CROATICA CHEMICA ACTA CCACAA **70** (3) 873–882 (1997)

ISSN–0011-1643
CCA–2461

Original Scientific Paper

Compiling an Adjacency Matrix of an Arbitrary Structure

E. C. Kirby

Resource Use Institute, 14 Lower Oakfield, Pitlochry, Perthshire PH16 5DS, Scotland UK.

Received August 19, 1996; revised April 28, 1997; accepted May 5, 1997

An experimental project on which work continues intermittently is the development of an interactive Personal-Computer program to assist in the capture of connectivity information about an arbitrary chemical graph of moderate size. (By *arbitrary* we mean one that does not fall, or is not recognized as falling, into a class which has useful symmetries or easily applicable analytic formulae to provide short cuts.) The approach is to deal with a graphics-screen image using a small set of tools, in the hope of being able to obtain a connection table faster and with a smaller risk of errors than by manual methods alone. It is applicable both to planar structures and to non-planar ones that have been cut to give a 2D representation.

INTRODUCTION

A chemical graph with N vertices is defined and represented as a list of connections between pairs of its vertices, so that if, as is common practice, the vertices are labelled with integers within the range 1-N, assigned in a unique although sometimes arbitrary manner, then a concise form is an $N \times 3$ integer matrix (for a regular graph of degree-3). Here, in the i^{th} row will be the three entries j_1, j_2, j_3, these being labels of the vertices adjacent to vertex i. The same information can be given by restricting entries j_x to those cases where $i < j_x$, but this saves only a little computer memory and makes searching more difficult. This table can obviously be entered manually, edge by edge, from a keyboard, but various short cuts may be deployed. One technique we have used extensively[1] is to automatically set up the table for a linear chain of the same size, and then make or break connections as nec-

essary to derive the required structure. Such a procedure works best if the structure of interest is path-Hamiltonian.

The adjacency matrix is a square matrix $\boldsymbol{A}$, where each element $a_{i,j}$ is 1 if vertex$_i$ and vertex$_j$ are connected (*i.e.* adjacent) or 0 otherwise. (Actually $n!$ different but equivalent matrices for any graph are possible, corresponding to different labellings.) It plays a fundamantal role in chemical graph theory; for example, its eigenvalues may be taken as energy levels within Hückel Theory; it is used for some structure encoding techniques; and numerous so-called topological indices relating chemico-physical properties to connectivity are based upon it.[2]

An active computer screen showing an image consists of a rectangular array of pixels – small areas of the screen which may be 'on' or 'off', and which if 'on' are emitting light of a particular hue (produced by combining primary colours from the electron guns in suitable proportion). A number of different physical standards are in use, dating from the various stages of commercial development and hardware evolution. At the present time a typical (not the best) display device for a popular personal computer (PC) is the 'VGA' graphics screen. In graphics mode this shows 640 × 480 pixels, each one of which may in principle show one of many colours, although for present purposes a palette of 16 colours is more than sufficient. Colour is not essential to the depiction of a chemical graph, although, as will be seen, it is very useful for aiding machine recognition if constituent parts of diagrams are rendered in different colours. (Note that we are *not* here making use of the action of »colouring« in the graph-theoretical sense to partition vertices or edges into two classes where no two members of the same class are adjacent.[3])

So far, so good, and, apart from the existence of graphics screen standards other than VGA (the usual difference from a software point of view being in the number of pixels used), the parameters needed to describe an image that is being shown on a computer screen are relatively straightforward. If we turn our attention to *stored* images on diskette, however, the picture is far more complicated. Because of competing commercial standards, and the need for data compression (graphics data are notoriously bulky, and sparse in terms of significant bits), a variety of encoding formats has arisen. Among the more popular image file types in use are TIF, GIF, PCX, JIPG, the acronyms (unimportant here) indicating the type of coding used. Many of these are further subdivided into different version numbers, as different compression algorithms have been developed.

There is a broad division between methods which simply store the array of pixel values (or a compressed set from which the array can be recovered with a decoding algorithm), essential for the more complex photographic type of image – and those which store a set of vectors and can be used for the simpler type of line drawing programs. In principle, this latter kind

could obviously be extended to encompass chemical graph information. However, besides the technical and legal difficulties of understanding and modifying an existing drawing program, it is highly desirable that any kind of image, including a free-hand drawing scanned into a computer, should be usable. It was therefore decided at an early stage that the program to be developed should use only an image already displayed on screen, from whatever source, as its input.

THE PROGRAM STRUCTURE IN SUMMARY

The ideal to be aimed at is a machine algorithm that will accept a drawing such as chemists commonly draw freehand, consisting of a line drawing that does not even show atoms or vertices; these being implicitly assumed to be wherever a line ends or two or more lines meet. Such a computer program should place vertices correctly, label them, list the connections and print out or file a connection table.

Not surprisingly, any rigid insistence of these goals is unrealistic if the program is to accept a useful variety of input formats. Consequently the strategy has been to try to create a flexible program that mostly provides more than one way to carry out each operation. Fully interactive or partially automatic modes made be chosen, depending on the quality of the graphics line drawing available. The simplest procedures are available when the drawing consists of lines that are all vertical, horizontal, or at 45° to these, and of one pixel width.

In operation the computer process followed is:

1. Provide a blank VGA graphics screen (»Screen 12«) with 640 × 460 pixels available for the chemical graph. The whole screen actually has 640 × 480 pixels, but for this program a top horizontal strip 20 pixels deep is reserved in order to display one line of text. All messages and instructions are displayed here.

2. Load a screen image of a chemical graph, as a white line drawing.

3. Place vertices where any line ends, or where two or more lines meet. In the most favourable cases this can be done automatically; in others vertices are placed under Mouse control. As each vertex is encountered and marked on-screen, it is assigned a label and is entered in the connection table.

4. Search for all pairs of connected vertices. For each connection found, recolour the connection line to reflect this, and update the connection table.

5. Where necessary, mark pairs of vertices on screen as being identical, and update the connection table accordingly. (This is necessary for certain 2D representations of 3-dimensional structures.)

6. Save a copy of the modified image together with its connection table.

In practice the program consists of about fifty small procedures, grouped together for operational purposes into eleven specialised toolkits which may be selected in any order. These are shown in Figure 1, and some details noted in the following section.

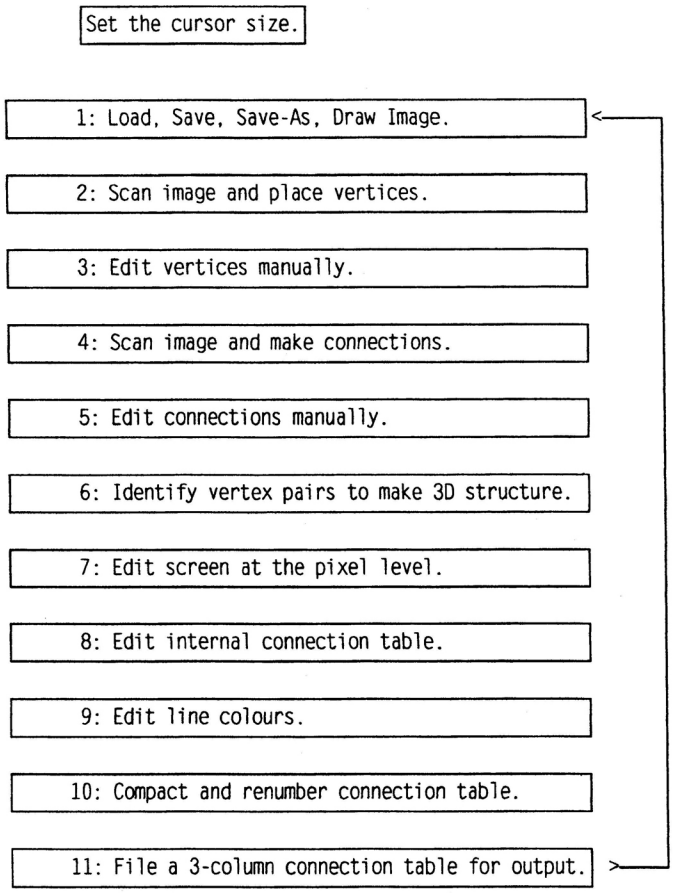

Figure 1. The menu of main program options available.

THE MENU OF PROGRAM TOOLS:

The main operations that can be invoked from the program's main menu are summarised below, the numbering and legends corresponding to those shown in Figure 1.

1. Load, Save, Save-As, Draw Image

Rudimentary line drawing tools are provided, mainly for effecting small repairs, but more commonly a previously drawn or scanned-image is displayed, using a program (such as WordStar's Inset) which can be loaded as a Terminate and Stay Resident (TSR) item. The image is then captured (by the SAVE-AS command and stored, for the purposes of this program, in five files. These are *filename*.G01 – *filename*.G04, which contain, respectively, blue, green, red and white component images; and *filename*.DAT which contains the raw connection table. This is more comprehensive than the final connection table to be exported (see 8. and 11. below).

2. Scan image and place vertices

This unit allows the image to be scanned vertically, horizontally, diagonally from top left to bottom right, and diagonally from bottom left to top right. Pixels are examined line by line across the screen. Each one is labelled 1 if it has an attribute equal to the draw colour, or 0 otherwise. Any sequence 1-0-1 in a line is assumed to be at a conjunction of two lines (see Figure 3), and to be associated with a vertex. So it is marked on screen, and an entry made in the connection table. This routine is preceded by an option to restrict the scan to a smaller section of the screen, but with the equipment used (see Appendix), it was generally found that use of this option did not save time.

The automatic method works well for simple and smooth line drawings, but is often useless for scanned images where lines may be thicker and have indentations. In such cases, extra spurious vertices may be assigned. When this happens, a vertex can be erased, but more often it is simpler to abandon the image and start again with a fresh copy, avoiding one or more of the automated sweeps and going straight to manual insertion.

3. Edit vertices manually

This allows precise positioning of vertices using a Mouse, with or without the usual screen arrow cursor controls. The cursor consists of a square box with a central cross-hair, and the program will refuse to assign a new vertex if any other exists within the area displayed. It's default size is 25×25 pixels, but this can be changed at the start of the program. A vertex is marked on screen as a 5×5 square of magenta coloured pixels. The central pixel is the one whose screen coordinates are tracked and used for identification.

4. Scan image and make connections

This is another automatic sweep (*cf.* 2, above), although in this case only vertically from top to bottom. A sequence of pixel lines one cursor box width apart are examined. Any pixels with the attribute of the drawing colour

which are encountered are assumed to belong to an edge, and a search is made above and below. If this results in contact with two vertices, then these are located in the connection table and connection entries made. Finally the drawn edge is painted in the 'edge colour' (cyan) so that the search will not be repeated, and the rest of the pixel line examined. False positives are fairly rare, but some existing connections are missed, for example because of an unnoticed gap in the drawing, or a bug in the edge-search routine. Such omissions must be completed manually.

5. Edit connections manually

Here again, vertices are located with the Mouse, and a sequence of clicks will instruct the program to enter a connection. When the mouse is swept over an image and encloses a vertex, it normally displays extra temporary lines, and a statement at the top of the screen, to confirm which connections are recorded for that vertex.

6. Identify vertex pairs to make 3D structure

This routine is most typically used to prepare the connection table for a toroidal fullerene,[4] when a planar structure is treated as if it is folded over and opposite edges fused to form first a cylinder, and then a torus. In the simplest cases the structure is a near rectangle. This means that vertices on the perimeter must be grouped into either pairs or triads which each represent a single 'real' vertex. This identification is again effected with Mouse clicks. Either pairs of individual vertices, or pairs of multi-vertex perimeter sequences may be matched.

The selection of vertices to be identified, and the order in which processing is done has not been automated in any way, and does require some care. As a precaution against error, the program checks that every vertex is of degree-3 before it will allow a completed identifying operation to become permanent.

7. Edit screen at the pixel level

This is option enables a part of the image to be examined in fine detail, and repaired or modified if necessary. A short sequence of pixel attributes straddling the cursor position is displayed at the top of the screen, and these can be changed one by one.

8. Edit the internal connection table

This gives access to the internal connection table by displaying it one line at a time, at the top of the screen. When this table exists, it is filed as *filespec*.DAT whenever a SAVE or SAVE AS instruction is given under option 1.

The table itself contains seven columns of integers, and as many rows as there are vertices displayed on screen. The seven columns are:

(1): A sequential label number, assigned automatically as vertices are marked. A value of –1 indicates that in the graph itself (rather than in this particular image) it is identical to another vertex; see (4) below.

(2) and (3): These give the screen coordinates (pixel count from the left and from the top, respectively) of the central pixel of a marked vertex. This pair of numbers is what is used by the program to search for and identify an existing vertex.

(4): This has a non-zero value only if this vertex has already been marked as being identical to another one (see Option 6 above). In such a case the value refers to the label (column 1) of the corresponding partner, and its own first column is set to a value of –1.

(5),(6) and (7): These give the labels of the adjacent vertices. If all are zero then either the edge assignment is incomplete, or the vertex is identical to another – see (4) above – and has a label in the first column of –1.

9. Edit line colours

This is used to change the colour of connecting lines in the image, for example, from draw-colour (white) to edge-colour (cyan) after making a manual connection under option 5 above, or vice versa. It has no effect on the internal connection table. The same thing is done automatically under Option 4 above.

The reason for using different colours for a drawn connection and an *installed* connection is (i) to provide a visual check on progress, and (ii) to enable the program to recognize and ignore connections that have already been made.

At the time of writing, this routine is not fully bug-free, and occasionally fails to follow a curved line to its end where it must abut a vertex. High level language 'FILL' or 'PAINT' commands, which change the colour of all contiguous pixels of existing like colour, are quite common, but we have not found an easy way to access their reporting flags.

10. Compact and renumber connection table

This routine is only needed when vertices have been deleted, or identified with others. It could be made subject to automatic calls under other options, but was separated so that the labelling remains constant during processing until a conscious decision is made to consolidate it.

A call to this procedure simply renumbers consecutively the set of 'real' vertices, (which has fewer than the original number if deletions have been made, and fewer than are on-screen if identifications have been made), and revises the whole table accordingly.

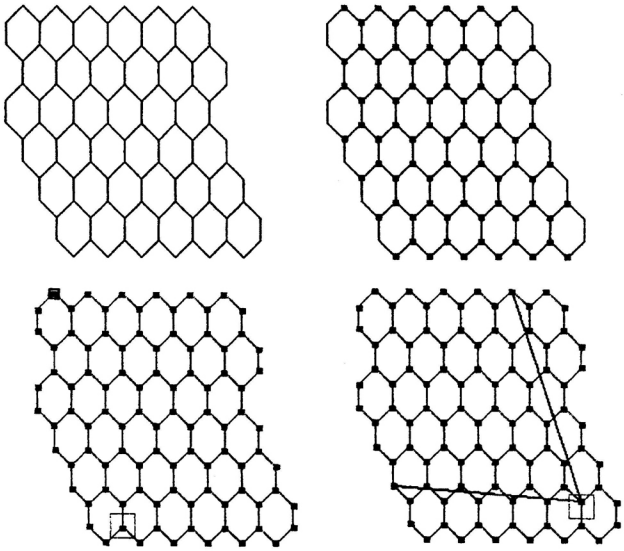

Figure 2. Stages in the processing of a line drawing: (a) image as input from a drawing program; (b) image after an automatic vertical sweep to place vertices; (c) image after manual completion of vertices, automatic recording of adjacencies and manual marking of identities (placing the cursor over one screen vertex of a pair brings up another smaller one to show its partner vertex that has the same graph label; and (d) the same stage with the cursor at another position, which again confirms the adjacencies of one vertex, and that this part of the required folding over of the structure has been done.

11. File a 3-column connection table for output

This prints or files (in ASCII alphabetic characters) a subset of the internal connection table, namely a 3-column matrix giving the connections for each vertex with non-zero connections.

The file can be passed as input to other programs.

CONCLUSION

This paper has given a short description of the main features of the program at this time. This is an ongoing development project rather than a finished and polished piece of software. Any specialised piece of software of significant size and negligible commercial value requires a substantial investment of programming time and effort, which is made in hopes of a continuing 'dividend' in the form of either previously inaccessible calculation results, or a saving of routine work, or a reduction of error risk. Only the last two are

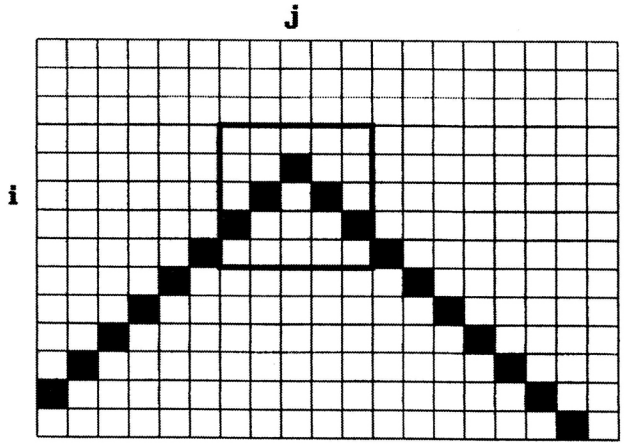

Figure 3. A magnified view of a small area of the screen where the tip of a hexagon has been drawn (each square represents one screen pixel). The heavy lined square shows the area that would be coloured as a vertex during a vertical sweep of the vertex-finding routine. The screen coordinates of the central pixel (i,j) are stored and the pair is used as the unique identifier of this vertex.

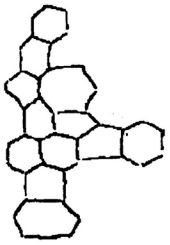

Figure 4. A typical result from scanning of a very rough pencil drawing. Most vertices would need to be placed manually for this example, followed by a check to see that all intended connections had been properly completed.

applicable to this program, and it will be some time before it 'pays its way' in terms of effort saved. Nevertheless, work continues, as resources permit, on gradual improvements.

APPENDIX

The system parameters used (not necessarily all system requirements) are: PC with a 100 MHz 486 CPU, 20 MB of RAM, VGA screen and Track ball. The operating system is DOS 6.22 (Use of Windows has, so far, been

avoided, in order to maximise available speed.), and program code was written in PowerBasic Version 3.20 (PowerBasic Inc, Carmel, CA, USA). At the time of writing the development program has about 3370 code statements, and occupies about 115 KB of disk space in its compiled 'EXE' version.

REFERENCES

1. (a) E. C. Kirby, in: *Graph Theory and Topology in Chemistry*, R. B. King and D. H. Rouvray (Eds.), University of Georgia, Athens, Georgia, USA – *Studies in Physical and Theoretical Chemistry* **51** (1987) 529, (Elsevier Science Publishers B. V., Amsterdam); (b) E. C. Kirby, *J. Math. Chem.* **1** (1987) 175.
2. See for example (a) Nenad Trinajstić *Chemical Graph Theory*, 2nd Edition, CRC Press, Boca Raton, Florida, 1992; (b) *Computational Graph Theory* Dennis H. Rouvray (Ed.), Nova Science Publishers, New York, 1990; (c) Sabljić and Horvatić, *J. Chem. Inf. Comput. Sci.*, **33** (1993) 292–295.
3. Robin J. Wilson and John J. Watkins, *Graphs An Introductory Approach*, John Wiley & Sons, New York, Chichester, 1990.
4. (a) E. C. Kirby, R. B. Mallion, and P. Pollak, *J. Chem. Soc. Faraday Trans.* **89** (1993) 1945; (b) E. C. Kirby, *Croat. Chem. Acta* **66** (1993) 13; (c) E. C. Kirby, *Fullerene Science and Technology* **2** (1994) 395.

SAŽETAK

Izračunavanje matrice susjedstva proizvoljne strukture

E. C. Kirby

Pokusni projekt na kojem se radi povremeno, jest razvoj interaktivnog programa za osobno računalo koji bi olakšao dobivanje informacije o povezanosti u proizvoljnom kemijskom grafu srednje veličine (pod »proizvoljnim grafom« podrazumijeva se graf koji ne spada, ili za koji se može smatrati da ne spada, u klasu koja ima upotrebljive simetrije ili lako primjenljive analitičke formule koje omogućuju pojednostavljenje). Pristup se temelji na upotrebi grafičkog prikaza uz primjenu malog skupa alata, u nadi da će se tablica povezanosti dobiti brže i uz smanjenu opasnost od pogrešaka nego u slučaju samo ručnih metoda. Pristup se može primijeniti kako na planarne tako i na neplanarne strukture koje su razrezane kako bi se svele na dvodimenzijski prikaz.