MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **58** (2007) 403-422

ISSN 0340 - 6253

FuiGui: A Graphical User Interface for Investigating Conjectures About Fullerenes

Wendy Myrvold * Bette Bultena Sean Daugherty
Bradley Debroni Sameer Girn Marsha Minchenko
Jennifer Woodcock
Department of Computer Science, University of Victoria,
Victoria, B.C., CANADA, V8W 3P6

Patrick W. Fowler †
Department of Chemistry
University of Sheffield
Sheffield, S3 7HF, UK

(Received July 31, 2006)

Abstract

Fullerenes are all-carbon molecules whose molecular structures correspond to 3-regular planar graphs that have face sizes equal to five or six. *FuiGui (FUllerene Interactive Graphical User Interface)* is a Java program under development whose goal is to aid the exploration of fullerenes and their parameters. This paper describes FuiGui and the design challenges faced when trying to make a fun and effective research tool.

1 Introduction

Fullerenes are all-carbon molecules whose underlying structures are planar graphs which have face sizes equal to five or six. There have been many publications on their chemistry.

*wendym@cs.uvic.ca, Research supported by an NSERC discovery grant

†P.W.Fowler@sheffield.ac.uk, Supported by the Royal Society/Wolfson Research Merit Award Scheme

Theoretical aspects of their chemistry and physics are discussed in An Atlas of Fullerenes [5]. Various properties of the graphs are of interest to chemists. For example, independent sets play a role in describing the the structure of addition patterns when bulky atoms bond to the carbon cage [6]. Perfect matchings correspond to the Kekulé structures (double bonds) in the carbon cage. There are various other parameters, such as the number of Hamilton cycles, that are of interest from a graph theory perspective.

Our goal is to develop a tool which aids researchers desiring to investigate conjectures about graph theoretic parameters of fullerene graphs. The new system is called *FuiGui (FUllerene Interactive Graphical User Interface)*.

This paper describes the design of FuiGui. Although the initial focus is on fullerenes, there is no reason why the template could not also work for other classes of graphs.

2 Existing Software

Various programs for automated conjecture generation have been written, with Graffiti being the first to make interesting research conjectures [3, 2]. Larson [13] provides a survey of these types of systems. Our approach is currently very different from these types of systems because the goal is to aid the graph theorists in making the conjectures rather than developing them automatically.

There are also several programs available that enable visualization of graphs and computation of some of their parameters. The one closest to what we have created is the CaGe program [11]. For fullerene pictures, this program is superior in many cases to FuiGui and it offers both 2-dimensional and 3-dimensional views. However, it was not created to facilitate exploration of fullerene parameters. Also, it is currently only available on machines running a Bourne-style shell (which rules out most laptops and personal computers).

Kocay's Groups and Graphs [12] program is another system which creates nice pictures of fullerenes. But it is awkward to explore large sets of them as each one is placed in a file by itself before the pictures are created. Also, this program only has full functionality on Macintosh systems.

There is a large collection of graph algorithms in the Combinatorica package of Mathematica [15]. GRaph INterface (GRIN) [14] is a stand-alone system which allows graph editing, visualization and computation of some parameters. Various other drawing systems are also available but do not have the parameter browsing capabilities of FuiGui.

The AutoGraphiX program [1] has a very appealing interface for graphing the ranges of parameter values which allows easy access to the extremal cases. No similar interface is currently included in FuiGui but it is being considered for future addition.

3 Using Java

Java was selected as the language for creating FuiGui. It has the disadvantage in that Java code can run significantly slower than code written in other languages such as C. However, it has the major advantage that it is very easy and fast to design user interfaces and draw pictures.

Java has a further advantage in that it is very portable. FuiGui has been run on a wide variety of computer types (both Mac's and PC's) and operating systems (including Windows XP, Solaris, Linux, and Mac OS/X) and there have been no problems encountered because of porting it to other machines.

CaGe [11] uses a mixture of C and Java in order to take advantage of the interface capabilities of Java and the speed of C. But this means that it is not as easy to port it to other systems.

4 Fullerenes

FuiGui requires files of fullerenes for its operation. For an input format, we wanted something compact so that the system could more easily handle large quantities of fullerenes. But it is also desirable to have a format which is human readable since this makes it easier to create the files and to understand input problems. For this reason a format based on fullerene face spirals (defined below) is used as the fullerene input format.

One way to select a path in an embedded graph is as follows. First, select a direction (clockwise or counterclockwise) and a starting edge (u, v). When the last edge of the path being created is (w, x), augment by adding edge (x, y) where y is the first unvisited neighbor of x which comes after w in the chosen direction. Continue until the last vertex has no unvisited neighbors. If such a path is a Hamilton path then it is called a *vertex spiral* of the embedded graph.

A *face spiral* of a fullerene is a vertex spiral of its dual. Not all fullerenes have face spirals. The smallest known example of a fullerene without one has 380 vertices [5, pp. 36-37]. The number of vertices in a smallest fullerene without a face spirals is still an open question. Gunnar Brinkmann has confirmed that all fullerenes on up to 200 vertices have face spirals (the result up to 176 is published [9] and he has continued the search since then).

Face spirals formed the basis of the first algorithm for generating fullerenes [5, pp. 27-31]. Because not all fullerenes have face spirals, the algorithm will not generate all of them. However since the approach becomes too slow before reaching the region where face spirals do not exist, this is not a problem for practical purposes. The spirals can be described by a sequence of 5's and 6's by writing down the degrees of the dual vertices in the order that

they are traversed. The lexicographically smallest face spiral sequence has been selected by chemists as a canonical form for a fullerene. The nomenclature for the fullerenes is then derived by sorting all canonical forms for the n vertex fullerenes in lexicographic order and then using $C_n : k$ to denote the kth n-vertex fullerene. For example, the 1812 different 60-vertex fullerenes are denoted as isomers $C_{60} : 1$ to $C_{60} : 1812$.

The isolated pentagon fullerenes have a similar notation of $C_n : k$ where it is just the isolated pentagon fullerenes which are considered. So to avoid confusion, it is necessary to specify when using this notation if it refers to all fullerenes or just the ones with isolated pentagons if this is not clear from the context.

Instead of listing a long sequence of 5's and 6's to represent a face spiral, it suffices to specify n and then list the 12 positions in the sequence of the five pentagons (in FuiGui fullerene files, these positions are numbered starting with 0). For example, the fullerene in Figure 1 with the face spiral indicated can be specified as:

40 0 1 5 8 9 11 12 14 15 17 19 21.

This is a lexicographically smallest face spiral - cases which start with three pentagons all short circuit before completing.

The fullerene files for FuiGui consist of spiral sequences of individual fullerenes, typically one per line. This allows the flexibility to group a set of fullerenes into a file for easy comparison. Such a group could be for example, all fullerenes of a given order, or fullerenes of various orders which share a certain property. This flexibility is helpful when exploring conjectures.

The fullgen program [10] offers formats which are more sparse but we are currently not using them because they are not human readable. A *rotation system* (an adjacency list with the neighbors of each vertex listed in clockwise order) is even easier to manage from a human perspective. We computed these as well and have been using them for computing parameters.

There are 285,914 fullerenes on 100 vertices. Listing the pentagon positions takes roughly 14MB of space. The adjacency list format uses 356MB.

The face spiral fullerene generation algorithm works well for small fullerenes but becomes far too slow for the larger ones. The fullgen program [10] is much faster and it is the only practical approach for generating the larger fullerenes. But it does not generate them in the order corresponding to the chemistry nomenclature. In order to take advantage of the speed of fullgen yet still retain the chemical ordering, we created a file of n-vertex fullerenes for each n ranging from 20 to 120 by generating them with fullgen and then sorting the minimum face spirals.

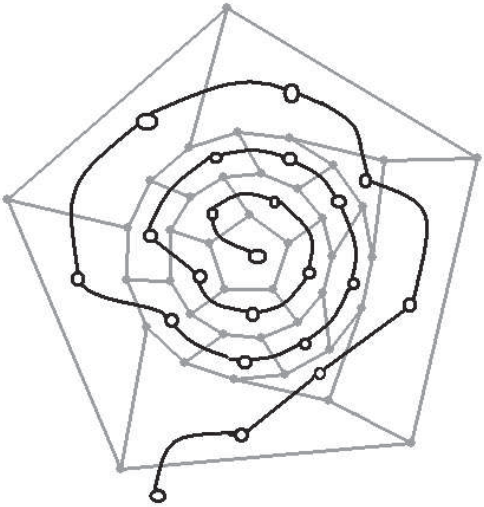

Figure 1: A lexicographically minimum face spiral of isomer C_{40} : 40.

5 Parameters

There are many interesting graph parameters (for example, the number of Hamilton cycles) that are difficult to compute. As a result, it is not possible to compute these on the fly and still have a system which is interactive. FuiGui overcomes this limitation by computing these parameters in advance.

The parameters can be integers (e.g. maximum independent set order), strings (e.g. the name of the automorphism group) or floating point values (e.g. an eigenvalue of the adjacency matrix). Given a file of fullerenes, each of its parameters is stored in another file where the parameter values are listed in the same order as the fullerenes.

The parameter values are designed to be "plug and play". A user can compute the parameter using any programming language (e.g. Fortran). A *header file* is used to specify the names, types and file names for each of the parameters associated with a file of fullerenes. Each parameter also has an associated text file which can be used to summarize the data. For example, a header file for the collection of 40-vertex fullerenes might look like this:

40-vertex fullerenes
f040 3
"Group Name" String gn040 gn040.summ

"Maximum Independent Set" int ind040 ind040.summ
"HOMO Eigenvalue" double homo040 homo040.summ

The first line gives a name (40-vertex fullerenes) for the collection of fullerenes. The second line gives the name of the fullerene file (f040) and the number of parameters (3). Each following line lists the name of a parameter, its type, the file that contains the parameter values and the summary file name.

Most previous systems have the code for computing the parameters as part of their system. The major advantages of FuiGui's approach are:

1. Parameter addition requires no collaboration with the FuiGui developers and so parameters which we have not considered can easily be added.

2. The code for FuiGui does not require modification to add a new parameter, and users wanting to add a parameter do not need to know how to program in Java.

3. Parameters that are difficult to compute (e.g. number of Hamilton cycles) can be computed in advance and then browsing is fast and interactive.

4. Even for parameters that are fast to compute (e.g. the number of face spirals), the system would slow down substantially on large files of fullerenes (e.g. the 285,914 fullerenes on 100 vertices) if computed on the fly.

6 The Interface

FuiGui is launched as a stand-alone applet. This must be done with an appletviewer. Trying to run it from a web browser does not work because for security reasons, this prevents access to the files which drive FuiGui.

A typical screen shot is shown in Figure 2. The first step is to select a file of fullerenes. Clicking on the "Select File" button results in a prompt for the header file name. The second button is a drop down menu which permits the selection of one of the fullerene parameters from the header file or the graph number. The top window on the left hand side displays the parameter information for the currently displayed fullerene. The window below that is the summary information for the parameter selected.

The summary window in Figure 2 shows that there are three 40-vertex fullerenes that have maximum independent set order 18. The first of these can be viewed by typing 18 into the box next to the parameter name and then clicking on the "First one" button. The user can then navigate forwards and backwards through the fullerenes which have maximum independent set order 18 by using the "+1" and "-1" buttons.

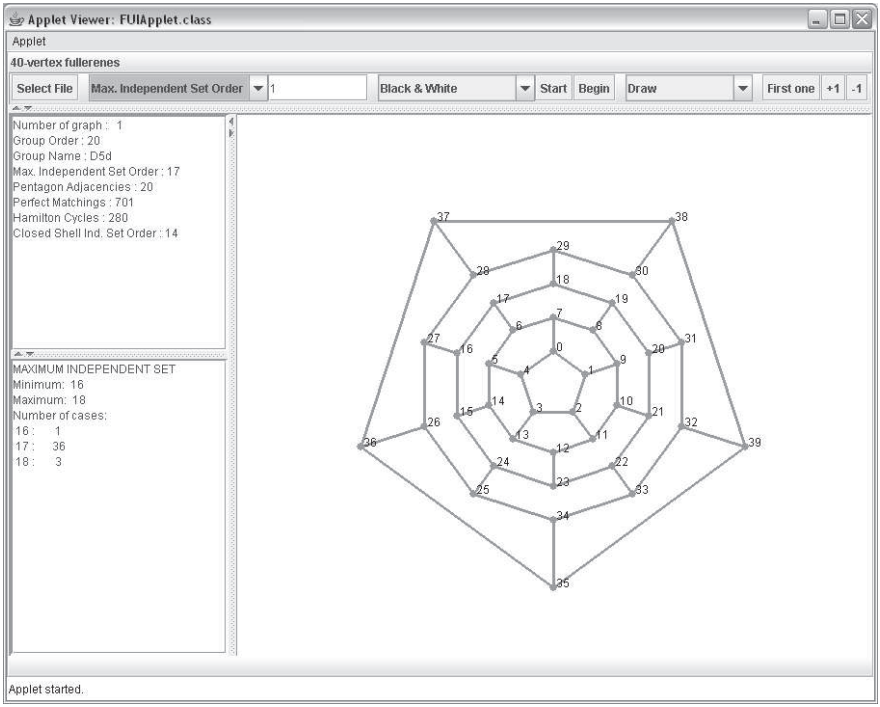

Figure 2: The interface for FuiGui

As mentioned earlier, we started with one file of fullerenes for each n up to 120 sorted so that the graph number is the isomer number. To get a specific isomer number, select graph number from the pull down menu, type the isomer number in the box beside that, and then use the "First one" button. The "+1" and "-1" buttons work slightly different in this case taking the user from graph number k to $k+1$ or $k-1$. In cases where the graphs are an arbitrary collection (for example, one such set we created had for each n all fullerenes with a minimum number of Hamilton cycles), the isomer number can be added as another parameter since in these cases, it will not be the same as the graph number.

There are various options for drawing the fullerenes. The current selection from the pull-down menu is "Black and White" which is useful for journal paper figures that cannot be made in color. The other options are:

1. The default picture can be selected to be face, vertex or edge-centered.

2. The graph displayed can be one of the primal, the dual, the primal and the dual superimposed, the primal plus the dual dots (dots means that just the vertices but not the edges are drawn), the dual plus the primal dots, the primal dots, and the dual dots.

3. There are various ways in which the fullerene can be colored. The default coloring has a cyan background, yellow vertices and red edges in the primal and blue vertices and purple edges in the dual. A "Pentagon Patches" coloring makes it very easy to see the locations of the pentagons. Vertices in three pentagons are colored blue, those in one or two are purple and those in no pentagons are yellow. Edges in two pentagons are blue, those in one are purple, and the rest are yellow. A "Site Group" coloring makes it easier to see the fullerene symmetries. The edges and vertices are given different colors to indicate the ways that there are automorphisms which map them back to themselves.

4. There are several options for labeling the vertices. The "Vertex Numbers" option labels them from 0 to $n-1$ according to the initial creation of the graph from the face spiral. The "No Labels" option can be used to get pictures without labels. The "Num Spirals" option changes the labels to indicate the number of successful vertex spirals starting at each vertex. If the dual dots are also displayed, their labels indicate the number of successful face spiral starts of the primal graph. An example of this is given in Figure 3. This might help to provide insight into when spirals fail, which could help in either finding a smaller fullerene with no spirals than the one on 380 vertices or proving that no such example can exist. The "Site Groups" option labels each vertex with the number of automorphisms mapping it

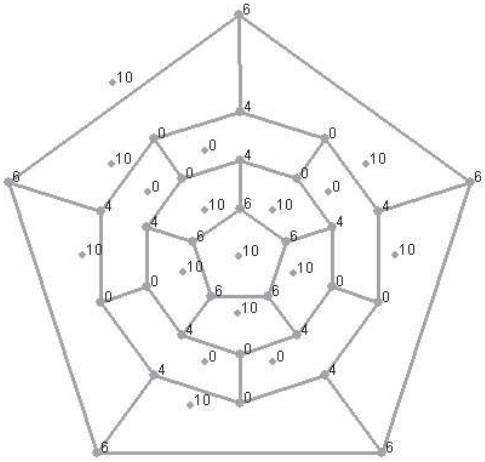

Figure 3: Labeling with the number of successful spiral starts.

to itself. By displaying the dual dots, the number mapping a face to itself can also be seen. The "Canonical Form" option labels the vertices from 1 to n according to the clockwise breadth first search which results in a lexicographically smallest adjacency matrix (following [7]).

5. The options "Specified Vertex Center", "Specified Face Center", and "Specified Edge Center", when selected, result in the picture being recentered on the closest vertex, face or edge, respectively, when a user clicks on the picture of the fullerene. The option of having the picture recentered on the closest graph component was not used because it could be too hard to determine, for example, if a given click was intended to be on an edge or on a face.

7 Fullerene Pictures

Some of the other drawing packages which can draw fullerenes result in pictures that have very dense regions where the faces are very small and condensed and other regions where the faces are much larger. The intent in designing the fullerene pictures was to make them look as much as possible like the Schlegel diagrams that a chemist would draw.

Internally, the vertex locations are recorded as having a level and an angle, where level zero is the middle of the picture. The other levels represent concentric circles. The angle varies from 0 to 359 and gives the position on the circle with radius equal to the level. The level can be any double value but for starting purposes, the even integral levels were the only ones used for vertex placement.

To draw the picture, the part of the graph at the center is labeled to be at level two if it is a face or an edge or level 0 if it is a vertex. The rest of the vertices are labeled in a breadth-first search type of way such that any vertex on a face with a vertex already labeled as level k is placed on level $k + 2$. If the edges are then given labels equal to the average of their vertex labels, the resulting labeling is the same as the one used in preparation to reduce a graph in [4].

The next step is to place the vertices in the appropriate cyclic order evenly spaced on each level on a circle with radius equal to the level. Then working from the inside out, each circle is rotated and a penalty is computed which reflects the differences in the angles between the vertices on that level and their neighbors on the level below it. The rotation with minimum penalty is selected for the picture.

The resulting pictures can have glitches. One common problem occurs when a vertex v on a level k is adjacent to two vertices at level $k - 2$ and hence only one vertex on the circle at its level. The two vertices beside v on level k are then adjacent, and this edge crosses the edges from v to its level $k - 2$ neighbors. To fix the glitch, the vertex v is removed from its circle and the other vertices on its circle and at higher levels are pushed out two levels.

8 Recentering the Pictures

The default pictures are not always the most aesthetically pleasing ones. The options to recenter on any of the vertices, edges or faces offer additional flexibility. One effective tactic for finding a picture for a fullerene which has symmetries is first to select the "Site Group Coloring" and the "Site Group" vertex labeling. The user can select graph components to center the picture to give a better display of the symmetry.

For example, the graph on the left of Figure 4 has two faces with 10 symmetries mapping the face to itself. Putting one of these faces in the middle results in the more

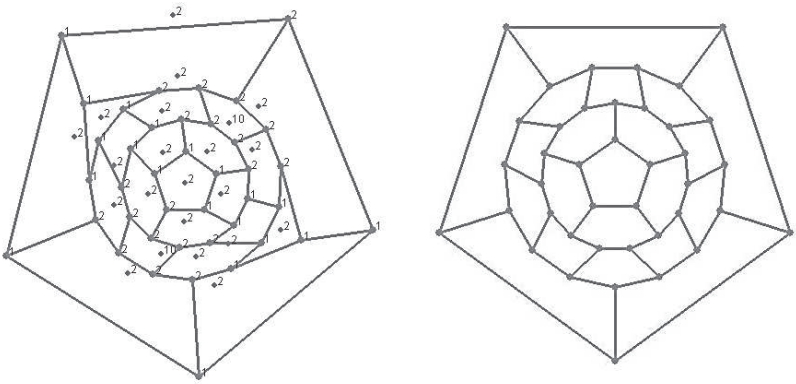

Figure 4: Recentering on a face.

symmetric picture on the right hand side. Similarly, the fullerene in Figure 5 has two faces with 12 symmetries mapping the face to itself.

The fullerene in Figure 6 has a picture on the left which has what we call *floaters* (the two degree one vertices are not actually vertices of the graph but are indicators that there is an edge which can be reinstated by identifying these two points). The floater vertices are green in the colored pictures to distinguish them from the other vertices. With a vertex labeling, they are labeled with the name of the other vertex the edge connects to. Recentering on the vertex which has 6 symmetries mapping it to itself results in the nicer picture on the right.

Pictures can also be centered on an edge. The edge on the left hand side labeled with e is colored red in the "Site Group" colored pictures to indicate that both a reflection and a rotation map this edge to itself. This can be more easily seen in the picture on the right of Figure 7 that is centered about this edge.

9 Automorphism Group Names

There are 28 different point groups that a fullerene can have [5, Ch. 5]. Knowledge of the group is useful for many chemical purposes. It is not difficult to compute the group names. However, writing the code was tortuously tedious. To save others the efforts of

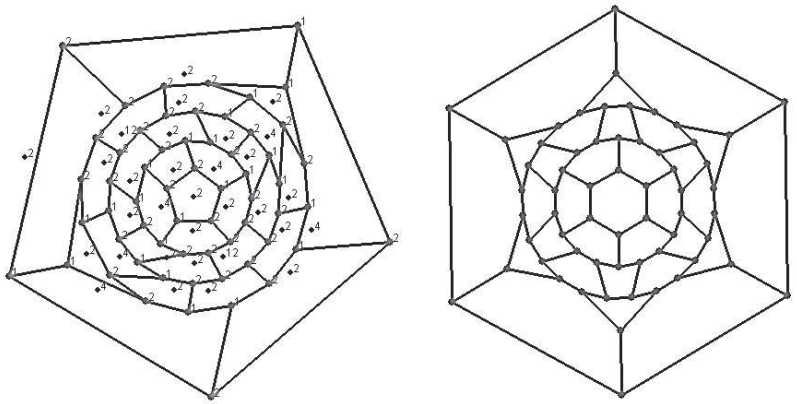

Figure 5: Another example of recentering on a face.

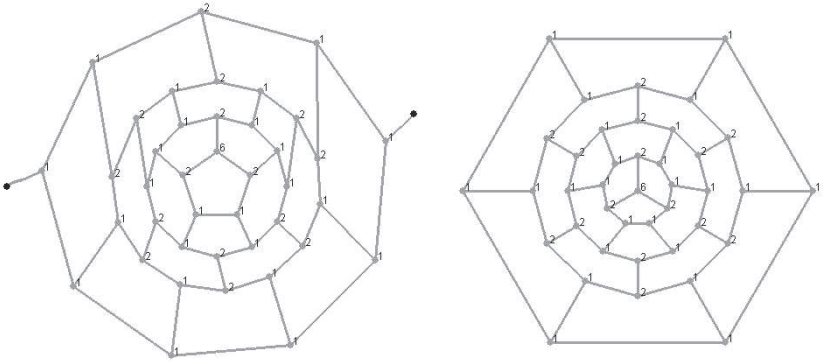

Figure 6: Recentering on a vertex.

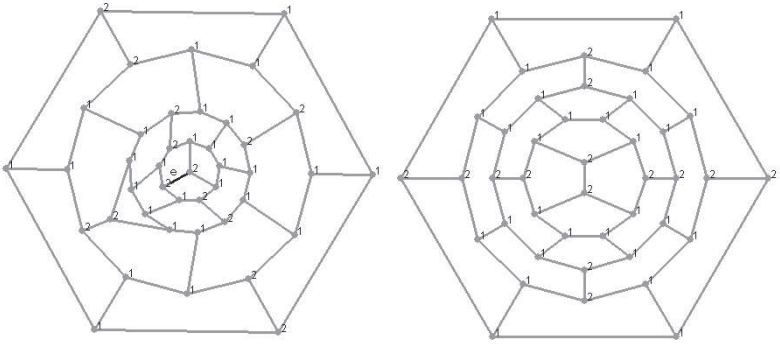

Figure 7: Recentering on an edge.

rediscovering conditions which identify the group names, pseudocode for an algorithm is included here.

The first step is to label the fullerene with a clockwise breadth-first search starting with a vertex on a pentagon (as done in [7]). The next step is to compute all the permutations which are automorphisms. These correspond to the clockwise and counterclockwise BFS renumberings which result in the same adjacency list as the initial clockwise BFS. Since there can only be agreement if the BFS starts on a pentagon vertex (because the original BFS did), only a constant number of such BFS steps must be performed and each takes $O(n)$ time.

The *order* of the automorphism group is the number of permutations in the group. There is an *inversion* if some counterclockwise BFS results in an automorphism which does not fix any of the edges. The *order* of a permutation is the least common multiple of the cycles sizes when written in cycle notation. The fullerene is *chiral* if none of the counterclockwise BFS's result in automorphisms and *achiral* otherwise.

There is only one possible group if the group order is 1, 3, 10, 60, or 120. These easy cases are handled first:

If the automorphism group order is

1: return("C_1")

3: return("C_3")

10: return("D_5")
60: return("I")
120: return("I_h")

Otherwise, count the number of permutations with each order. Since there are at most a constant number of permutations (120 is the maximum for a fullerene, but the order is at most 24 if this part of the code is required), this step takes $O(n)$ time. Note also that determining if a fullerene is chiral is easily computed when finding the automorphisms, and testing for an inversion takes at most $O(n)$ time.

If the automorphism group order is

24: if there are 8 permutations of order 6 return("T_h")
else if there are 6 permutations of order 4 return("T_d")
else if there are 6 permutations of order 6 return("D_{6h}")
else return("D_{6d}")

20: if there is an inversion return("D_{5d}")
else return("D_{5h}")

12: if the fullerene is achiral
    if there is an inversion return("D_{3d}")
    else return("D_{3h}")
else
    if there are 8 permutations of order 3 return("T")
    else return("D_6")

8: if there are 2 permutations of order 4 return("D_{2d}")
else return("D_{2h}")

6: if the fullerene is achiral
    if there are 2 permutations of order 6
        if there is an inversion return("S_6")
        else return("C_{3h}")
    else return("C_{3v}")
else return("D_3")

4: if the fullerene is achiral
    if there are 0 permutations of order 4

if there is an inversion return("C_{2h}")
else return("C_{2v}")
else return("S_4")
else return("D_2")

2: if the fullerene is achiral
if there is an inversion return("C_i")
else return("C_s")
else return("C_2")

None of the above steps takes more than $O(n)$ time. Thus, this approach for computing the group name takes $O(n)$ time in total.

In order to verify that this algorithm is correct, the results were checked with the tables in the Atlas of Fullerenes [5, pp. 106-107]. The results matched exactly taking into account that the book has a known typographical error. Table 5.2 states that there are three isomers of C_{100} of T symmetry and none of T_d but the correct result is that there are two isomers of T symmetry and one of T_d. This has been corrected in a soon to be released reprinting of the book.

10 Animated Algorithms

FuiGui includes the capability to provide animated algorithms for instructional or research purposes. Users wanting to add their own animated algorithms must update the Java file Animated.java to plug the algorithm into the system and provide a class for implementing the animated algorithm.

The current design results in only limited change to the existing system to plug in a new algorithm and the process was successfully carried out by students in a recent Graph Algorithms class. The five steps required for plugging in a new algorithm are:

1. Add the name of the algorithm to an array of strings which give the names of the available animated algorithms.

2. Declare an object whose type corresponds to the new class in which the algorithm is implemented. This is used to keep track of the current state of the algorithm.

3. Add a call to the constructor for the animated algorithm and set a delay increment for the timer.

4. Add a call to the routine which implements the next step of the animated algorithm.

5. Add code to get rid of the object and do any necessary clean up when the algorithm terminates.

It would be less cumbersome to be able to implement animated algorithms so that they perform their task and when desired, the picture on the screen is updated. Unfortunately though, because of the way Java works, the system can opt not to bother updating the current picture when it is in the middle of executing a thread. For this reason, the design approach has been to develop animated algorithms with a next step routine which performs the next step of the animation and returns control to FuiGui after the picture has been updated. The animation continues with repeated calls to this next step routine. Buttons are available on the main interface which allow an animated algorithm to pause, stop, or resume the execution.

Some of the animations currently available are:

1. An illustration of how the fullerene pictures are created by first placing an initial pentagon or hexagon, assigning vertices to levels, and then rotating levels one at a time in order to find an optimal placement for them.

2. The user can specify a starting vertex and then all six of the possible ways to create a spiral at that vertex are attempted until a spiral is found or the path short circuits.

3. The steps of a simple backtracking algorithm for finding perfect matchings.

4. Feo and Provan's algorithm [4] for finding a Delta-Wye-Delta reduction of a planar graph.

5. The Hamilton cycles of the fullerene are shown one at a time.

6. The fullerene can be moved left, right, up or down, and also it is possible to zoom in or out on the picture. Because of the way the pictures were designed it is extremely easy to implement these.

11 Use on Research Problems

So far, time spent has been mainly focussed on development of the FuiGui system and it has only seen limited use as a research tool. However, it has shown to be very valuable on at least one application where it has been applied- in a search for pairs of non-isomorphic graphs (fullerenes or fullerene duals) with adjacency matrices having identical multi-sets of eigenvalues.

It is not known if there exists any fullerenes which are pairwise nonisomorphic yet the adjacency matrices have the same eigenvalues. However, there are several examples of pairs of fullerenes where the duals have the same same eigenvalues [8].

These fullerenes and their parameters were extracted from the files with a C program applied to an input file which had for each interesting graph, the number of vertices and its isomer number. One surprising result which was easily noticeable when FuiGui was invoked on this collection of graphs is that for each pair of graphs, the two fullerenes have the same number of Hamilton cycles. FuiGui facilitates exploration of the observation that a reasonable conjecture is that there are infinite families of pairs. For example, four graphs which start a family are shown in Figure 8. This observation is potentially relevant to the chemistry of both carbon and boron, in that the primal graphs define series of closed carbon nanotubes with specific shared invariants, and their duals define structural candidates for tubular bare-boron clusters that would share π-electron energies, with consequences for their spectroscopy and magnetic properties.

12 Future Enhancements

The code has already proven very useful in preparing pictures for journal papers and presentations. It also makes it much easier to explore the fullerenes and their various parameters. It is also valuable as a teaching tool, in particular, in the demonstration of the algorithms that are animated.

Not all of the fullerene pictures are nice ones. Further experimentation with drawing algorithms and addition of new approaches would be a benefit. There are also many parameters and animated algorithms which could be added.

It would be interesting to extend the capabilities of FuiGui to handle other classes of graphs. The only big difficulty in extending it is that a routine for drawing the graphs is required.

FuiGui has capabilities that other systems do not and other systems have features that FuiGui does not have. As a long term goal, an ideal approach would be to try to tie together the capabilities of the various systems which have been created. Hopefully these workshops on Computers and Discovery will result in a unified system which includes the abilities of the various types of research tools which have been developed.

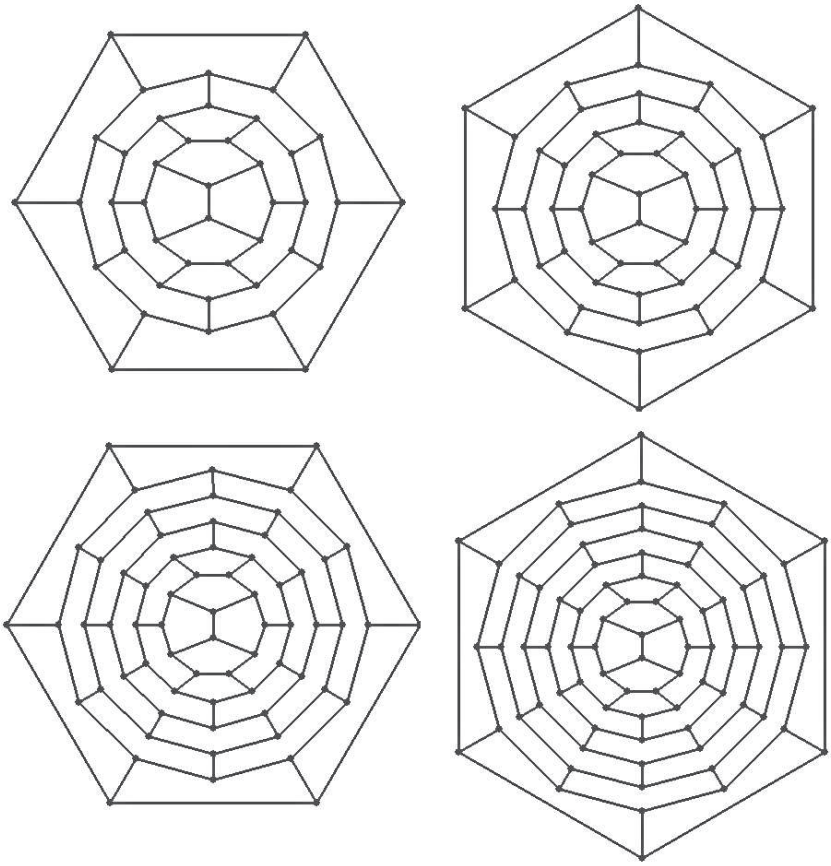

Figure 8: The first four fullerenes in an infinite family.

References

[1] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs 1: The AutoGraphiX system. *Discrete Math.*, 212:29–44, 2000.

[2] Ermelinda DeLaVina. Some history of the development of Graffiti. In *Graphs and Discovery, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, volume 69, pages 81–118, 2005.

[3] Siemion Fatlowicz. On conjectures of Graffiti. *Discrete Mathematics*, 72:113–118, 1988.

[4] T. A. Feo and J. S. Provan. Delta-wye transformations and the efficient reduction of two-terminal planar graphs. *Operations Research*, 41:572–582, 1993.

[5] P. W. Fowler and D. E. Manolopoulos. *An Atlas of Fullerenes*. Oxford University Press, 1995.

[6] P. W. Fowler, K. M. Rogers, K. R. Somers, and A. Troisi. Independent sets and the prediction of addition patterns for higher fullerenes. *J. Chem. Soc. Perkin 2*, pages 2023–2027, 1999.

[7] Patrick Fowler, Daniel Horspool, and Wendy Myrvold. Vertex spirals in fullerenes and their implications for nomenclature of fullerene derivatives. *Chemistry: A European journal*, 13(8):2208–2217, 2007.

[8] Patrick Fowler, M. J. Roberts, and Wendy Myrvold. Cospectrality in fullerene duals. In preparation, 2006.

[9] G.Brinkmann. Problems and scope of the spiral algorithm and spiral codes for polyhedral cages. *Chemical Physics Letters*, 272(3–4):193–198, 1997.

[10] G.Brinkmann and A. W. M. Dress. A constructive enumeration of fullerenes. *Journal of Algorithms*, 23:345–358, 1997.

[11] G.Brinkmann, O. Delgado Friedrichs, A. Dress, and T. Harmuth. CaGe- a virtual environment for studying some special classes of large molecules. *Match Commun. Math. Comput. Chem.*, 36:233–237, 1997.

[12] William Kocay. Groups & Graphs- Software for graphs, digraphs and their automorphism groups. See software corner of this issue, 2006.

[13] C. E. Larson. A survey of research in automated mathematical conjecture-making. In *Graphs and Discovery, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, volume 69, pages 297–318, 2005.

[14] Vitaly Pechenkin. GRaph INterface (GRIN). http://www.geocities.com/pechv_ru/, 2007.

[15] Sriram Pemmaraju and Steven Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, 2003.