CaGe V0.3
User's Guide

Thomas Harmuth
harmuth@@mathematik.uni–bielefeld.de

February 1997

Contents

1 Introduction

CaGe is a computer program which takes the output of some graph generation program as input and provides several output styles, especially graphical output. The name of the program is based on the two tasks *gen*erating and *view*ing which can be done by *CaGe*. Windows, buttons etc. are used to interact with *CaGe*. *CaGe* is based on Tcl/Tk and thus behaves like typical Tcl/Tk–applications. Since the generated graphs can be interpreted as molecules, *CaGe* is interesting not only for graph theorists, but especially for chemists.

Since chemists use a rather different terminology than graph theorists, here is a short list of correspondences: An *atom* in chemistry is represented by a *vertex* in graph theory. The same way, a *bond* between two atoms is represented by an *edge*. A double or triple bond is only represented by a single edge. A vertex is *adjacent* to another vertex if they are connected by an edge. So, if two vertices are adjacent to each other, this means that (considered as atoms) they have bonds. The *valence* of a vertex is the number of adjacent vertices. Since the vertex is interpreted as an atom, it is the number of other atoms it has bonds with, but not the number of bonds itself. E.g. for a carbon atom, the valence is **not** always 4: Double bonds are considered as one adjacency.

A *graph* (sometimes a graph is also called *map*) corresponds to a *molecule*. A graph is *x–regular* if every vertex is x–valent. So this means that every atom has bonds to exactly x other atoms.

2 Tutorial — Tour de CaGe

Let's suppose you have started *CaGe*. If you do not know how to do this, have a look at the `README` file which is provided together with *CaGe*. After you have started the program, a message appears. After a few seconds either a window or the frame of a window appears. In the first case new windows are placed automatically, in the second case you must place every new window yourself by moving the frame (this is done by moving the mouse) and pressing the left mouse button afterwards.

There are two windows which appear: The **main window** and the **selection window**. The main window is titled "CaGe V0.3" while the selection window is just titled "CaGe".

Main window: The main window will stay on the screen as long as *CaGe* is running. You can always exit *CaGe* by clicking the "Quit" button in the main window (clicking a button on the display means that you press the left mouse button while the mouse pointer is located on that button). You should exit *CaGe* only this way. You should never make use of the window options which are offered by your window manager (close, delete, destroy...). When a generation program is running, the main window provides some information about the state in which the generation program is.

Selection window: You can choose a graph generation program via the selection window. All generation programs which are provided and the kind of graphs which they generate are listed in the selection window. Actually, there are five different programs which will be discussed later. There is a little space between the third and the fourth entry in the list because the last two programs are intended to support not only chemical applications, but various tasks.

Let's choose the first program as an example. So we want to generate fullerenes, and the program which does this is called *fullgen*. As soon as you click the "Fullerenes (fullgen)"–button, the selection window disappears and another window appears which is called the **option window**. Here you can change some parameters which determine what kind of fullerenes shall be generated and where the output shall be sent to. To change the parameters, different **widget types** are supported. One widget type you have already used is the **button**. By clicking a button you invoke a certain process (e.g. entering a new window). There are some other types:

Slider: A slider contains a button which can be moved and a container in which the button can be moved. By moving the button you change the value of a parameter. This value is shown either above the button or to the left of the container. To move the button, put the mouse pointer on it, press the left mouse button and hold the left mouse button while moving the mouse. To change the parameter value for the smallest possible unit, put the mouse pointer on the container and press the left mouse button. If the parameter value becomes smaller or bigger depends on the position of the mouse pointer in relation to the slider button.

Checkbutton: A checkbutton is a small light with a label at its right side. The light can be either on (red) or off (grey). You can turn the light on or off by clicking it. A checkbutton is used to mark something.

Entry: An entry can be used to show text. You can change this text by clicking the entry. In this case a cursor appears and the program accepts keyboard input. If the text is long, then it might happen that the entry is too small to show the whole text. But it is nevertheless there, it will not be cut.

Radiobutton: Radiobuttons always appear in groups. They look like checkbuttons, but the light is a rhombus instead of a square. Only one radiobutton in each group can be switched on. Whenever you turn one radiobutton on, all other radiobuttons in this group will be turned off automatically.

Label: Labels are used to show text. If the text represents a parameter, then it might change. But different to an entry the user cannot change the text himself.

Most widgets that can be used interactively will highlight (with white colour) whenever the mouse pointer is put on them. By pressing the left mouse button you invoke the widget. For example, you do not need to put the mouse pointer exactly on the checklight to invoke a checkbutton. It suffices to put the pointer somewhere on the related label.

The **parameters** can be split into three categories:

Basic options: In *fullgen*, there is only one basic option. It is the number of vertices (atoms) each generated fullerene must have. In the first row of the window there are two sliders. By moving one slider button you can change the minimum number of vertices or the maximum number of vertices each generated fullerene must have. Note that the other slider moves simultaneously whenever you move one slider. This is because the "min=max"–checkbutton at the right of the sliders is switched on. When the checkbutton is turned off you can move the two sliders separately of each other. By default, *CaGe* switches on the "min==max"–checkbutton because it assumes that you want to generate fullerenes with a fixed number of vertices. As you will see, *CaGe* often does such assumptions to minimize the amount of changes you need to do (especially if you change output options). So sometimes it switches checkbuttons on or off automatically when you invoke another widget. Another reason for automatical changes is that not all parameter values fit to each other.

Whenever there is a combination of values which is not allowed, *CaGe* tries to change the values itself.

Output options: The output options are the same for all generation programs. So the option window for every generation program contains the same frame named "Output". For each generated graph there are basically three output styles: You can have the simple adjacency information of the graph, you can have an embedding into the two–dimensional Euclidean plane (such embeddings are called "Schlegel–diagrams") and you can have an embedding into the three-dimensional Euclidean space. While the adjacency information can only be stored into a file or sent into a pipe, the embedded graphs can also be shown on the display. You can invoke some of the options simultaneously, some others you can't. We will discuss this in detail in section **??** (see "output options" there).

Extra options: Like the basic options, extra options are different for every generation program. Extra options are handled separately from basic options because in most cases they are not needed. Sometimes the extra options are so sophisticated that you cannot use them without having a deep impression of how the generation program works. So we will discuss extra options later.

We will now do some example calls of *fullgen*. For the first call, please change the number of vertices to 60 and leave all other options unchanged. So, what we want to do is to generate all fullerenes with exactly 60 vertices. The default output option is to save the adjacency information of the generated graphs into a file. The name of this file is determined automatically and depends on the options we use. You can see the filename in the related entry box ("Full_codes_60"). The code which is used for saving is called *writegraph2d*. When you click the "Start"–button at the bottom of the option window, then the generation will begin. The option window vanishes. Since there is no output for display the only window which remains open is the main window. The main window provides some information to the user about what the generation program is doing. Like every other generation program *fullgen* performs two steps:

- Preparing generation
- Generating graphs

The program will need about 20 seconds to generate all fullerenes with 60 vertices. After the generation has finished, a window called "Fullerenes — Logfile contents" appears. *Fullgen* generates a so–called logfile into which it writes information about the performed generation that might be useful for the user. For example, the number of generated fullerenes is written into the logfile. If the logfile contents does not fit into the text display, you can use the slider on the right side of the display to scroll up and down inside the text. Except the slider, the "Logfile contents"–window has only one more button. By clicking this button (named "Continue") you close the window and return to the option window.

The purpose of this call was to generate and store simple adjacency information. Since (like all other generation programs) *fullgen* already provides this information we could have got it without the use of *CaGe*. We could have called *fullgen* directly. However, calling *fullgen* is much easier using *CaGe* since *CaGe* provides buttons, sliders etc. for the options. If you want to have a look at the generated file "Full_codes_60", you can load this file into an editor. We will give a description of the code in section **??**.

We are now back at the option window. For the second call of *fullgen*, please change the number of vertices to 44. Now we try to get embedded graphs. We try to get two–dimensional and three–dimensional embeddings in one go. Please switch off the checkbutton named "Adjacency information". This means that pure adjacency information will not be stored. As you see, all related checkbuttons and radiobuttons are switched off automatically. Now, please switch on the radiobuttons named "File" which are positioned in the same lines as the checkbuttons named "2D representation" and "3D representation". The latter two checkbuttons are automatically switched on. So, the checkbuttons named "3D representation", "2D representation" and "Adjacency information" are used to mark that the related generation will be done at all. To **avoid** one generation you can just switch off the related checkbutton. If you want to **include** a generation you can choose between various output directions (e.g. file and display) and it suffices to switch on the related **radio**button. If you switch on the related **check**button instead, *CaGe* chooses a default output direction.

The names of the files into which the embedded graphs are written are determined automatically. There is one file for 3D representation and one file for 2D representation. The names are similar to those determined for files which contain adjancency information. Only suffixes ".3d" and ".2d", respectively, are added. Now, please click the "Start"–button. Now the same action as in the first example call will take place. The only difference is that it takes much more time to generate embedded graphs than to generate pure adjacency information (that is why we wanted you to reduce the number of vertices to 44). For this reason *CaGe* tells the user about the number of already embedded graphs. *CaGe* shows this number in the main window. In this example, 89 graphs will be generated and embedded. Finally the logfile contents will be shown. After clicking "Continue", you will return to the option window.

Let us now have a third example. This time we want to generate embedded graphs and have a look at them on the display. We suppose you have already returned to the option window. Note that the parameters you have changed in the second example keep their values (e.g. the "Adjacency information"–checkbutton is still switched off). So the parameter changes we will now perform base on the second example. Please switch on the radiobuttons named "RasMol" (only if a program with this name is provided on your computer, otherwise you cannot have 3D embeddings on the display in this example) and "Display". The radiobuttons named "File" are automatically switched off. Furthermore, please change the number of vertices to 36. By clicking the "Start"–button *fullgen* will be called.

The next task is to arrange the new windows which appear on the screen. One of these windows is called "RasMol Version *xxx*". *RasMol* is an external program which is very useful for displaying three–dimensional representations of graphs. It has various options which are not to be discussed here. Just notice that every 3D representation is sent to *RasMol*. Every 2D representation is sent to the window called "Schlegel diagram". The third window is called "Output panel". It contains buttons to control which graph is displayed. The numbers of the currently displayed graphs are printed in red. Normally the 2D and the 3D numbers are equal. It takes some seconds until the first graphs appear on the screen. To make the first graph appear in the *RasMol* window, it is necessary to move the mouse pointer inside this window (this is a peculiarity of *RasMol*).

You can change the view of both representations. To change the view in the *RasMol* window, use the sliders on the right hand side and at the bottom of the window. To change the view in the "Schlegel diagram" window, click inside a face of the 2D representation. This face is surrounded by an alternating cycle of vertices and edges, and after a few seconds a new 2D representation will appear where this cycle becomes the outermost cycle. If you click the button named "Show numbers", then every vertex will be numbered. Unlike for the other windows of *CaGe*, here it is useful to enlarge the window (how this is done depends on your window manager). The "Show numbers"–button is now relabeled. Its label is now "Hide numbers". If you click on the "Hide numbers"–button, the vertex numbers will vanish. If you want to save the Schlegel diagram in PostScript style, then click the button named "Save as PostScript file". **Before** doing that, first enter the name of the destination file in the "Filename"–entry below. Default is "unnamed.ps".

If you want to have a look at another graph, you can type the number of this graph directly into the entry named "Show graph no." in the output panel. Or you can use the "+1" or "+10"–buttons to approach to the next graphs: If x is the number of the currently displayed graph, then after clicking the "+1"–button it will be $x + 1$ and after clicking the "+10"–button it will be $x + 10$. If the number of available graphs is lower than $x + 1$ or $x + 10$, respectively (so that the desired graph does not exist), then *CaGe* will cancel the output and move to the logfile window. You can perform the same step by clicking the "Cancel"–button.

When graph x is shown on the display, you cannot turn to graphs with lower numbers than x, with one exception: All graphs that were shown on the display are stored in an internal list. You can go through this list using the buttons "<" and ">". Doing this, you enter the **review mode** (actually there is no hint on the display which says that you are in review mode). The first graph which is shown after entering the review mode is always the last graph in the list. Clicking "<", you go to the previous graph, clicking ">", you go to the next graph in the list. You leave the review mode by invoking one of the widgets described above (apart from "<" or ">").

There are two checkbuttons named "Save 2D" and "Save 3D". When you switch on the "Save 2D"–

button, the 2D representation which is currently displayed is saved in a file. Furthermore, every new 2D representation which appears on the display is also saved as long as the checkbutton remains switched on. The name of the file depends on the default name which is determined in the option window. A suffix ".s2d" is added. Switching on the "Save 3D"–button has the same effect for 3D representations. However, there is one difference when entering review mode: a 3D representation is only saved once even if it is reviewed several times, while a 2D representation is saved each time it is reviewed. This is necessary because you can change the 2D representation substantially by clicking a face, and you might want both the old and the new representation to be saved.

The last checkbutton which needs to be explained is the "skip"–button. While this checkbutton is switched on, all graphs that come in from the generation program are skipped. When you switch off the checkbutton, *CaGe* continues in the usual way: It embeds the next graph which comes in and displays it. This checkbutton is useful only if you know that the list of generated graphs is very large.

Let us suppose you have finished the generation process. There are three ways to do that:

- You have clicked the "Cancel"–button
- You have skipped over the last incoming graph
- You wanted to see a graph which did not exist (i.e. the number of the graph which should be shown was too high; in this example, there are 15 graphs)

After finishing the generation process, the "Logfile contents"–window appears as usual. With these three examples you have got an impression about what *CaGe* is good for. After clicking the "Continue"–button, the option window reappears. If you click the "Cancel"–button there, you return to the selection window where you can choose another generation program. To quit, you do not need to return to the selection window. You can click the "Quit"–button in the main window whenever you want. If other programs are running which were called by *CaGe* (e.g. the generation program or *RasMol*), then *CaGe* will stop them automatically.

3 Main program loop and external programs

Figure **??** shows the main program loop. The option window can be one out of five different option windows, one for each graph generation program. For each graph generation process various external programs are called. Figure **??** gives an overview. Each arrow indicates data flowing from one program or file to another. In figure **??**, programs are indicated by either their names or by "pipe" and surrounded by double–line rectangles while files are indicated by either their names or their contents and surrounded by single–line rectangles. During one graph generation process, some temporary files named "CaGex.tmp" are created (where $x = 1, 2, 3$). These files can be deleted after a generation process has finished. The same holds for other temporary files which might be created by graph generation programs (e.g. *tubetype*).

The program *spring* computes 3D representations and the program *schlegel* computes 2D representations. The programs *RasMol* and *GeomView* are used to show the 3D representations on the display. The pipe connects *CaGe* with another external program which receives the adjacency information. This program can be chosen by the user.

If you click the "Quit"–button in the main window while a graph generation process is running, not only *CaGe*, but all external programs which *CaGe* has called will be stopped simultaneously. The only exceptions are:

- *GeomView* must be stopped by the user.
- The pipe program is not stopped automatically. However, the pipe is closed, and this usually means that the pipe program will finish soon.

If you click the "Go to background"–button while a graph generation process is running, then *RasMol* will be stopped (if running) and the Schlegel output window will be closed (if open) so that no more graphical output is supported (furthermore, the output panel will be closed). However, the graph generation process will continue in the background, and *CaGe* will continue writing output into the destination files or into the pipe. As soon as the graph generation program will have finished, *CaGe* will be stopped. If you click the "Go to background"–button while no graph generation process is running, then *CaGe* will be stopped as if the "Quit"–button had been clicked.

4 The graph generation programs

In the selection window, all available graph generation programs are listed. Usually there should be five programs available. These five programs and their option windows are described below. However, some programs might not be available or some further programs might be added in future versions of *CaGe*. Programs which are not available are not listed in the selection window.

For every generation program there exists an own option window with parameters independent from each other. So if you change the output parameters in one option window, the output parameters in the other option windows remain unchanged.

4.1 *Fullgen*

Fullgen is a generation program which generates fullerenes. Fullerenes are spherically shaped molecules built of carbon atoms where every carbon atom has bonds to exactly three other carbon atoms so that each carbon ring consists of either five atoms (pentagon) or six atoms (hexagon). Unlike other fullerene generators, *fullgen* generates complete lists and it is currently the fastest fullerene generator. The generation of all fullerenes with 60 atoms takes less than one minute.

The number of vertices (atoms) can be selected by two sliders named "Minimal number of vertices" and "Maximal number of vertices". If the "Min=Max"–checkbutton is not switched on, then these two sliders can have different values.

The following description of the **output options** holds for all graph generation programs:

3D representation: Exactly if this checkbutton is switched on, then three–dimensional embeddings will be generated (using *spring*). Where these embeddings are sent to depends on which of the radiobuttons named "RasMol", "GeomView" or "File" is switched on. If you switch on the checkbutton, "RasMol" will be taken per default. But you can as well choose the output direction by directly selecting a radiobutton. The checkbutton will then be switched on automatically. The only possibility to prevent 3D embeddings is to switch off the checkbutton.

RasMol: The 3D representation will be sent to *RasMol* (if provided on your computer).

GeomView: The 3D representation will be sent to *GeomView* (if provided on your computer).

File: The 3D representation will be sent to a file. The file will contain data in *writegraph3d* code (see section **??** for description). The name of the file can be entered in the "Filename"–entry below. However, if the "automatic"–checkbutton is switched on, then the filename will be determined automatically. It is a good choice to let *CaGe* determine the filename, for two reasons: The first is that *CaGe* considers all options used while building the filename. So by having a look at the name of a file you know how its contents were generated. The second reason is that *CaGe* chooses the same filename that *fullgen* would if you generated the fullerenes without using *CaGe* (the only difference is that *CaGe* adds the suffix ".3d"). So the filenames are consistent. Now, because it is such a good idea to let *CaGe* determine the filename, *CaGe* automatically invokes the "automatic"–checkbutton if you switch on the "File"–radiobutton. If you want to choose the filename by yourself, then you should just enter the filename entry. In this case *CaGe* assumes that you want to store 3D embeddings in a file, so it switches on the "File"–radiobutton automatically. Furthermore, the "automatic"–checkbutton is switched off.

2D representation: Exactly if this checkbutton is switched on, then two–dimensional embeddings will be generated (using *schlegel*). Where these embeddings are sent to depends on whether the "Display" or the "File"–radiobutton is switched on. The behaviour of the checkbutton and the radiobuttons is the same as for 3D representation. The same holds for the filename (here the suffix ".2d" instead of ".3d" is added). It is possible to generate 2D and 3D embeddings simultaneously. However, it is not possible to direct one of these representations into a file and the other one onto the display. So if these two directions collide, then *CaGe* will switch off one representation checkbutton. If the 2D embeddings are sent to a file, then this file will contain data in *writegraph2d* code.

Adjacency information: Exactly if this checkbutton it switched on, then pure adjacency information will be generated. Where this information is sent to depends on whether the "File" or the "Pipe"–checkbutton is switched on (both can be switched on simultaneously). If the information is sent to a file, then for the name of the file the same holds as for the 3D representation (no suffix will be added). If the information is sent to a pipe, then the "Program call"–entry must contain a program call, i.e. a program name and — if needed — some options. An example of a program call is "cat". This program takes the adjacency information as input and writes it to the standard output channel.

There are two different codes in which the adjacency information can be represented: *writegraph2d* and *planar_code*. See section **??** for a description. Which code is chosen depends on which of the related radiobuttons is switched on. Default is *writegraph2d* since it is an ASCII code.

Widget interactions: You have seen that there are several widget interactions. There are various reasons for them:

Inferences: If you select a widget which provides a special option, then *CaGe* automatically sets all general options on which the special option depends. For example, if you enter the "Filename" entry for 3D representation, then *CaGe* assumes that you want to generate 3D embeddings and send them to a file. So it automatically switches on the "File"–radiobutton and the "3D representation"–checkbutton.

Group dependencies: This is the opposite to inferences. If a general option is not invoked, then all related special options must not be invoked. For example, if you switch off the "3D representation"–checkbutton, then no 3D embeddings will be generated. So all radiobuttons and the "automatic"–checkbutton are switched off (the "Filename"–entry could be cleared as well, but this is not done).

Conflicts: Most of the widgets are not independent of each other. Not all combinations of options are allowed. So if a combination is not allowed, then the related widgets are changed. An easy example is that you cannot choose a filename by yourself and by *CaGe* simultaneously. So if you invoke a "Filename"–entry, the related "automatic"–checkbutton must be switched off. Another example is described above (see item "2D representation"). Sometimes conflicts are prevented automatically by providing an appropriate widget type. For example, since it is not possible to direct 3D representations to *RasMol* and to *GeomView* and to a file simultaneously, radiobuttons instead of checkbuttons are provided to direct the 3D output.

Defaults: This is a special case of group dependencies where the special options to be chosen are not unique. In this case *CaGe* chooses default options. For example, if you switch on the "3D representation"–checkbutton, then by default the output will be directed to *RasMol*. Another example: If you switch on a "File"–radiobutton, then by default the related "automatic"–button will be switched on.

These interactions basically concern the output options. However, conflicts may also occur using extra options or basic options, so there the related technique is used as well.

Extra options: The following options are options which are rarely used. Some of them are so specific that you must know about the details of the generation program *fullgen*. Such options are not explained in detail. Instead the related option which you would have to use for a direct call of *fullgen* is provided. It is enclosed in brackets "{" and "}". For a further description of such an option see the original manual of *fullgen*.

IPR: If you switch on the "IPR"–checkbutton, only fullerenes with isolated pentagons are generated. An isolated pentagon is a pentagon which is surrounded only by hexagons.

Dual output: If this checkbutton is switched on, then every generated fullerene is dualized before it is read by *CaGe*, i.e. every vertex becomes a face and every face becomes a vertex. The result of such a dualization is a triangulation.

Spiral statistics: {spistat} If you switch on this checkbutton, then a statistics about the number of spirals in the generated fullerenes will be written to the logfile. In this case the minimum and the maximum number of vertices must be the same.

Symmetry statistics: {symstat} If you switch on this checkbutton, then a statistics about the symmetries in the generated fullerenes will be written to the logfile. There are 28 different symmetry groups which can occur in a fullerene, named e.g. S_i, C_i, C_{ih}...

Symmetry filter: If you click this button, another window appears where you can filter fullerenes which have a certain symmetry. There are 28 different symmetry groups. For every generated fullerene its symmetry group is determined. If the related checkbutton is switched on (i.e. the symmetry is *valid*), the fullerene will be sent to *CaGe* by *fullgen*, otherwise it will be dropped. By default, all symmetries are valid. If you change the default, the "Symmetry filter"–button will be highlighted. Note that the restriction to some symmetries will not speed up the generation process. Only a simple filter is invoked. Except the checkbuttons, there are three more buttons inside the "Symmetry filter for fullerenes"–window: By clicking "Clear all" you switch off all checkbuttons (after that, you should switch on at least one checkbutton), by clicking "Set all" you switch on all checkbuttons, by clicking "Done" you close the window. If you close the window, its contents remains valid. Closing the window is just good for decreasing the confusion arising by too many windows on the display.

Select cases: {case} The generation of fullerenes can be split into three disjoint parts (cases). By default, all cases are considered. If you restrict the generation to one case, the "Select cases"–button will be highlighted. By clicking "Done" you close the window. The selected case remains valid.

4.2 *Tubetype*

Tubetype is a generation program which generates tubes or fullerenes that can be split into two tubes. Such fullerenes are called tube–type fullerenes. Like fullerenes, tubes consist of only pentagons (exactly 6 pentagons) and hexagons. However, only inner vertices are adjacent to three other vertices. Tubes have an outer cycle where the vertices are alternately 2- and 3–valent, with two exceptions: There is one place on the outer cycle where two successive vertices are 2–valent and one place where two successive vertices are 3–valent. There are two different ways to measure the distance between these places (the following description is not precise, it gives only a clue):

Perimeter and shift: The perimeter is the length of the outer cycle. It is measured in hexagon units. The shift is the number of 3–valent vertices between the two places.

Offsets: The length of the outer cycle is given implicitly by giving two offsets which denote the number of 3–valent vertices between the two places. Two values are necessary because there are two paths on the cycle which connect the two places.

Basic options: By the radiobuttons named “tubes” and “fullerenes” you determine what kind of objects to be generated. If the “IPR”–checkbutton is switched on, then only objects with isolated pentagons are generated.

When a tube is generated, it can easily be extended to a tube with more vertices by adding hexagons to the outer cycle. One complete *ring* consisting of hexagons must be added. In this case the resulting outer cycle after adding the ring looks the same as before. So each tube has a number of hexagon rings called the *tube length*. The tube length can be chosen by the “Tube length”–slider. If the “default”–checkbutton is switched on, then the default tube length will be taken which depends on the perimeter and shift. For fullerenes, the tube length means the number of inner hexagon rings.

The “Perimeter”, “Shift”, “Offset 1” and “Offset 2”–sliders are used to choose the related parameters as described above. You should use either “Perimeter” and “Shift”, or “Offset 1” and “Offset 2”, depending on the kind of imagination you prefer. There are several interactions between the sliders. Note that not all values for “Offset 1” can be translated to “Perimeter” values since then the limit for “Perimeter” would have to be raised.
Output options: See section **??**.

4.3 *HCgen*

HCgen is a generation program which generates hydrocarbon structures. Each carbon atom has bonds to exactly three other atoms where these atoms are either carbon atoms or hydrogen atoms. Carbon atoms are always part of either pentagon or hexagon rings (compare section **??**).
Basic options: Choose the number of carbon atoms by using the slider “Number of C atoms”. Choose the number of hydrogen atoms by using the slider ”Number of H atoms”. Choose the exact number of pentagon rings by using the slider “Number of pentagons”. The three numbers interact so that not every combination is possible. For example, if the number of carbon atoms is 6 and the number of hydrogen atoms is 6, then the number of pentagons must be zero. Not every illegal combination can be prevented that easy. So it might happen that you can start the generation program and it will return immediately because it has found illegal parameters. In this case a small window appears which asks you to look at the standard output for details. The standard output is the text window where all generation programs print their information. The reason for the small window to appear is that *CaGe* has found no appropriate logfile. So *CaGe* assumes that no proper generation process has taken place.
Output options: See section **??**.
Extra options: The following options are options which are rarely used.

Isolated pentagons: If this checkbutton is switched on, only hydrocarbons with isolated pentagons are generated, i.e. no two pentagons share an atom.

Include H atoms: Exactly if this checkbutton is switched on, hydrogen atoms are included into the graphs. If hydrogen atoms are not included, then their positions can be uniquely determined by looking at atoms which have only bonds to two other atoms. The third bond is an implicit bond to a hydrogen atom. If hydrogen atoms are included, the size of the output becomes a bit larger and the computation of 2D and 3D embeddings takes more time. Besides, 2D output looks rather ugly.

Peri–condensed: If this checkbutton is switched on, no two carbon atoms which lie on the outer cycle must have bonds to each other except this bond (represented by an edge) is itself part of the outer cycle.

Maximum size of a gap: A gap is a sequence of two or more carbon atoms on the outer cycle of the hydrocarbon which do not have bonds to a hydrogen atom. The maximum size of such a gap can be determined by this slider.

4.4 *CPF*

CPF is a program which generates 3–regular planar maps with given face sizes. Interpreted as molecules, each vertex is a carbon atom. Each atom has bonds to exactly three other atoms. The face sizes are the sizes of the carbon rings. One special case is where only pentagons and hexagons are allowed. Those molecules are exactly the fullerenes (which can be generated more efficiently by using *fullgen*).

Basic options: Note that there are two windows which include basic options. In the well–known window including the output options you can choose the minimum and maximum number of vertices. The number of vertices can be selected by two sliders named "Minimal number of vertices" and "Maximal number of vertices". If the "Min=Max"–checkbutton is not switched on, then these two sliders can have different values.

In the other window you can choose the allowed face sizes. All allowed face sizes are listed inside the frame named "Face sizes to be included". For every face size there is a checkbutton named "Limits". If you switch on this checkbutton, then two sliders appear. With these sliders you determine how many faces with the related size must appear at least and at most in every generated map. Of course the minimum cannot be bigger than the maximum. If the "Limit"–checkbutton is not switched on, then no limits will be assumed. This means also that *CPF* might generate maps where the related face size will not appear at all.

In the frame named "Selected face size" there is a slider which you can use to choose a face size. If this face size is already included into the list of allowed face sizes, then you can reject it from the list by clicking the button "Discard face size". If the face size is not yet in the list of allowed face sizes, then you can include it by clicking the button "Include face size". In each case exactly one of the two buttons appears.

Output options: See section **??**.

Extra options: The following options are options which are rarely used. Some of them are so specific that you must know about the details of the generation program *CPF*. Such options are not explained in detail. Instead the related option which you would have to use for a direct call of *CPF* is provided. It is enclosed in brackets "{" and "}". For a further description of such an option see the original manual of *CPF*.

Use alternative ECC: {alt} If this checkbutton is switched on, then the algorithm for the generation is slightly changed. In most cases, this alternative algorithm will do worse (i.e. it needs more time and memory). But if both hexagons and another face size with more than 6 vertices are allowed without limits, then the alternative algorithm might do better.

Pathface maximal: {pathface_max} For normal use this option is completely unimportant.

Dual output: If this checkbutton is switched on, then every generated fullerene is dualized before it is read by *CaGe*, i.e. every vertex becomes a face and every face becomes a vertex. The result of such a dualization is a triangulation. If you use the dual output, you should restrict the generation to 3–connected maps (see extra option "Connectivity filter").

Face statistics: {facestat} If this checkbutton is switched on, then some additional information about the number of generated maps with certain face sizes is written into the logfile.

Patch statistics: {patchstat} If this checkbutton is switched on, then some additional information is written into the logfile.

Select cases and priority: {no_1, no_2, no_3, priority} The generation of 3–regular planar maps can be split into three disjoint parts (cases). By default, all cases are considered. If you click the "Select cases and priority"–button, another window appears where you can switch off cases. Furthermore, you can change the priority but this will not be explained here since for normal use the priority is completely unimportant. By clicking "Done" you close the window. The chosen cases and the priority remain valid. If you have changed the default values, the "Select cases and priority"–button will be highlighted.

Connectivity filter: {con} If you click this button another window appears where you can choose the connectivity of the generated maps. The connectivity is the minimum number of vertices which must be rejected to split a map into two or more parts. For chemical purposes, only 3–connected maps are interesting. Furthermore, the 2D and 3D representations of 1- or 2–connected maps look strange. By clicking "Done" you close the window. The chosen connectivities remain valid. If you have changed the default values, then the "Connectivity filter"–button will be highlighted. Note that restricting the connectivity will not speed up the generation. It is only a filter.

4.5 *Input file*

"Input file" is **not** a generation program. If you choose this option, then the graph data will be read from a file instead. However, this case can be treated like a generation process since it doesn't matter where the graphs come from.

There are several codes which are used to store graphs. Not every code is understood by *CaGe*. The codes which are understood by *CaGe* are exactly the codes which can be written by *CaGe*, namely *planar_code*, *writegraph2d* and *writegraph3d*[1]. If the input file contains a different code, then *CaGe* will do nonsense. If the input file contains one of the three codes mentioned, then *CaGe* will recognize automatically which code is contained in the file.
Input options:

Input filename: The name of the input file is necessary. You can type it into the related entry.

The following input options are important only if you wish to output embedded graphs (on the display or into a file). They determine where to get the embeddings from.

Use old embedding: If the input file contains embedding information, i.e. information about the coordinates of the vertices (atoms), then this information is used exactly if the "Use old embedding"–checkbutton is switched on. If the input file does not contain coordinates, then this checkbutton has no meaning.

Re–embed : If this checkbutton is switched on, then every graph will be re–embedded (using *spring* and/or *schlegel*) before it will be shown or written into a file.

If the input file does not contain embedding information, then *CaGe* will call the embedding programs even if the "Re–embed"–checkbutton is not switched on. This includes the case that the input file contains 2D information and you want to see 3D representation (or vice versa). If the input file contains embedding information and both checkbuttons are switched on, then the behaviour of *CaGe* is as follows:

- Every 2D embedding is determined using *schlegel* without concerning the old embedding.
- Every old 3D embedding is used to build the new 3D embedding upon. If the "Use old embedding"–checkbutton was not switched on, then *spring* would take a default 3D embedding to start with.

The 3D embedding can be divided into four parts a,b,c and d. The four related sliders ("Parameters for 3D re–embedding") can be used to determine how carefully each part will be computed. The value of b should be at least 10 times as high as a. The value of d should not be chosen too high since the computation time depends linearly on d^2. For details see the original manual of *spring*.
Output options: See section **??**.

5 Codes

In this section the codes which are read and written by *CaGe* are described. For a detailed description and an introduction to other codes see the original manual of *gconv*.

[1]See section **??** for a brief description

***writegraph2d*:** This is a Combinatorica format. It is ASCII, so it can be read by loading the related file into an editor. The file begins with one of the following headers:

```
>>writegraph2d<<
>>writegraph2d planar<<
```

Then the graph information follows. For each graph the information is given as follows: Vertices are numbered 1,...,n. For each vertex x there is one line in the file. The first number in the line is the vertex number x to which the following numbers are related, then two float coordinates follow. They contain the position of vertex x in the plane. After that, the numbers of those vertices which are adjacent to vertex x follow (the neighbours of x). After the last vertex there is a line that contains only a zero. Then the next graph follows. So more than one graph can be stored in one file.

If the order in which the vertex neighbours are listed indicates a planar embedding, then the header contains the parameter `planar` right after the code name. If the header does not contain the parameter `planar`, this does not automatically mean that the codes are no planar embeddings.

Here is an example file (containing the two fullerenes with 28 vertices):

```
>>writegraph2d planar<<

1 500 866.025 12 13 2
2 1000 0 1 3 23
3 500 -866.025 2 20 4
4 327.062 -561.364 3 5 24
5 18.681 -505.199 4 19 6
6 108.393 -344.286 5 7 25
7 -88.795 -157.214 6 17 8
8 5.716 2.986 7 9 26
9 -89.192 165.809 8 16 10
10 102.363 346.616 9 11 27
11 19.263 511.485 10 14 12
12 327.451 561.554 11 1 28
13 -500 866.025 1 14 21
14 -305.74 552.598 13 11 15
15 -447.152 241.398 14 16 22
16 -344.259 88.378 15 9 17
17 -342.855 -75.43 16 7 18
18 -443.29 -234.616 17 19 22
19 -308.899 -538.802 18 5 20
20 -500 -866.025 19 3 21
21 -1000 0 20 13 22
22 -644.641 5.733 21 15 18
23 634.271 -6.36 2 24 28
24 436.421 -267.701 23 4 25
25 255.983 -261.512 24 6 26
26 189.348 4.006 25 8 27
27 258.3 261.219 26 10 28
28 430.465 273.377 27 12 23
0

1 500 866.025 12 13 2
```

```
2 1000 0 1 3 21
3 500 -866.025 2 19 4
4 321.919 -581.442 3 5 22
5 3.725 -536.829 4 18 6
6 19.874 -385.553 5 7 23
7 -260.627 -228.273 6 17 8
8 -141.191 41.533 7 9 25
9 -285.077 262.325 8 15 10
10 -42.105 428.45 9 11 26
11 -29.18 564.25 10 14 12
12 271.706 615.179 11 1 27
13 -500 866.025 1 14 20
14 -334.507 573.974 13 11 15
15 -447.472 277.783 14 9 16
16 -636.953 0.808 15 17 20
17 -432.968 -266.301 16 7 18
18 -314.325 -567.202 17 5 19
19 -500 -866.025 18 3 20
20 -1000 0 19 13 16
21 662.252 -73.473 2 22 28
22 458.915 -309.832 21 4 23
23 315.913 -254.598 22 6 24
24 355.774 24.301 23 25 28
25 122.788 118.125 24 8 26
26 197.526 323.948 25 10 27
27 336.98 354.08 26 12 28
28 467.657 117.43 27 21 24
0
```

If a graph is not embedded before storing it, then all coordinates are zero.

***writegraph3d*:** See *writegraph2d*, but with three coordinates for each vertex and with the appropriate code name inside the header (`writegraph3d`).

***planar_code*:** This code is binary and so it cannot be read on a screen. It does not provide coordinates, so the adjacency information of graphs is all what can be stored using this code. The file begins with a header which is one of the following:

```
>>planar_code<<
>>planar_code le<<
>>planar_code be<<
```

Here “le/be” stands for “little endian”/“big endian” (this will not be discussed here). After the header, for each graph the following information is given: First the number of vertices of the graph appears. Vertices are numbered 1,...,n. Let’s assume that we have already embedded the graph into the plane. First the neighbours of vertex 1 are listed in clockwise direction, followed by a zero. Then the same continues with vertex 2,3,...,n. After the last zero, the next graph follows.

The above example file would look like this (where the numbers are **binary**, not readable, and the commas are meta symbols which are not contained in the file): `>>planar_code<< 28, 12 13 2 0, 1 3 23 0, 2 20 4 0, 3 5 24 0, 4 19 6 0, 5 7 25 0, 6 17 8 0, 7 9 26 0, 8 16 10 0, 9 11 27 0, 10 14 12 0, 11 1 28 0, 1 14 21 0, 13 11 15 0, 14 16 22 0, 15 9 17 0, 16 7 18 0, 17 19 22 0, 18 5 20 0, 19 3 21 0, 20 13 22 0, 21 15 18 0, 2 24 28 0, 23 4 25`

```
0, 24 6 26 0, 25 8 27 0, 26 10 28 0, 27 12 23 0, 28, 12 13 2 0, 1 3 21 0, 2 19 4 0,
3 5 22 0, 4 18 6 0, 5 7 23 0, 6 17 8 0, 7 9 25 0, 8 15 10 0, 9 11 26 0, 10 14 12 0,
11 1 27 0, 1 14 20 0, 13 11 15 0, 14 9 16 0, 15 17 20 0, 16 7 18 0, 17 5 19 0, 18 3
20 0, 19 13 16 0, 2 22 28 0, 21 4 23 0, 22 6 24 0, 23 25 28 0, 24 8 26 0, 25 10 27
0, 26 12 28 0, 27 21 24 0
```

Normally the code entries are of type *unsigned char* (1–byte size, i.e. numbers between 0 and 255). But if the number of vertices for one graph is higher than 252, then the code for this graph begins with a zero (unsigned char) and then each following entry is of the type *unsigned short* (2 bytes). In this case it makes a difference whether the entry is stored in little endian or big endian style (the endian determines whether a number like 19 is stored by storing the 1 or the 9 first). The output style that is produced **always** depends on the definition of the internal constant ENDIAN_OUT (see section **??**) which value is usually consistent to the endian of the computer. The input style that is assumed depends on the definition of the constant ENDIAN_IN. But if the input file begins with a header that contains a "be" or "le"–parameter, then the input style is big endian or little endian, respectively. If the file contains no graphs with more than 252 vertices, then it is not important if the style is little endian or big endian.

6 Internal constants

In this section internal constants are discussed. The behaviour of *CaGe* depends on the values of some internal constants. Usually there is no need to change the values of these constants. Changing the values is done by editing the source program `CaGe.c` and then recompiling the source code (using the command `make` — see `README` file). This means that you need to have basic knowledge about the programming language *C*. The following constants are listed at the top of the source code (called *public definitions*) or after the first big comment page (called *internal definitions*):

ENDIAN_OUT/ENDIAN_IN: The value depends on the machine the source code is compiled on. SGI machines are known to be big endian while most other machines are little endian. If the definition does not fit to your computer, you do not need to change it if other programs which interact with *CaGe* are able to handle both little and big endian. Furthermore, you do not need to change it if the endian is unimportant (see section **??**).

MAXN: If an input graph has more vertices than indicated by this number, then *CaGe* will do nonsense. So if you plan to read in bigger graphs, it is necessary to increase this number.

MAXENTRIES: This number indicates the maximum number of code entries for one input graph, using *planar_code*. For an n–valent vertex $n + 1$ entries are needed (the numbers of the n adjacent vertices and a zero at the end). If an input graph requires more entries, then *CaGe* will do nonsense. However, this will not occur if a graph with maximal MAXN vertices or its dual will be read from one of the four generation programs *fullgen*, *tubetype*, *HCgen* or *CPF*.

NL: This is the string which is assumed to be the newline symbol in all ASCII–files which are read or written.

Figure 1: Main program loop

Figure 2: Data flow during graph generation