J Math Chem (2009) 45:471–477
DOI 10.1007/s10910-008-9420-5

ORIGINAL PAPER

GrInvIn in a nutshell

Adriaan Peeters · Kris Coolsaet · Gunnar Brinkmann · Nicolas Van Cleemput · Veerle Fack

Published online: 7 August 2008

Abstract GrInvIn (**Gr**aph **Inv**ariant **In**vestigator) is a software framework for teaching graph theory and for research in graph theory and graph theoretic chemistry. It enables users to construct graphs, compute invariants (e.g. topological indices in chemistry) and investigate relations between these concepts. The design of GrInvIn emphasizes easy usage and makes use of software engineering techniques that enable the user to easily extend the system (e.g. by adding new topological indices to investigate).

Keywords Graph · Topological index · Graph invariant · Conjecture · Software framework

1 Introduction

Various programs to support research in graph theory have been developed and successfully used, such as AGX/AGX2 [1], Cabri-graph [2], Graffiti [3], Graffiti.pc

A. Peeters · K. Coolsaet · G. Brinkmann (✉) · N. Van Cleemput · V. Fack
Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281 S9, 9000 Ghent, Belgium
e-mail: Gunnar.Brinkmann@UGent.be

A. Peeters
e-mail: Adriaan.Peeters@UGent.be

K. Coolsaet
e-mail: Kris.Coolsaet@UGent.be

N. Van Cleemput
e-mail: Nicolas.VanCleemput@UGent.be

V. Fack
e-mail: Veerle.Fack@UGent.be

[4], GRAPH [5], GraPHedron [6], LINK [7], and newGRAPH [8]. Some of them emphasize the manipulation of graphs and computation of invariants, others focus on (graph) conjecturing.

LINK was the first graph theory application we know of that was designed with flexibility as the highest priority. This was implemented as a scripting interface in the high-level programming language Scheme. Unfortunately it turned out to be hard to maintain and not very portable.

As to the goal of GrInvIn we were most influenced by Graffiti.pc which was developed by Ermelinda Delavina. It was created for research in graph theory as well as for teaching graph theory by means of graph conjecturing.

The emergence of new software engineering techniques made it possible to design software that is easy to extend, maintain and install. Thus extensibility, re-usability and portability were key requirements in the development of GrInvIn.

In the remainder of this text we will use the terms *topological index* and *graph invariant* interchangeably. They describe the same thing, but the concept *topological index* is more often used by chemists while mathematicians usually use *graph invariant*.

2 GrInvIn in a nutshell

GrInvIn is a software for the investigation of graphs, their topological indices and conjectures about (classes of) graphs. GrInvIn provides several ways to input graphs and can compute invariants and make conjectures based on the set of graphs under investigation.

Although GrInvIn does already contain a basic conjecturing engine, this is not our main purpose and we hope that people specialized in conjecturing will also use this framework to plug in their advanced routines and thereby make them available to the public.

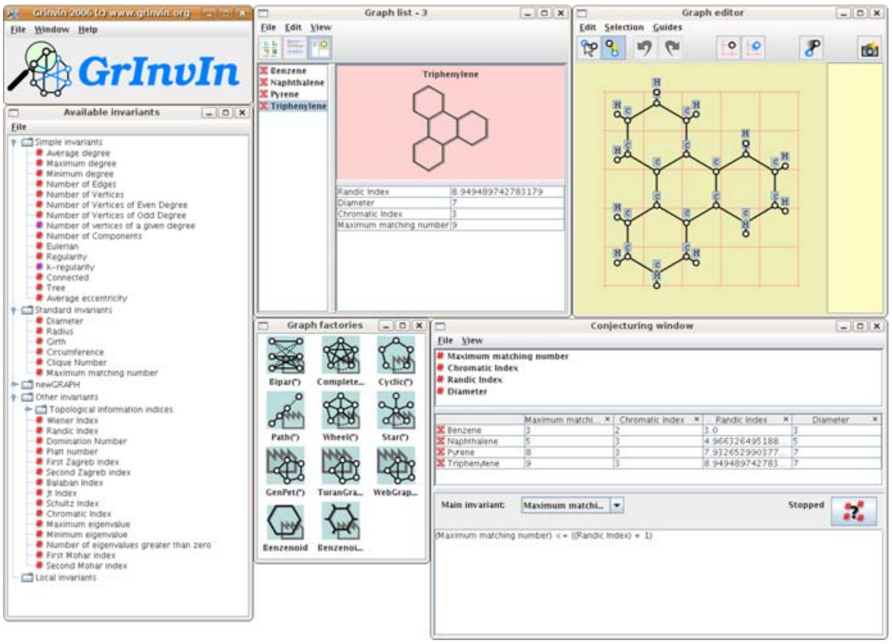
The GrInvIn framework provides the core functionality needed to implement an application for graph theory in general. It includes basic functionality to work with graphs, invariants (topological indices) and conjectures. In addition to data structures and interfaces for these concepts, the framework also has a basic graph editor and an intuitive graphical user interface.

In order to guarantee portability, the interface and most of the subroutines are written in the highly portable programming language Java. Some parts that are performance critical and interact less with the operating system are written in C.

2.1 User interface

The GrInvIn user interface (Fig. 1) makes use of the *drag and drop* concept to work with graphs and topological indices. All GrInvIn objects are represented as icons. By dragging and dropping those objects, the user can create lists of graphs and invariants. This user interface works in a way similar to that of the file manager found in most recent operating systems.

To allow the user to work in their own native language, the entire user interface is internationalized. Translating the framework only involves providing a few language



Randic Index	8.949489742783179
Diameter	7
Chromatic Index	3
Maximum matching number	9

	Maximum matchi...	Chromatic Index	Randic Index	Diameter
Benzene	3	2	3.0	3
Naphthalene	5	3	4.966326495188...	5
Pyrene	8	3	7.932652990377...	7
Triphenylene	9	3	8.949489742783...	7

```
(Maximum matching number) <= ((Randic Index) + 1)
```

Fig. 1 Screenshot of the GrInvIn user interface

files, without any changes in the application source code. So far GrInvIn comes with language files in English and Dutch.

2.2 Graphs

The graphs used in the framework can come from several sources: they can be drawn in the editor, created using a *graph factory* or imported from an external file. In the next version it will also be possible to read them online from graph generation programs that produce huge lists of graphs. Graph factories are included for the most prominent classes of graphs, such as complete graphs, complete bipartite graphs, cycles, paths, Turán graphs, generalized Petersen graphs, wheels, stars, benzenoids, etc…The graphs currently supported are simple and undirected. Both vertices and edges can be labelled or annotated with complex information. Other representations such as directed and/or multi-graphs can be added easily.

2.3 Topological indices

GrInvIn already includes several basic graph invariants and topological indices. They range from trivial ones such as the number of vertices to non-trivial ones such as the Wiener index or the chromatic number. This library will grow as the framework is being extended and new algorithms are contributed using the plug-in mechanism as described in Sect. 2.5.

Among others, the following invariants/indices come with the program. The list is permanently extended.

Chemical	**Mathematical**
• Balaban index	• Chromatic index
• J_t index	• Circumference
• Mohar indices 1 & 2	• Clique number
• Platt number	• Diameter
• Randic index	• Domination number
• Schultz index	• Girth
• Wiener index	• Maximum matching (Kekulé)
• Zagreb indices 1 & 2	• Minimum/maximum eigenvalue
• Several topological information indices	• Number of eigenvalues greater than zero
	• Radius

2.4 Conjectures

For the conjecturing engine currently contained in GrInvIn one chooses a fixed invariant I and a set S of invariants that may be used for conjectures.

GrInvIn then computes expressions of the form $I \leq f(S)$, with $f(S)$ being a function of one or more invariants in S, that are at least true for all graphs in the list. The function f is computed by minimizing a cost function that is based on the structural complexity of the function and $\sum_G (f(S))(G) - I(G)$ for all graphs G in the list. Due to the contribution of the structural complexity of the function, $f(S)$ will normally use only few of the invariants in S.

An example e.g. useful in teaching (see later for how conjectures can be used in teaching) would be:

- The chosen invariant I is the size of a maximum matching.
- The set S of invariants that may be used for S contains e.g. the chromatic index, the Randic Index, the diameter, etc.
- The list of graphs under investigation is benzene, naphthalene, pyrene, and triphenylene
- The computed expression is then e.g. max matching number $\leq$ Randic Index $+1$.

This conjecture is wrong, but it is a good example for use in teaching and we will come back to it in part 2.7.

Other conjecturing engines will in general return different output. We hope to have more elaborate conjecturing engines available soon.

2.5 Extensibility

Since extensibility by the user is one of the key design goals of GrInvIn, adding an algorithm for a topological index is very straightforward. It comes down to writing the algorithm code and providing some basic documentation about the invariant. When reusing existing programs, some simple wrapper code has to be written.

To add a new algorithm for the computation of a topological index, one has to extend the Java class `AbstractInvariantComputer` and implement the `compute()` method. The algorithm can access the graph data as an adjacency matrix, adjacency list, list of eigenvalues or a list of vertex and edge objects.

Existing code can be used by calling C, Pascal or other routines from within the `compute()` method.

The plug-in functionality of GrInvIn also makes it trivial to add new input and output formats, new lists of graphs, and later also new generators for graph input.

2.6 Testing and documentation

When software is being used in a research environment, special care is necessary to ensure correctness of the results. Software engineering methodology addresses this by introducing unit tests: small sections of the code (*units*) are individually tested. Those tests are usually written before the code to be tested is written. This allows verification of the output.

The unit tests are run repeatedly during the development cycle: usually after every change to the modules that are tested, which makes it possible to detect changed or unexpected behaviour of the software in an early stage.

Each of the graph invariants and topological indices that come with the program has been tested against an independent implementation—sometimes using different (e.g. very straightforward) algorithms and also other programming languages—to decrease the probability of errors. The online documentation contains information about the amount of testing done.

Since complete and correct documentation is important for research as well as education (for example to know how a certain topological index is defined), the GrInvIn framework includes a context sensitive help system that provides documentation on the built-in topological indices, graph factories, basic graph theory concepts and on the GrInvIn user interface. Example graphs found in the documentation can be dragged straight from the help pages into the GrInvIn application.

The documentation is provided in several languages and can be translated easily. Since GrInvIn is also intended to be used by students it is important for them to have the definitions available in their native language.

2.7 Using conjectures in teaching

Especially chemistry students who often prefer the practical side of chemistry need a motivating and interesting method to teach them graph theory.

The GrInvIn software can be used in an educational environment to teach students what topological indices are and how they are related. In fact the development of GrInvIn was motivated by very promising results of applying Graffiti and Graffiti.pc in teaching graph theory in Houston [9,10] and later also in Bielefeld. The fact that already from the start the student can have a research like experience motivated a lot of the students who took a Graffiti or Graffiti.pc supported beginners course to afterwards continue their studies in graph theory and often even write their theses in this field.

The principle is to use GrInvIn (like Graffiti.pc before) to compute a conjecture that the student has to refute (by finding a counter example) or prove. The conjectures are computed based on the graphs and topological indices thus far selected by the student.

The student usually starts by selecting some fixed invariant he/she wants to work on and a graph list normally consisting of one or very few basic graphs. The set of invariants for the right hand side of the inequality normally consists of a very large set of invariants the student doesn't know in the beginning. Based on this data, GrInvIn comes up with a conjecture that the student can often refute easily by giving a counterexample. To be able to do so, he first has to understand the new invariants on the right hand side of the inequality. But experience shows that students tend to identify themselves with **their** invariant (which is individual for each student) and the fact that the new invariant is being combined with his/her invariant has a motivating effect on the students.

In case of a wrong conjecture, the student has to find the smallest counter example and prove that it is smallest possible in order to gain experience in proving graph theoretical results right from the start. Then this counterexample is added to the list of graphs and another conjecture is computed.

In case of a true conjecture it is the student's task to determine whether the difference between the right and the left hand side can be arbitrarily large. In this case an example graph forcing the engine to choose another conjecture can be added to the list, otherwise one of the invariants used for the right hand side are deleted from the list of usable invariants. In any case a new conjecture will be obtained and the process of refuting or proving can continue.

In the example from part 2.4 "max matching number $\leq$ Randic Index $+1$" the student would *play around* with small examples and find the conjecture true. But he would surely also see that the Randic number in these examples is in fact never smaller than half the number of vertices minus one. Maybe—at least with a little help—he would then try to proof that (e.g. by induction on the number of faces) and see how adding a face changes the Randic Index, and thereby finding a way to construct a counterexample and—even more important—understand the behaviour of the Randic Index for benzenoids much better.

So the student begins with very easy—in general easily refutable —conjectures and together with the list growing and the student improving his/her knowledge, the conjectures become more difficult to refute and more challenging. This way the difficulty of the problems grows together with the knowledge and abilities of the student.

2.8 Planned additions

The framework is still in development. The first public release went online in January 2007 and is available from http://www.grinvin.org/.

If the user wants to examine a certain class of graphs, in the first version he still has to make sure that just graphs of this class are added to the list. The following version will already have the possibility to describe the class of graphs and make the program check that every graph chosen for the list really belongs to that class or filter input graphs for those belonging to the class.

On the long term we plan to include automated conjecture refutation. This will be done once by using files with graphs that are known for some especially difficult structures (e.g. *Snarks*, see [11]) and online generators for classes of graphs that can more efficiently be generated, like e.g. *all graphs* [12] or—maybe more interesting to chemists—benzenoids [13]. In addition to simply testing conjectures on lists, we also hope to be able to include more sophisticated techniques like the ones used for AGX/AGX2. As for conjecturing we hope in this context for cooperation with other groups. Automatic conjecture refutation can be used for research in order to automatically advance to more challenging conjectures (the computer takes the role of the student in the beginning) and in teaching in order to help the teacher check minimality of a counterexample.

References

1. G. Caporossi, P. Hansen, Discrete Math. **212**(1–2), 29–44 (2000)
2. Y. Carbonneaux, J.-M. Laborde, R.M. Madani, in: *Graph Drawing*, ed. by F.-J. Brandenburg (Springer, Berlin, 1996) pp. 123–126
3. E. DeLaVina, in: *Graphs and Discovery*, ed. by S. Fajtlowicz, P.W. Fowler, P. Hansen, M.F. Janowitz, F.S. Roberts (American Mathematical Society, Rhode Island, 2005) pp. 81–118
4. E. DeLaVina, in: *Graphs and Discovery*, ed. by S. Fajtlowicz, P.W. Fowler, P. Hansen, M.F. Janowitz, F.S. Roberts (American Mathematical Society, Rhode Island, 2005) pp. 71–79
5. D. Cvetković, S. Simić, in: *Graphs and Discovery*, ed. by S. Fajtlowicz, P.W. Fowler, P. Hansen, M.F. Janowitz, F.S. Roberts (American Mathematical Society, Rhode Island, 2005) pp. 39–70
6. H. Mélot, Discrete Appl. Math. **156**(10), 1875–1891 (2008)
7. J.W. Berry, N. Dean, M.K. Goldberg, G. E. Shannon, S. Skiena, in: *Graph Drawing*, ed. by G.D. Battista (Springer, Berlin, 1997) pp. 425–437
8. D. Stevanović, V. Brankov, D. Cvetković, S. Simić. newGRAPH. http://www.mi.sanu.ac.yu/newgraph/
9. R.D. Pepper, in: *Graphs and Discovery*, ed. by S. Fajtlowicz, P.W. Fowler, P. Hansen, M.F. Janowitz, F.S. Roberts (American Mathematical Society, Rhode Island, 2005) pp. 341–349
10. E. DeLaVina, Graph Theory Notes of New York, **XLII**(3), 26–30 (2002)
11. G. Brinkmann, E. Steffen, Ars Combinatoria **50**, 292–296 (1998)
12. B.D. McKay, J. Algorithm. **26**, 306–324 (1998)
13. G. Brinkmann, G. Caporossi, P. Hansen, J. Algorithms. **45**, 155–166 (2002)