AGD – A Library of Algorithms for Graph Drawing

Michael Jünger¹, Gunnar W. Klau², Petra Mutzel³, and René Weiskircher³

¹ University of Cologne, Department of Computer Science, Pohligstraße 1, D-50969 Köln, Germany
² Konrad–Zuse–Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin, Germany
³ Vienna University of Technology, Institute of Computer Graphics and Algorithms, Favoritenstraße 9–11, A-1040 Wien, Austria

1 Introduction

The development of the AGD software, an object-oriented C++ class library of Algorithms for Graph Drawing, has started in 1996. AGD is a general purpose graph drawing tool suited for beginners as well as for advanced users. It contains a variety of layout algorithms leading to different layout styles. However, the primary goal of development has been to provide users with a toolbox for creating their own implementations of graph drawing algorithms according to their specific needs. Since, in many cases, users want the layouts to satisfy application-specific requirements that are not foreseen in generic graph drawing methods. AGD is designed in such a way that it is easy to add user-specific changes to the layout algorithms.

Another important goal of AGD was to bridge the gap between theory and practice in the area of graph drawing. E.g., for drawing general graphs, Battini et al. [3,2] suggested a method based on planarization which often leads to good drawings for many applications. However, until 1996, no publicly available software layout tool used the planarization method. The reason for this was twofold: On the one hand, a lot of expertise is necessary concerning planarity testing algorithms, combinatorial embeddings, planar graph drawing algorithms, and (often \(NP\)-hard) combinatorial optimization problems. On the other hand, great effort is needed to implement all necessary algorithms and data structures, since the planarization method consists of various phases that require complex algorithms.

Recently, major improvements have been made concerning the use of the planarization method in practice (e.g., [32,19,26–28]). Today, there exist some (publically available) software libraries using the planarization method successfully for practical graph layout [1,22,23]. In AGD, the planarization method is implemented in a modular form, so that it is easy to experiment with different approaches to the various subtasks of the whole approach. This enables experimental comparisons between various algorithms in order to study and understand their impact on the final drawing. Not only in graph
drawing, the empirical study of combinatorial algorithms is getting increasing
attention.

Also the Sugiyama-style method for drawing graphs with preferred direc­
tion is rather a methodological frame than a fixed algorithm. For each phase
layer assignment, crossing minimization, and coordinate assignment, a vari­
ety of possible implementations exists. AGD allows users to simply switch
among a variety of implementations, and gives software programmers the
possibility to introduce new algorithms.

Another reason for building AGD was our intention to show how mathe­
matical methods can help to produce good layouts. Many of the optimization
problems in graph drawing are $\mathcal{NP}$-hard. However, this does not mean
that it is impossible to solve them in practice. AGD shows that problem instances
can often be solved to provable optimality within short computation time by
using polyhedral combinatorics and branch-and-cut algorithms.

2 Applications

We can distinguish two groups of users of AGD: The first group uses only
the algorithms that are already implemented without making any changes or
extensions while the second group writes new modules in order to change the
behavior of the drawing algorithms already contained in AGD. The first group
only needs the executables of the demo programs. The $\text{agd.demo}$ executable
contains only algorithms working without additional software packages while
$\text{agd.opt.demo}$ contains the exact optimization algorithms that only work on
systems where ABACUS [35] and CPLEX are installed.

Both demo programs have graphical user interfaces that are based on
the class $\text{graph.win}$ of the LEDA-library [43]. This class already contains
methods for loading and storing graphs in different formats and also for
creating and manipulating them. The demo programs extend the menus of
$\text{graph.win}$ by the methods implemented in AGD for generating and drawing
graphs. Since $\text{graph.win}$ allows drawings to be exported in PostScript format,
drawings generated using AGD algorithms can easily be used as illustrations
in documents.

The demos thus constitute a very general graph drawing environment.
Data we collected from people who downloaded the demos show a diverse
range of applications. In the field of biology, AGD is used to visualize meta­
bolic networks, protein interactions, gene regulatory networks, and plant dis­
tributions. In social science, uses include drawing genealogical trees, collective
labor agreements and net structures in cognitive models. Other applications
include the visualization of neural networks, electrical grids, flowcharts and
the dependency graph of university courses.

The disadvantage of the generality of the demos is that they are not
suitable for applications where the drawings have to meet a very restrictive
set of requirements. Therefore, the second group of users writes new modules