Teaching Algorithmics – Theory and Practice

Petra Scheffler

Nowadays, teaching graph algorithms includes practical questions like suitable data structure libraries, efficiency, stability, modularity and reusability of software as well as the classic theory. The software framework Vinets supports this effort by providing a simple interface for programming exercises in algorithmics. It has been integrated in our business informatics classes successfully.

Graph Algorithms, Software Development, Data Structures, Interactive Teaching

1. Introduction

Efficient algorithms and data structures are important fundamentals of all sophisticated computer applications. For that reason, these topics are included in Computer Science and Business Informatics curricula at almost all universities. Indeed, the classic textbooks present algorithms mainly as pure mathematical results [3]. They leave the challenges of their implementation to the readers.

On the other hand, algorithm engineering is a very popular and dynamic research topic. Studies in this area are directed at developing efficient programs that are suited for real machines and real world problem instances [1, 4]. Think, for instance, about modern navigation systems that need fast realizations of shortest path algorithms for graphs with Millions of nodes on a hand-held computer with low memory resources. The research includes the design, analysis, implementation and evaluation of different algorithms by experiments.

The new challenges from practice must be addressed in teaching graph algorithms, too. Questions like suitable data structures, efficiency, stability, modularity and reusability of software are very important and closely related to algorithms. At the University of Applied Sciences Stralsund, a corresponding modern Algorithmics course is offered for several years [13]. It considers the mentioned topics in their interrelations. The course includes theoretical lectures on graph algorithms and practical exercises on their realization in the computer laboratory. Here, the tool VinetS [17] has been used successfully. We have designed and implemented the framework VinetS in Java for the visualization of network structures at our university. The design of this software is sketched in Section 3. In addition to the standard exercises, the best students are involved in projects for further development of the tool itself. Some of their results are presented in Section 4.

2. Graph Algorithms in the Business Informatics Curriculum at FH Stralsund

The Business Informatics Bachelor and Master programs at the University of Applied Sciences Stralsund are accredited according to the guidelines of the German Computer Science Society for computer science studies with one application area (here: Business Administration). That means that classic computer science amounts to one third of the courses taught, supplemented by economics and intrinsic business informatics courses.

As anywhere else, Discrete Mathematics is one of the modules taught during the first year. This module includes also the first acquaintance with graph theoretic concepts and standard algorithms like that of Dijkstra for the single source shortest paths problem. We follow the classic approach here like it is presented in usual text books as [3] and their German counterparts. Various applications of graphs for modeling practical economic problems are addressed, too.

Next, our students learn about data structures and algorithms during their second year. Here, graphs occur for instance in the form of linked lists or binary search trees. At that time, the students have already well-founded knowledge and experience in Java
programming. So, it is possible to join the lectures on data structures and algorithms by practical exercises in the computer laboratory. My lectures follow more or less the book [8] of Goodrich and Tamassia. Their course gives beside theory real Java implementations and also highlights software engineering questions. Moreover, my own main focus is on the Java Collection Framework in the practical classes. In my opinion, this library is a representative example of well designed software. Students should not only learn which interfaces and classes it contains and how to use them. It is also worth to have a look into the source code. One can learn a lot about good reusable object oriented code. Students find applied there many software design patterns like decorator, factory or template method and others that they know theoretically from their software engineering course [6]. Also, they see how powerful generics in Java are and how simple it becomes to ensure type safety. Most of nowadays students will never have to implement new data structure classes like lists and neither they have to code sorting algorithms in their job – so it is time to change the curriculum anywhere in this direction – enabling students to understand and use efficiently existing software libraries. Also, it is important to connect the knowledge of different teaching modules with each other for getting enduring skills.

The module Algorithmics [13] is part of the elective Application System Development in our curriculum. It is based on the basic knowledge explained above. Its aim is to obtain a deeper understanding of graph algorithms, their design and analysis and their implementation and to achieve the ability to apply graphs for practical problems. In addition to traditional lectures and seminars, we have adopted modern teaching methods in two ways.

First, every student has to implement some of the discussed algorithms. This work is supported by a software tool called VinetS that we have developed at our university. It includes careful designed graph data structures that facilitate the typical algorithm steps used in text books. The tool has a simple interface to integrate new algorithms. The student can generate graph examples interactively in a graphical user interface and she can run her own algorithm on it – being immediately faced with the results. Students are motivated by this activity with positive or negative reply to their effort.

![Figure 1: A student algorithm implementation executed in VinetS.](image)

In Figure 1, the result of a call to a DFS implementation is shown as an example. The algorithm determines the two-connected components (blocks) of an undirected graph. In the result, the edges of the DFS-tree and the articulation points are highlighted by another
color. The nodes of all computed blocks are listed in the communication window at the bottom. This way, a student gets direct visual and textual information about the effects of her algorithm. One can check the correctness easily. Often, students understand important details and special cases of an algorithm only at this time, by their own practical experience. They are inspired to test their program with different graph examples.

We describe the applied tool VinetS in more detail in the next section of this paper. Our teaching approach can be recommended to other universities. It turned out that implementing and testing graph algorithms not only trains the software engineering and programming skills of our students, but also supports their understanding of the algorithmic ideas.

Second, our students are encouraged to read recent scientific papers from journals or conferences (in English) in order to see the algorithm design process at work and its importance for business. Usually, I choose several papers on one popular application area – one for every student to read, understand and explain to others. The discussion becomes very lively, when the themes of the talks are interesting and close to each other. For example, in the last year we studied algorithms for modern navigation systems. The problem there is to find shortest paths in large graphs with short answer times using only restricted resources of memory and computing power. For this, we used the papers presented to the implementation challenge at Rutgers University, see [4]. Students found it ambitious to read these sophisticated papers that are not yet so well polished like textbooks. On the other hand, the students liked to learn how algorithms are used profitably in practice. Theoretical analysis proved to be useful and became more interesting to students by this way.

3. Our Software Tool VinetS

At the University of Applied Sciences Stralsund, we have designed and implemented the generic Java tool VinetS for the visualization of network structures. Its application area is not restricted to teaching. The software is useful wherever structure and relations of real world entities are modeled, presented and studied. Circuit diagrams, maintenance plans for piping in water supply and distribution or in chemical industry, utility network plans in facility management, but also flow charts and function diagrams in control technology, and work flow diagrams are classical applications of schematic network drawings. More recent applications include hypertext, knowledge and content management, visual data mining, and social networks. Diagrams of various types are used in software development for design and documentation purposes.

VinetS is intended as a framework for the development of several application specific software tools. It contains a graphical user interface to edit graphs and hierarchical hyper graphs as well as several layout algorithms. The design of the software is modular. Simple interfaces allow for modifications of the visual presentation style and for algorithmic extensions. Network data are stored in the extensible XML-format GraphML, see [9].

On the one hand, VinetS is an application to manipulate and to draw graphs and networks. The implementation in Java makes it platform independent. A Swing graphical user interface allows the user to draw nodes and edges interactively, to move parts of the graph and to perform other common graphic interactions.

On the other hand, VinetS is a library of classes and interfaces useful for graph analysis and representation. Its building blocks can serve as starting point for the development of specialized schema drawing tools. The kit delivers a set of data structures that are designed and tested carefully. They implement simple interfaces; and thus are extendable and easy to use. The system is divided into different modules in order to enforce flexibility.

The core of the application is responsible for the treatment of network data. Graphs are stored in three layers: The structure and geometry packages administrate the data that
are usually processed by graph algorithms – the relations of edges and nodes and their position. In addition, the *graphics* package adds functionality for their visual representation on the screen and the rendering process.

The *algorithm* package contains the library of classes for analyzing graphs – it is a growing reservoir of implemented graph theoretic knowledge. Algorithms for automatic placement and routing are contained in a special *layout* package.

The *control* package is responsible for the program flow and the interaction of all modules. This includes tool configuration management and event handling.

The gateway of the application to its environment is implemented in two packages. First, the *gui* package contains the graph editor and user interaction. Second, input and output from and to files are delivered with the *io* package. In VinetS, networks can be stored in the graphics formats *.jpg and *.png. In addition, all network data can be stored in GraphML format. This is an extensible XML language for graphs and hyper graphs. It has been developed by the graph drawing community; see [9] for a description. The modular architecture of VinetS allows for easy addition of new graph readers or writers for other formats. Application specific information can easily be added to the graph.

Graphs can be stored with or without layout information. When the graph data file contains only the structure information – determined from relationships of the application area, the VinetS-reader computes random coordinates for all nodes. Of course, the first drawing becomes unreadable in this case. Figure 2 shows such an example. Now, the tool offers some layout algorithms that automatically improve the picture, see Section 4.

![Figure 2: A large planar graph with random layout in VinetS.](image)

The system VinetS is well suited for teaching Algorithmics. Students can integrate their programs into the algorithm package by adding one line of text into the configuration file. The structure package contains the interfaces Node, Edge and Graph for usual graph objects. Their protocol assures methods for all basic operations of algorithms as they are described in graph theory textbooks. For example, one can simply iterate over the neighbors of a node or over all its outgoing edges; and one can add or delete a node or an edge of the graph. Moreover, any graph element (node, edge or graph) is labeled with a hash map, so that it can store additional information of any kind. The labels can be used for the problem input (e.g. edge weight, capacity or a mark as start node) and also for
notes during the execution of an algorithm. This makes it easy to translate an algorithm from pseudo code into a Java method.

When a student has implemented a graph algorithm, the VinetS framework helps her to make it visible in action. The extension mechanism of the tool enables the attachment of a user algorithm. The only precondition is that the attached user class should implement the Algorithm interface of VinetS. This interface requires an execute method acting on a graph with labels. Now, the user of the tool can call this method via an automatically generated button or menu item. The algorithm is executed then on the graph shown currently in the editor window. Afterwards, the manipulated graph and any textual output are displayed. A configuration mechanism is available for more involved algorithms. If the programmer has specified a set of options for his algorithm, the option values are asked for interactively by a generated dialog. These options can be weight conditions, start and end vertices or even a list of possible subgraphs to choose from.

A free demo version of VinetS can be downloaded from our web site [17] for evaluation. Furthermore, a collection of sample exercises and their solutions are available upon request.

4. Graph Drawing Algorithms

The initial development of the VinetS framework was supported by a research project in 2002 / 2003. Since then, most of the enhancements are the result of students work. Several projects and final theses have contributed to the layout module. We present some of them in this section.

The VinetS software is able to handle directed, undirected, or mixed simple graphs, as well as hierarchical hyper graphs with ports and variable block sizes. Several graph layout algorithms are implemented. This includes some general purpose algorithms like random, circular or grid layout and some algorithms for special graph classes. Hyper graphs are drawn in orthogonal layout style while simple graphs use straight-line edges. Some algorithms respect the user-defined layout improving it only a little (like grid or level alignment), others compute the graph geometry completely from structure ignoring given input positions.

![VinetS Graph Editor](image)

Figure 3: Example 2 drawn without crossings by Kant's algorithm.
Up to now, a lot of effort has been made to draw planar graphs without crossings; see [14] for a more detailed description. In Figures 2 and 3, the input and the result of one of our implementations are shown. The graph has 250 nodes. First, its structure is read from file and the nodes are laid out in random positions (Figure 2). Then, the plane drawing is computed very quickly (Figure 3).

The planarization algorithm is performed in two Phases. First, the structure of the graph is analyzed. If the input is identified as a planar graph, a so-called topological embedding for it is determined. This arranges the nodes to faces, i.e. cycles of the graph that surround empty regions in a drawing without edge crossings. For this recognition phase, we use a linear time implementation of the algorithm of Boyer [2, 19].

The geometry is computed in the second phase of the algorithm. In Figure 3, all nodes have got positions on an integer grid by our implementation of the algorithm of Kant [11]. It works well only for 3-connected graphs. We have implemented also some other algorithms for the geometric phase of planar graph layout, among them some own heuristics. Another algorithm taken from the literature is one that draws all nodes of a planar graph without crossings on a minimal number of concentric circles [5, 18]. Figure 4 shows a simple drawing generated by our implementation of this algorithm.

![Figure 4: A radial drawing of a planar graph](image)

In practical applications, simple graphs are not always sufficient for modeling. Often, networks are constructed hierarchically from smaller ones as building blocks. Nodes are not points anymore, but they have determined sizes. Moreover, edges are not always connections between only two endpoints. For instance in electronics, a link connects the output from one block to the input of several other elements typically. The entry point of a connection to a block is fixed now and it cannot be computed by a layout algorithm anymore. The tool VinetS can handle all these conditions in its hierarchic hyper graph modus. Figure 5 shows a small example. The layout is done by heuristics in this case, links are routed orthogonally. For more details, we refer the reader to [7].

![Figure 5: A hierarchical hyper graph](image)
Another important application of graph methods is the design of electrical networks. Usually, the models are series-parallel directed graphs here. Networks are composed from smaller pieces such that there is much symmetry. In a nice drawing of a series-parallel digraph, equivalent subgraphs should be visible apparently. This can be assured by a layout algorithm that computes certain automorphisms of the input graph and then uses them for constructing a symmetric drawing [10, 12]. Figure 6 shows an example of a symmetric drawing for a series-parallel digraph that was generated by the VinetS tool automatically.

![Figure 6: A symmetric drawing generated for a series-parallel digraph](image)

In closing, VinetS is an extendible framework useful for teaching algorithms as well as for analyzing and drawing networks in different application areas.

5. References


6. E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley 1995.
7. B. Goetze, M. Pleßow, P. Scheffler: 
   *Level-Layout für die Generierung grafischer Dokumentationen in der Leittechnik.*


9. GraphML Team: *The GraphML File Format.*
   http://graphml.graphdrawing.org

10. S. Hong, P. Eades, S. Lee: *Drawing Series Parallel Digraphs Symmetrically.*

11. G. Kant: *Drawing planar graphs using the canonical ordering.*

12. J. Lewandowski:


15. P. Scheffler: *Software zur Visualisierung vernetzter Strukturen.*
    Landestecnologieanzeiger Mecklenburg-Vorpommern 2 (2005), S. 8–9.


19. A.-M. Törsel:

Prof. Dr. Petra Scheffler
FH Stralsund, FB Wirtschaft
Zur Schwedenschanze 15
D-18435 Stralsund
Germany

Petra.Scheffler@fh-stralsund.de