```

Fig. 3. Excerpt of the algorithm implementing a step of the nested dissection ordering.

As an example, consider the graph and the matrix depicted in Fig. 4(a). After interactively choosing the vertices 6, 7, 8 and 14 as the vertex separator S_1, the student will get from EXPLAIN a visualization of a one-way dissection depicted in Fig. 4(b). Here, the rows and columns that correspond to the separator S_1 are numbered last. This one-way dissection produces not only S_1 but also the components V_1 and V_2 shown in blue and red, respectively. However, in contrast to the previous implementation, the student can now interactively choose another two vertex separators S_2 and S_3 associated with V_1 and V_2. In Fig. 5(a), the graph and matrix view are given for the separators $S_1 = \{6, 7, 8, 14\}$, $S_2 = \{1, 13\}$ and $S_3 = \{17, 18\}$. In the graph view, these separators are drawn at the bottom of the graph. The separator S_1 from the first level of recursion is visually divided from the separators S_2 and S_3 of the second recursion level by a dashed line. The remaining four components of the graph are shown in circular layout using different colors at the top of this graph view.

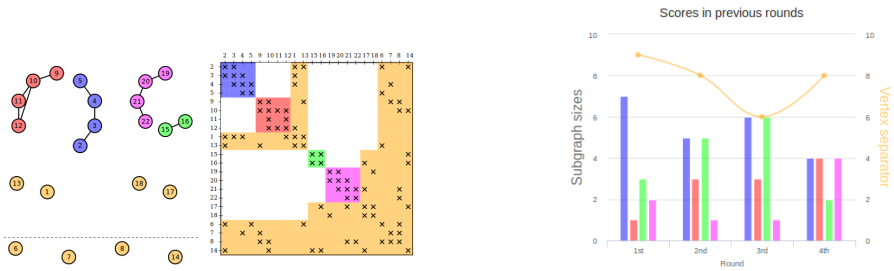


(a) (b)

Fig. 4. (a) Initial graph and matrix view. (b) View after $L = 1$ level of recursion.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

The corresponding score diagram is shown in Fig. 5(b). Here, the size of the vertex separator as well as the sizes of the four remaining components are visualized. The separator size is the sum of the sizes of the vertex separators of all recursion levels. The figure illustrates these quantities for a sequence of four rounds. The line chart shows the history of the size of the vertex separator. For each round, a four-bar chart grouped together presents the four sizes of the remaining components. The colors used in the graph view correspond to the matrix view. The goal is to minimize the size of the vertex separator while balancing the sizes of the remaining components. The hierarchical structure of a nested dissection ordering is also visible in the matrix view.



(a) (b)

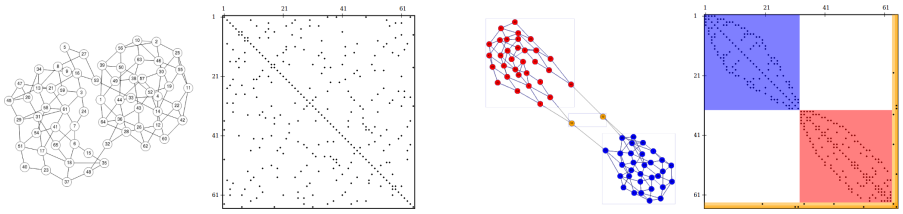
Fig. 5. (a) Graph and matrix view after $L = 2$ levels of recursion. (b) Score diagram with four remaining components, i.e., $L = 2$ recursion levels.

5 Larger Problem Instances

While the basic idea of a specific CSC problem in EXPLAIN is best taught by starting with a small problem instance, the learning process typically also benefits from scaling the problem to larger problem sizes. Increasing the problem size in D3 is straightforward for the matrix view, though some modifications are in order, including hiding the majority of row/column indices on the top and on the left of the matrix as well as replacing the cross symbol denoting a nonzero entry by a small point. However, it is necessary to withdraw from D3 in the graph view. Since visualization in D3 is based on the Scalable Vector Graphics (SVG) technology, the performance is insufficient for displaying larger graphs. The redesign therefore switches to Cytoscape.js [10] which is based on the more efficient canvas element in HTML5 if the problem size is above some threshold. The graph is visualized with the compound layout based on [8].

In Fig. 6(b) the layout of the 65×65 matrix *GD96_c* with 250 nonzeros taken from the SuiteSparse Matrix Collection [7] is shown. As you can see, although the graph is relatively big, the student can find a vertex separator in this layout easily. In Fig. 6(b), the result after selecting a vertex separator is given.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29



(a) (b)

Fig. 6. (a) Initial view of *GD96_c*. (b) View after $L = 1$ level of recursion.

6 Concluding Remarks

When teaching elements of combinatorial scientific computing one of the fundamental concepts is the intimate connection between scientific computing and combinatorial analysis. The EXPLAIN software is a web-based set of interactive educational modules designed to illustrate the underlying correspondences between an object from scientific computing and a suitable graph model. Its goal is to enrich, improve, and accelerate the learning process by displaying these two entities next to each other. When an interactive transformation or an iterative algorithm is applied to a problem instance, the resulting modifications on these two entities are dynamically visualized. The redesign of EXPLAIN includes the integration of a simple scripting language enabling the student to solve a problem from combinatorial scientific computing by developing an algorithm using an online editor.

References

1. Auber, D., et al.: TULIP 5. In: Alhajj, R., Rokne, J. (eds.) Encyclopedia of Social Network Analysis and Mining, pp. 1–28. Springer (2017). https://doi.org/10.1007/978-1-4614-7163-9_315-1
2. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: Third Intern. AAAI Conf. Weblogs and Social Media. pp. 361–362 (2009)
3. Bostock, M., Ogievetsky, V., Heer, J.: D^3: Data-driven documents. IEEE Transactions on Visualization and Computer Graphics **17**(12), 2301–2309 (2011). https://doi.org/10.1109/TVCG.2011.185
4. Bücker, H.M., Rostami, M.A.: Interactively exploring the connection between bidirectional compression and star bicoloring. In: Koziel, S., et al. (eds.) Intern. Conf. Computational Science, ICCS 2015, Reykjavík, Iceland, June 1–3, 2015. Procedia Computer Science, vol. 51, pp. 1917–1926. Elsevier (2015). https://doi.org/10.1016/j.procs.2015.05.456
5. Bücker, H.M., Rostami, M.A., Lülfesmann, M.: An interactive educational module illustrating sparse matrix compression via graph coloring. In: Proc. 16th Intern. Conf. Interactive Collaborative Learning (ICL), Kazan, Russia, September 25–27, 2013. pp. 330–335. IEEE, Piscataway, NJ (2013). https://doi.org/10.1109/ICL.2013.6644591

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29

6. Carbonneaux, Y., Laborde, J.M., Madani, R.M.: Cabri-graph: A tool for research and teaching in graph theory. In: Brandenburg, F.J. (ed.) Graph Drawing. LNCS, vol. 1027, pp. 123–126. Springer, Berlin, Heidelberg (1996). https://doi.org/10.1007/BFb0021796
7. Davis, T.A., Hu, Y.: The University of Florida Sparse Matrix Collection. ACM Transactions on Mathematical Software **38**(1), 1:1–1:25 (2011). https://doi.org/10.1145/2049662.2049663
8. Dogrusoz, U., Giral, E., Cetintas, A., Civril, A., Demir, E.: A layout algorithm for undirected compound graphs. Information Sciences **179**(7), 980–994 (2009). https://doi.org/10.1016/j.ins.2008.11.017
9. Duff, I., Erisman, A., Reid, J.: Direct Methods for Sparse Matrices. Numerical mathematics and scientific computation, Oxford University Press, 2nd edn. (2017)
10. Franz, M., Lopes, C.T., Huck, G., Dong, Y., Sumer, O., Bader, G.D.: Cytoscape.js: a graph theory library for visualisation and analysis. Bioinformatics **32**(2), 309–311 (2016). https://doi.org/10.1093/bioinformatics/btv557
11. H. M. Bücker, M. A. Rostami: Interactively exploring the connection between nested dissection orderings for parallel Cholesky factorization and vertex separators. In: IEEE 28th Intern. Parallel and Distributed Processing Symposium, IPDPS 2014 Workshops, Phoenix, Arizona, USA, May 19–23, 2014. pp. 1122–1129. IEEE Computer Society, Los Alamitos, CA, USA (2014). https://doi.org/10.1109/IPDPSW.2014.125
12. Heath, M.T.: Scientific Computing: An Introductory Survey. No. 80 in Classics in Applied Mathematics, SIAM, revised 2nd edn. (2018)
13. Hendrickson, B., Pothen, A.: Combinatorial scientific computing: The enabling power of discrete algorithms in computational science. In: Daydé, M., et al. (eds.) High Performance Computing for Computational Science–VECPAR 2006: 7th Intern. Conf., Rio de Janeiro, Brazil, June 10–13, 2006. LNCS, vol. 4395, pp. 260–280. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71351-7_21
14. Lülfesmann, M., Leßenich, S.R., Bücker, H.M.: Interactively exploring elimination orderings in symbolic sparse Cholesky factorization. In: Intern. Conf. Computational Science, ICCS 2010. Procedia Computer Science, vol. 1(1), pp. 867–874. Elsevier (2010). https://doi.org/10.1016/j.procs.2010.04.095
15. Moler, C.B.: Numerical Computing with MATLAB. SIAM, Philadelphia, PA, USA (2004)
16. Naumann, U., Schenk, O. (eds.): Combinatorial Scientific Computing. Computational Science Series, Chapman and Hall/CRC, Boca Raton, FL, USA (2012)
17. Rostami, M.A., Bücker, H.M.: An educational module illustrating how sparse matrix-vector multiplication on parallel processors connects to graph partitioning. In: Hunold, S., et al. (eds.) Euro-Par 2015: Parallel Processing Workshops, Vienna, Austria, August 24–28, 2015. LNCS, vol. 9523, pp. 135–146. Springer, Cham, Switzerland (2015). https://doi.org/10.1007/978-3-319-27308-2_12
18. Rostami, M.A., Bücker, H.M.: An online scripting language for teaching combinatorial scientific computing. In: Amme, W., Heinze, T.S. (eds.) Programmiersprachen und Grundlagen der Programmierung, 19. Kolloquium, KPS 2017, Weimar, 25.–27. September, 2015, Tagungsband. pp. 83–85. Jenaer Schriften zur Mathematik und Informatik Math/Inf/02/2017, Friedrich-Schiller-Universität Jena, Jena (2017), http://www.kps2017.uni-jena.de, extended abstract
19. Schliep, A., Hochstättler, W.: Developing Gato and CATBox with Python: Teaching graph algorithms through visualization and experimentation. Multimedia Tools for Communicating Mathematics pp. 291–310 (2002). https://doi.org/10.1007/978-3-642-56240-2_18

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_29