communications in mathematical
match
and in computer chemistry

no. 41, March 2000

ISSN 0340-6253

MATCDY (41) 79-97 (2000)

Exhaustive Isomer Generation using the Genetic Algorithm

Christopher LE BRET
Dept. of Research, PMSI, 52 rue Mouffetard, F-75005 Paris
(33 1) 45 35 87 99, pmsi@altern.com

Abstract

Building upon previous work, software that generates isomers from real-life constraints (large fragments, hybridisation states, etc.) is presented. The software is based on the genetic algorithm. Isomer generation seems exhaustive. Numerical analysis objections against floating-point approximations to real number-based topological indices are delivered. Also described is a novel, integer-based, efficient way to discriminate between isomers by computing their "fingerprints". This also allows determination of topological equivalence classes.

Results are compared with those from DENDRAL, MOLGEN+, AEGIS, CHEMICS, Hendrickson's SKEL_GEN and ESESOC. Omissions in DENDRAL's isomer lists are observed and illustrated. Comparison with SKEL_GEN and MOLGEN show small, explained discrepancies with Galvastructures for some very unsaturated molecules; perfect agreement is obtained in all other cases.

The software used is available freely, to let readers replicate the results and test the method on the information of their choice.

Keywords

Structural elucidation, connectivity matrices, topological equivalence classes, isomer generation, genetic algorithm, GalvaStructures.

1. The Problem

In a previous paper (Ref. 1), I had described a method (and its associated software, GalvaStructures) for reconstituting correct connectivity matrices from partial information, to wit, fragment lists or atom numbers. It seems a good thing to use the genetic algorithm's peculiar competence in managing any mixture of constraints through the use of a "fitness function", rather than using the graph theory formalism and finding the symmetry group. The present paper builds on these first results in a few directions.

First, thanks to the flexibility of the genetic algorithm's fitness computation, isomers satisfying **any mixture of criteria**, from the very lax ("find all isomers having *N* carbons and any number of hydrogens") to the very strict (complete the given partially-filled connectivity matrix, with so many Hs and specified hybridisation states) can be reconstituted. It is now possible to complete partially-filled connectivity matrices, containing for instance fragments and hanging bond specifications deduced from ^{13}C NMR spectra, making the software close to actual, practical lab use. Practical experimentation showed us that reassembling fragments is fraught with difficulties, even for an experienced practitioner. This is a thankless, uncreative task, well worth automating.

Secondly, we were faced with the problem of **many connectivity matrices actually describing the same molecule**. This abundance of duplicates made the task of checking isomers, as well as comparing to other generators' outputs, very painful. There has to be a "preferred" form among all those possible, the "canonical matrix". Producing this canonical matrix may involve working on the matrix itself, or finding some "canonical numbering" for the atoms.
Of course the problem of canonical matrices is not new, and not easy either. It is equivalent to that of unambiguously ordering atoms in a molecule, since once a "canonical labelling" has been obtained, one only has to rewrite the connectivity matrix with the atoms in the canonical order.
After a careful review of the literature, and convinced of the difficulty of the problem as it is traditionally posed, we ended up completely avoiding working on matrices and instead generated unique "fingerprints" for the connectivity matrices, fingerprints for two connectivity matrices being equal if & only if they describe identical molecules. These fingerprints play the same role as molecular topological indices while avoiding the pitfalls of floating-point calculations. **There does not seem to be any degeneracy.** Encouraged by this partial success, we then increased our ambitions from finding a couple of isomers to exhaustively generating them.

Thirdly, various improvements were made to increase the efficiency of the software. **Compact encoding** was implemented in addition to "straight" bonding matrix encoding, to shorten the genotype when large molecules are concerned. Our encoding differs from Hibbert's "bond tuples" (Ref. 2) in that we explicitly encode the number of bonds of each type in the tuple instead of repeating the tuple as many times as the order of the bond; this makes decoding quicker & simpler. GalvaStructures automatically chooses whichever encoding is shorter.

2. Floating-point topological indices : a numerical analysis viewpoint

In a nutshell, topological indices are real numbers used in order to condense the information contained in a whole matrix (local information for an adjacency matrix, global information for a connectivity matrix). The transformation of the matrix into a single number is performed using a mathematical function or an algorithm-like series of steps.
There can be two aims to this. First, we may want that the size of the difference between two indices reflect that of the differences in their associated structures. Here the numerical value has meaning.
In the other and more common use, we are not interested in the actual value produced, but we

want each different structure to **map to a different real number** for indexing or discrimination purposes. Here we are concerned only with that latter aspect.

With all due respect to my distinguished predecessors, there are a few fundamental problems associated with numerical topological indices.

Topological indices and hashing

This problem of transforming a large amount of data into a small one while ensuring a one-to-one mapping is well known in computer science under the name of **hashing**. There is a small formal difference in that programmers read the number generated as a large integer and use it as an index into a database. But the need for a unique mapping is the same, and both types of numbers are represented in computer memory as series of bits.

It is surprising to notice, when reading the chemical literature, the **complete absence** of the term **hashing**, or of any reference to standard algorithm books - most of them have a whole chapter on the subject. This is a pity because the first thing they mention is that "no hash function is perfect" (Ref. 3). "**Collision resolution processes**" (actually, second-level hashing) **always** have to be introduced to resolve the **unavoidable** cases where two distinct records map to the same index.

As if this were not enough, computer-style «real numbers» also pose problems of their own.

Real numbers vs. floating-point numbers

Mathematical, continuous **real numbers are not the same as their discrete computer approximations**, floating-point numbers. Floating-point representations of real numbers (as finite-size mantissa and exponent, plus a sign bit) pose special problems. The apparent confusion between real numbers & their floating-point computer representations is quite surprising to any programmer with a smattering of **numerical analysis**. This is not uncommon and resembles the erroneous distinction often made between "binary encoding" and "real encoding" in the genetic algorithm world - both end up encoded as bit strings anyway.

First, it is a well-known rule among numerical programmers to **never test floating-point numbers for equality**. Any calculation generates rounding-off errors, and index computation procedures risk not yielding the exact same results according to the order in which calculations were carried out, the brand and model of the processor, even compilation options. "A computer result always being the (sometimes very bad) approximation of a real number, equality of computer images does not imply equality of the numbers themselves, and conversely. Thus, **test `x == y`, if one is not working with integers, is absurd on a computer**" (Ref. 4).

Also, the accumulation of operations has the consequence of increasing loss of precision : "pretty much any arithmetic operation among floating numbers should be thought of as introducing an additional fractional error of at least ε_m" (Ref. 5). ε_m, the "machine accuracy", is the smallest floating-point amount such as $1 + \varepsilon_m$ is computationally different from 1. It equals around $2.2\ 10^{-16}$ for double precision numbers (coded on 64 bits), and is defined as `DBL_EPSILON` in the C programming language, in standard header `float.h`.

Liberal use of square roots, whose values are often computed by the floating-point processor by summing a slowly-converging series, compounds the problem as their precision is generally much less than optimal, sometimes as little as 12 significant bits on older computers.

The numerical analysts's dilemma

Even if the procedure is sound as far as the order of calculations is concerned, we are **falling between two stools** : either we use short floating-point numbers at the risk of not discriminating between two similar molecules because their associated indices differ only at a too-remote decimal place that gets rounded off, or we use very long representations, at the risk of rounding procedures and calculation order making different two numbers that should have been equal in

a "perfect" mathematical calculation.
Only one author (Ref. 6) broaches the subject of the number of "useful digits" necessary. Using IEEE-754 double precision numbers (52-bit significand, 11-bit exponent, 1-bit sign), there cannot be more than 15 significant digits at best. This writer's experience is to perform calculations using double precision, and then cast the results to single precision (32 bits) before comparing them. There are then 8 significant digits only. This is similar to the operation of the computer's floating-point processor : the internal format for computation of exponentials and such is 80 bits during intermediary computations, and the result is truncated to the regular 64-bit format only in the end.

The only "safe" computer representations are those involving only **integers**.

Indices such as Hu & Xu's (Ref. 7) are claimed to never give the same result for two different molecules, but it can be argued that they will not even give the same result for two different connectivity matrices describing the same molecule, or on a computer with a longer or shorter floating-point size. On top of this, a simple, quick numerical analysis performed on their algorithm indicates that, in using a formula like
weight = Term1 + Term2 x 10^{-j},
where Term1 and Term2 are of the order of unity, beyond atom rank j = 16, the contribution of Term2 x 10^{-j} is smaller than ε_m, and therefore **computationally zero**. Such a procedure may then not perform any better than BID (Ref. 8) or SID (Ref. 9) indices, that were at least valid up to rank 18 or 20.

Moreover, the use of 10^{-j}, or 0.1^j, lends itself to criticism as 0.1 cannot be exactly represented in the binary system (it is 0.0001100110011...).

All this may or may not be a problem in practice, but it would be reassuring to be able to think that the authors of any procedure are at least aware of the existence of the problem. **"Numerical methods without some determination of the errors involved are of questionable usefulness"** (Ref. 10).

Also and more fundamentally, it seems quite wrong to claim that any molecule can be described by a fixed-size number. One cannot fit **an unlimited amount of descriptive information into 64 bits**. Unambiguous description has to be similar to the IUPAC naming scheme : the more complex the molecule, the longer the name or the molecular descriptor.

3. Unambiguous "fingerprints" and topological equivalence

The fingerprints I propose are an ordered list of integer values describing as completely as possible the environment of each atom in the molecule. Information-reducing procedures such as replacing a list of values by their sum, as well as arbitrary, *ad hoc* coefficients, were avoided.

The method will be generally exposed, with a practical example following. It is independent from the rest of the software and could be **coupled with any structure generator**, preferably as a sieving stage for the output of a (possibly crude) systematic generator.

As has been already observed, a truly unambiguous atom environment descriptor must take into account all of the atom's environment, **up to the extremities of the molecule**. I followed Carhart's (Ref. 11) good advice to "carry out a full atom-by-atom, bond-by-bond comparison of the total topological environments of atoms being compared".

Each atom's description has four parts :

- description of the atom : nature, hybridisation state, number of first neighbours, rank of most distant neighbour, rank of most distant bond.

- ordered list of its first neighbours' descriptions (nature, hybridisation state, number of first neighbours and rank, for each neighbour),
- concentric list of the bonds as seen from the central atom, and
- ordered list of the individual fragments.

The ordering we mention here is by increasing rank as seen from the atom under consideration.

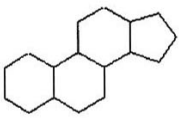
Each fragment is described by the quadruplet (class index of atom 1, class index of atom 2, bond rank, bond type). Atom 1 and Atom 2 are ordered so that atom 1 always has the smallest class index. The concentric list of bonds mentioned above is a list of (number of single bonds, number of aromatic bonds, number of double bonds, number of triple bonds) by increasing bond rank.

Atom ranks are the usual ones : 1 for the central atom, 2 for its first neighbours, etc.

Bond ranks are designed as follows : rank 0 describes bonds between the central atom and the first neighbours, rank 1 describes bonds between first neighbours if any (if there are such bonds, this means the atom being considered belongs to a 3-cycle), rank 2 describes bonds between first neighbours and second neighbours, rank 3 describes bonds between second neighbours if any (if there are such bonds, this means the atom being considered belongs to a 5-cycle), etc.
Any difference in bond location will make a local difference from the viewpoint of the atoms close to it : there will therefore be a difference in these atoms' environments. Practical results indeed reveal no degeneracy, except in one case analysed farther.

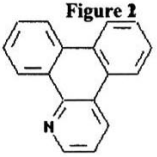
Each atom's environment description will have length proportional to N-1 (in addition to the central atom's description), N being the number of atoms in the molecule. There will be N such descriptions, so the total length of the molecule's fingerprint will be of the order of N squared. This is computationally unsatisfying, but looks more robust from a theoretical viewpoint. Advances in computer memory size & processing power make this **a small price to pay** if true nondegeneracy is actually achieved.

All complete descriptions are then sorted with respect to each other, so we end up with an unambiguous descriptions list, no matter in what order we considered each atom. The order in which atoms are sorted is used to attribute class indices to them. The comparison function used for sorting **compares everything** in the description. To the extent that the environment description is complete, two atoms comparing as equal are **chemically equivalent**.



Recalcitrant compounds such as Shelley & Munk's (Ref. 12) 1 and 2 (Figures 1 & 2) as well as all others in this paper are correctly dealt with : GalvaStructures does find that all atoms in these molecules are nonequivalent. We therefore seem to have, *en passant*, stumbled across a method for reliably identifying topologically equivalent atoms. But for us, this is merely a method of renumbering the atoms in a representation-independent way, for use in each atom's fragment list.

Figure 2



As class indices are used in the fragments, themselves used for sorting, there is a **loop** between sorting the descriptions and renumbering the atoms, repeated as long as the numbering is not stable. No oscillatory behaviour was observed during this procedure.

4. A practical example : C_5 isomers

The following describes atoms and their environments for the C_5 molecule, illustrating the method used to build the fingerprints. The detailed fingerprint information displayed here is available using GalvaStructures' `/DFP` option.

```
1 :
C1 C2 C3 C4 C5
C1 S S S S
C2 S S S
C3 S S
C4 S
C5
0.0 H, 6 Cy, 0 ArCy. 1 equivalence class. Highest atom rank=2.
```

First the connectivity matrix is displayed, as well as some molecule-wide information : number of Hs (in this example it is 0), number of cycles (6 for this isomer), number of aromatic cycles and of equivalence classes, and the highest atom rank.

```
Atom 1 (Class 0) : C, MaxRank :2, Hybr. :sp3, 4 first+0 other
neighbours.
```

First comes information about the central atom : nature, rank of the farthest neighbour, hybridisation state and number of nearest and farther neighbours.

```
Neighbours : Atom 2 :Rk2, C, 4 1stNg, Hysp3, BdS. Atom 3 :Rk2, C, 4
1stNg, Hysp3, BdS.
Atom 4 :Rk2, C, 4 1stNg, Hysp3, BdS. Atom 5 :Rk2, C, 4 1stNg, Hysp3,
BdS.
```

Then each neighbour is described : first its number, then its rank with respect to the central atom, its nature, number of its first neighbours, and finally the nature of the bond between it and the central atom (this latter information mentioned only for first neighbours, of course).

```
Bonds by rank : Rk0 :4+0+0 Rk1 :6+0+0
```

Now the list of bonds of each of the three types (S, D, T - in this isomer there are only single bonds) as seen in concentric order from the central atom is mentioned : There are 4 bonds of type S between the central atom (drawn in black). All other bonds bind second neighbours together and are all of rank 2, which is not intuitive when looking at a 2-D representation. These bonds are shown in blue. The central atom considered is that on top.

```
Fragments : F0 :0-0,BdS,Rk0 F1 :0-0,BdS,Rk0 F2 :0-0,BdS,Rk0 F3 :0-
0,BdS,Rk0
F4 :0-0,BdS,Rk1 F5 :0-0,BdS,Rk1 F6 :0-0,BdS,Rk1 F7 :0-0,BdS,Rk1 F8 :0-
0,BdS,Rk1
F9 :0-0,BdS,Rk1
```

Last, a complete ordered list of all fragments in the molecule is given. Fragments are numbered from 0 as is common with programmers. Each fragments is described by the classes of the atoms bound (in this example, all atoms belong to class 0), the bond nature (here, always S), and the fragment's rank (as in the bond ranks earlier).

```
Atom 2 : same as 1.
```

Descriptions of equivalent atoms are not repeated.

```
Atom 3 : same as 2.
Atom 4 : same as 3.
Atom 5 : same as 4.
```

The other isomer descriptions are similar and follow :

```
2 :
C1 C2 C3 C4 C5
C1 . D D .
C2 . D D
```

```
C3 . D
C4 .
C5
0.0 H, 1 Cy, 0 ArCy. 1 equivalence class. Highest atom rank=3.

Atom 1 (Class 0) : C, MaxRank :3, Hybr. :sp, 2 first+2 other
neighbours.
Neighbours : 3 :Rk2, C, 2 1stNg, Hysp, BdD 4 :Rk2, C, 2 1stNg, Hysp,
BdD
5 :Rk3, C, 2 1stNg, Hysp 2 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :0+2+0 Rk1 :0+0+0 Rk2 :0+2+0 Rk3 :0+1+0
Fragments : F0 :0-0,BdD,Rk0 F1 :0-0,BdD,Rk0 F2 :0-0,BdD,Rk2 F3 :0-
0,BdD,Rk2
F4 :0-0,BdD,Rk3
Atom 2 : same as 1.
Atom 3 : same as 2.
Atom 4 : same as 3.
Atom 5 : same as 4.


3 :
C1 C2 C3 C4 C5
C1 . D S S
C2 S D S
C3 . S
C4 S
C5
0.0 H, 4 Cy, 0 ArCy. 2 equivalence classes. Highest atom rank=3.

Atom 1 (Class 0) : C, MaxRank :3, Hybr. :sp2, 3 first+1 other
neighbours.
Neighbours : 3 :Rk2, C, 4 1stNg, Hysp3, BdS 4 :Rk2, C, 3 1stNg, Hysp2,
BdS
5 :Rk2, C, 3 1stNg, Hysp2, BdD 2 :Rk3, C, 3 1stNg, Hysp2
Bonds by rank : Rk0 :2+1+0 Rk1 :2+0+0 Rk2 :2+1+0
Fragments : F0 :0-0,BdD,Rk0 F1 :0-0,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
1,BdS,Rk1
F4 :0-1,BdS,Rk1 F5 :0-0,BdD,Rk2 F6 :0-0,BdS,Rk2 F7 :0-1,BdS,Rk2
Atom 2 : same as 1.
Atom 3 : same as 2.
Atom 4 : same as 3.

Atom 5 (Class 1) : C, MaxRank :2, Hybr. :sp3, 4 first+0 other
neighbours.
Neighbours : 1 :Rk2, C, 3 1stNg, Hysp2, BdS 2 :Rk2, C, 3 1stNg, Hysp2,
BdS
3 :Rk2, C, 3 1stNg, Hysp2, BdS 4 :Rk2, C, 3 1stNg, Hysp2, BdS
Bonds by rank : Rk0 :4+0+0 Rk1 :2+2+0
Fragments : F0 :0-1,BdS,Rk0 F1 :0-1,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
1,BdS,Rk0
F4 :0-0,BdD,Rk1 F5 :0-0,BdD,Rk1 F6 :0-0,BdS,Rk1 F7 :0-0,BdS,Rk1
```

```
4 :
C1 C2 C3 C4 C5
C1 . T . S
C2 . T S
C3 . S
C4 S
C5
0.0 H, 2 Cy, 0 ArCy. 2 equivalence classes. Highest atom rank=3.

Atom 1 (Class 0) : C, MaxRank :3, Hybr. :sp, 2 first+2 other
neighbours.
Neighbours : 3 :Rk2, C, 4 1stNg, Hysp3, BdS 5 :Rk2, C, 2 1stNg, Hysp,
BdT
2 :Rk3, C, 2 1stNg, Hysp 4 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :1+0+1 Rk1 :1+0+0 Rk2 :2+0+0 Rk3 :0+0+1
Fragments : F0 :0-0,BdT,Rk0 F1 :0-1,BdS,Rk0 F2 :0-1,BdS,Rk1 F3 :0-
1,BdS,Rk2
F4 :0-1,BdS,Rk2 F5 :0-0,BdT,Rk3
Atom 2 : same as 1.
Atom 3 : same as 2.
Atom 4 : same as 3.

Atom 5 (Class 1) : C, MaxRank :2, Hybr. :sp3, 4 first+0 other
neighbours.
Neighbours : 1 :Rk2, C, 2 1stNg, Hysp, BdS 2 :Rk2, C, 2 1stNg, Hysp,
BdS
3 :Rk2, C, 2 1stNg, Hysp, BdS 4 :Rk2, C, 2 1stNg, Hysp, BdS
Bonds by rank : Rk0 :4+0+0 Rk1 :0+0+2
Fragments : F0 :0-1,BdS,Rk0 F1 :0-1,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
1,BdS,Rk0
F4 :0-0,BdT,Rk1 F5 :0-0,BdT,Rk1
```


```
5 :
C1 C2 C3 C4 C5
C1 . . D D
C2 T S .
C3 . S
C4 S
C5
0.0 H, 2 Cy, 0 ArCy. 3 equivalence classes. Highest atom rank=3.

Atom 4 (Class 0) : C, MaxRank :3, Hybr. :sp2, 3 first+1 other
neighbours.
Neighbours : 1 :Rk2, C, 3 1stNg, Hysp2, BdS 2 :Rk2, C, 2 1stNg, Hysp,
BdS
5 :Rk2, C, 2 1stNg, Hysp, BdD 3 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :2+1+0 Rk1 :0+1+0 Rk2 :1+0+1
Fragments : F0 :0-2,BdD,Rk0 F1 :0-0,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
2,BdD,Rk1
F4 :1-1,BdT,Rk2 F5 :0-1,BdS,Rk2
Atom 5 : same as 4.

Atom 2 (Class 1) : C, MaxRank :3, Hybr. :sp, 2 first+2 other
neighbours.
Neighbours : 3 :Rk2, C, 3 1stNg, Hysp2, BdS 4 :Rk2, C, 2 1stNg, Hysp,
BdT
```

```
5 :Rk3, C, 3 1stNg, Hysp2 1 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :1+0+1 Rk1 :0+0+0 Rk2 :2+1+0 Rk3 :0+1+0
Fragments : F0 :1-1,BdT,Rk0 F1 :0-1,BdS,Rk0 F2 :0-2,BdD,Rk2 F3 :0-
0,BdS,Rk2
F4 :0-1,BdS,Rk2 F5 :0-2,BdD,Rk3
Atom 3 : same as 2.

Atom 1 (Class 2) : C, MaxRank :3, Hybr. :sp, 2 first+2 other
neighbours.
Neighbours : 4 :Rk2, C, 3 1stNg, Hysp2, BdD 5 :Rk2, C, 3 1stNg, Hysp2,
BdD
2 :Rk3, C, 2 1stNg, Hysp 3 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :0+2+0 Rk1 :1+0+0 Rk2 :2+0+0 Rk3 :0+0+1
Fragments : F0 :0-2,BdD,Rk0 F1 :0-2,BdD,Rk0 F2 :0-0,BdS,Rk1 F3 :0-
1,BdS,Rk2
F4 :0-1,BdS,Rk2 F5 :1-1,BdT,Rk3


6 :
C1 C2 C3 C4 C5
C1 . S S D
C2 D D .
C3 . S
C4 S
C5
0.0 H, 3 Cy, 0 ArCy. 3 equivalence classes. Highest atom rank=3.
Atom 1 (Class 0) : C, MaxRank :3, Hybr. :sp2, 3 first+1 other
neighbours.
Neighbours : 3 :Rk2, C, 3 1stNg, Hysp2, BdS 4 :Rk2, C, 3 1stNg, Hysp2,
BdS
5 :Rk2, C, 3 1stNg, Hysp2, BdD 2 :Rk3, C, 2 1stNg, Hysp
Bonds by rank : Rk0 :2+1+0 Rk1 :2+0+0 Rk2 :0+2+0
Fragments : F0 :0-0,BdD,Rk0 F1 :0-1,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
1,BdS,Rk1
F4 :0-1,BdS,Rk1 F5 :1-2,BdD,Rk2 F6 :1-2,BdD,Rk2
Atom 5 : same as 1.

Atom 3 (Class 1) : C, MaxRank :3, Hybr. :sp2, 3 first+1 other
neighbours.
Neighbours : 1 :Rk2, C, 3 1stNg, Hysp2, BdS 2 :Rk2, C, 3 1stNg, Hysp2,
BdS
5 :Rk2, C, 2 1stNg, Hysp, BdD 4 :Rk3, C, 3 1stNg, Hysp2
Bonds by rank : Rk0 :2+1+0 Rk1 :0+1+0 Rk2 :2+1+0
Fragments : F0 :1-2,BdD,Rk0 F1 :0-1,BdS,Rk0 F2 :0-1,BdS,Rk0 F3 :0-
0,BdD,Rk1
F4 :1-2,BdD,Rk2 F5 :0-1,BdS,Rk2 F6 :0-1,BdS,Rk2
Atom 4 : same as 3.

Atom 2 (Class 2) : C, MaxRank :3, Hybr. :sp, 2 first+2 other
neighbours.
Neighbours : 3 :Rk2, C, 3 1stNg, Hysp2, BdD 4 :Rk2, C, 3 1stNg, Hysp2,
BdD
1 :Rk3, C, 3 1stNg, Hysp2 5 :Rk3, C, 3 1stNg, Hysp2
Bonds by rank : Rk0 :0+2+0 Rk1 :0+0+0 Rk2 :4+0+0 Rk3 :0+1+0
Fragments : F0 :1-2,BdD,Rk0 F1 :1-2,BdD,Rk0 F2 :0-1,BdS,Rk2 F3 :0-
1,BdS,Rk2
F4 :0-1,BdS,Rk2 F5 :0-1,BdS,Rk2 F6 :0-0,BdD,Rk3
```

From complete descriptions to fingerprints

All this information is then collected into an integer array, itself compacted into a byte array, taking into account the fact that all values considered have well-defined ranges. For instance, a number of neighbours will never be lower than 1 or higher than 4, so it can be encoded into 2 bits (values 0 to 3).
Then each atom's description is compared with the others, using ordinary memory comparison (`memcmp` in the C language). Different environments describe nonequivalent atoms. The binary descriptions can then be sorted in arbitrary order (using `qsort` and `memcmp`); this creates our canonical numbering.
Even though complete descriptions, omitting nothing, are necessary for distinguishing nonequivalent atoms, only a subset of the data is necessary for fingerprints.
This information is useful because fingerprints take up lots of room and the complete descriptions contains redundancies as a precaution. Even so, a 10-atom molecule will need around 500 bytes for its fingerprint, much more than the genotype itself (around 10 bytes) !
Some kind of second-level compaction is obviously needed, while taking into account memory-versus-computation time tradeoffs. Like compact encoding versus straight, this would bring fingerprint size dependency from the order of N^2 to that of $N^{1.4}$ or so, fractal dimension of a maximally-branched molecule.

The one degeneracy observed

GalvaStructures fails to distinguish between the two isomers shown opposite : this the cause of the difference in isomer numbers between MOLGEN and GalvaStructures, as explained in the next section.
This degeneracy is due to the fact that we consider concentric environments globally while we should explore them in each of the four bond directions separately.
It should be noticed that this is a kind of cis-trans isomerism, of the kind notoriously not supported by the connectivity matrices that underlie GalvaStructures' operation.

We are planning to generate separate fingerprints for each bond direction. This should remove the degeneracy observed and will be implemented in the next version of GalvaStructures.

5. Practical results. Comparison with other generators.

Intensive tests were performed on compounds from multiple sources in the literature. At least **eight tries** were made for each compound. In the main series of experiments, performed in 1996, population sizes were as large as would fit in 32 MB, i.e. 40,000 to 130,000 chromosomes depending on the number of atoms in the molecule. Evolution was stopped at stabilisation, this being defined as no new isomer appearing after **five million** genetic operations or twelve hundred times the expected number of isomers, whichever is greater. Thus we can be confident (though not completely sure) that the isomer numbers mentioned are **the maximal possible numbers of isomers** within the limits of the method, not underestimated numbers due to lack of patience.

Due to the population-based nature of the generator, it is very slow to use our method beyond a few thousand expected isomers : the last isomers take very long to appear. Each new isomer being independently created, the more complete the isomer population, the least likely it is for a not-previously-present one to appear.

The population size does not need to be large : we obtained all 4679 isomers of C_8H_{10} from a population of only 20,000 in five minutes. Using large populations is just a good-practice precaution. Obtaining the 7981 isomers of C_8H_6 used all of 230 MB of memory in a population of 600,000 chromosomes and took 20 minutes on a Pentium II-350. On that machine and for C_8H_6, one fitness computation takes 0.07 ms, that is 13,000 fitness computations a second. All this is nevertheless very slow compared to MOLGEN, which is nearly-instantaneous. The worst-case performance is very bad : while trying to compute the C_{10} isomers, we had only 4286 after **24 hours** (1.1 10^9 tries) out of the 4330 expected (see below, Table 2). The more cycles and insaturations, the slower the convergence.

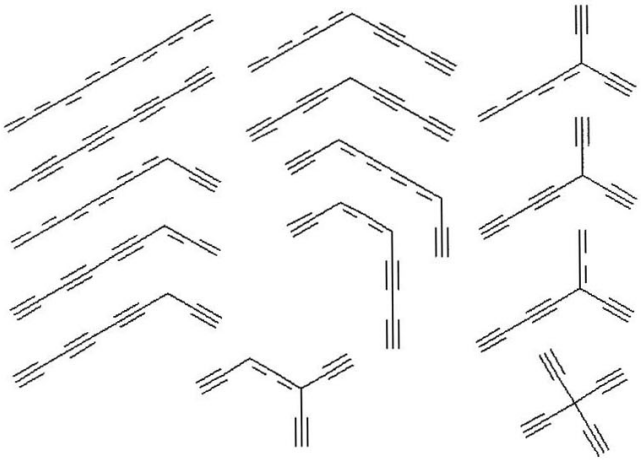
Table 1 describes the isomer numbers found by GalvaStructures for some acyclic compounds, compared to those of DENDRAL (Ref. 13). The DENDRAL values are at the top of each square, ours at the bottom if different. Squares in italics have been independently checked by hand by two qualified people, and we have found our numbers to be the correct ones.

Nb of Cs	4	5	6	7	8	9	10	11	12
C_nH_{2n+2}	2	3	5	9	18	35	75	159	355
C_nH_{2n}	3	5	13	27	66	153	377	915 914	2315 2281
C_nH_{2n-2}	4	9	23	55 58	152	375 400	1048 1072	2877 2876	
C_nH_{2n-4}	2	6	*21* *22*	59 66	195 210	563 629	1823 1914		
C_nH_{2n-6}	1	4	15	45 55	182 200	629 698	2270 2388		
C_nH_{2n-8}			5	*21* *24*	110 122	511	2113 2083	8057 7963	
C_nH_{2n-10}			1	8	45 48	262 261	1304 1283		
C_nH_{2n-12}					9	77 76	532 524		
C_nH_{2n-14}					1	*13* *14*	135 134		
C_nH_{2n-16}							17		
$C_nH_{2n+2}O$	7	14	32	72	171	405	989	2460 2426	6123 6045
$C_nH_{2n+3}N$	8	17	39	89	211	507	1238	3057	

Table 1

The differences in numbers obviously invite comments. I did not find it likely that we could be right and DENDRAL wrong, but careful checking by hand of some of the least numerous isomer families (grey background in Table 1) confirmed that some isomers are indeed "forgotten" by DENDRAL (see Figure 10 for a list of the 14 isomers of C_9H_4, Figure 11 for a list of the 24 isomers of C_7H_6).

Figure 10 : isomers of C_9H_4



This is unexpected, first because the DENDRAL isomer generator has been "rigorously proved" (Hu & Xu) ...or not (Ref. 14), secondly because we expected some degeneracy in GalvaStructures because of the bond-by-rank summation described earlier. We would have expected GalvaStructures to find fewer isomers than DENDRAL - actually, the opposite happens most of the time. The intricate nature of most other isomer generators does not allow one to exclude the occasional forgotten isomer.

When dealing with large numbers of isomers, other considerations apply. It would not be surprising that GalvaStructures find a smaller number of isomers than expected, because the last few isomers take a long time to appear. This **asymptotical behaviour** is due to two things : the first isomers taking up lots of room in the population, there is less room available for the next ones to come into play and the "usable" population size diminishes; and also, any isomer having (*a priori*) the same probability to appear, the more isomers, the less likely are newly-appeared ones not to be already present in the population.

Indeed, at the beginning of evolution, new isomers appear by the hundred at each generation; at the end, one often sees a new isomer only once every few generations. As far as large numbers of isomers are concerned, speed is not GalvaStructures' most remarkable feature. However, the memory size used for the experiments at the time, 32 MB, is small compared to today's (1999) equipment.

Considering its good deduplicating power, it would probably be more advantageous to use the fingerprint method on a systematic generator, even a crude and very redundant one, instead of a stochastic one. The advantage of using the genetic algorithm is that only very basic chemical knowledge is needed, and the time to a working program is short.

Figure 11 : 24 isomers of C_7H_6

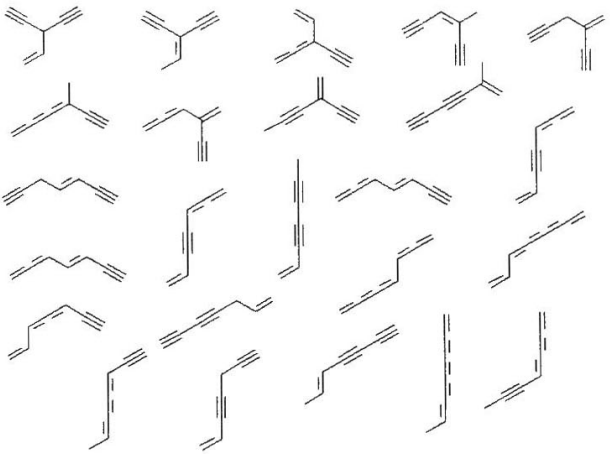


Table 2 describes isomer numbers for the C_n family, as taken from Hu & Xu's paper mentioned above. Our numbers are at the bottom of each square.

Nb of Carbons	**5**	**6**	**7**	**8**	**9**	**10**
Nb of C_n isomers	**6** **6**	**19** **19**	**50** **50**	**204** **199**	**832** **828**	**4330** **4286**

Table 2

The C_n molecules are the trickiest because there are very few differences between two isomers, while acyclic alkanes seem to be the easiest to count, and were historically the first ones tackled (as early as 1877, Ref. 15). As for very cyclic molecules (e.g. C_n) that are very far from being flat, a sheet of paper is just **inadequate for representation** and identifying nonrepeating isomers without omissions is nearly humanly unfeasible. In other words, the necessary inventory and comparison of isomer lists are difficult, bordering on the impossible, and while the results are obviously different, it is difficult to know which one is correct. I did not even **try** to draw the 204 (or is it 199 ?) C_8 isomers.

It is well-known that manual generation is prone to duplicates & omissions (Ref. 16). C_n molecules are probably **the best benchmark for a molecular topological index**. Dealing with them adequately may be useful in the emerging field of **buckminsterfullerenes** chemistry.

Table 3 lists isomer numbers for a few compounds from Hu & Xu's paper cited above.

Formula	**C_7O**	**$C_7H_{12}O$**	**C_7NH**	**C_6HNO**	**$C_6H_{13}NO$**	**$C_7H_{13}N$**
Nb of isomers	**356** **356**	**2589** **2589**	**2991** **2879**	**7038** **6339**	**3418** **3345**	**3826** **3809**

Table 3

The large discrepancy observed for **C_6HNO** warrants further investigation, especially when contrasted with the fair agreements observed for **$C_nH_{2n+3}N$** and **$C_nH_{2n+2}O$** (Table 1). The ready explanation is that we do not consider the **N-O** dative bond; for us, nitrogen is always trivalent. But this explanation would not be adequate to explain the small difference observed for **$C_6H_{13}NO$**.

Considering the respective natures of the isomer generators, the risk with Galvastructures is "collision" or degeneracy, where two distinct molecules have the same fingerprint, therefore missing isomers. The risk with other, generate-and-deduplicate isomer generators is to fail to recognise isomorphic structures, that is, extra, spurious isomers.

Table 4 lists isomer numbers for a few compounds generated using various isomer generators, using data from Contreras (Ref. 17) and Bohanec (Ref. 18). Empty squares mean no data.

Generator	CAMGEC	AEGIS	CHEMICS	DENDRAL	GEN	MOLGRAPH	GAStr
$C_2H_5NO_2$	84	84	87	84	84	84	84
C_3H_7NO	84	84	87	84	84	84	84
C_4H_7NO	767	764	802	764	764	764	764
C_3H_4BrCl			8		10	10	10
C_5H_8BrCl	140	140	108		140	140	140
$C_6H_{10}O$	748	747	745	747	747	747	747

Table 4

There is agreement between GalvaStructures, MOLGRAPH (now MOLGEN), AEGIS and GEN for these data. As usual, there is more consensus when the compound is saturated and has few atoms.

CHEMICS is wide off the mark. DENDRAL results are good in this table, but there were big differences exposed in Table 1.

CAMGEC is essentially in agreement with us except for the two slight discrepancies mentioned in (Ref. 17). I tried some of the Hendrickson data (Ref. 19) that were giving problems using CAMGEC. For **C_9H_{14}** with 3 cycles (obtained with `GASTR /nc9 /nh14 /bd1#0#0#0 /ny3`), we do find the expected 1278 isomers. For **$C_{11}H_{22}$** with 1 cycle, GalvaStructures conjures up 1231 isomers, also in agreement with Hendrickson.

A general comparison of Hendrickson's results against ours follows in Table 5. Numbers of carbon skeletons are easy to obtain using GalvaStructures : `GASTR /nc8 /bd1#0#0#0` finds all molecules with 8 carbons and only single bonds allowed, with no condition on the number of hydrogens or of cycles. This is the number of skeletons. The breakdown by number of cycles automatically supplied by GalvaStructures can be directly compared with the results.

		Breakdown by number of cycles										
Nb of Cs	Total skels	0	1	2	3	4	5	6	7	8	9	10
C_4	6	2	2	1	1							
C_5	21	3	5	5	4	2	1	1				

C_6	78	5	12	17	18	14	8	3	1			
C_7	353	9	29	56	79	79	59	31	9	2		
C_8	1929 1927	18	73	182	326	430	427	298	134	35	6 4	
C_9	12207 12204	35	185	573	1278	2161	2768	2616	1714	707 706	154 153	16 15

Table 5

Again, there are small discrepancies for some highly unsaturated compounds. It would be interesting to obtain the 6 C_8 isomers having 9 cycles from the authors. For repeated tries, we only obtained four of them. Paper representation of the 15 isomers of C_9 having 10 cycles has been attempted, but the result is absolutely **incomprehensible** and not repeated here.

Galvastructures was also put to work on compounds from Benecke's data (Ref. 21), all hydrocarbons with a specified gross formula (from C_3H_2 to C_8H_{18}), as opposed to Table 1 where only acyclic ones were represented. **The same pattern of small differences for larger, more unsaturated compounds is observed.**

Nb of Hs	2	4	6	8	10	12	14	16	18
C_3	2	3	2	1					
C_4	7	11	9	5	2				
C_5	21	40	40	26	10	3			
C_6	85	185	217	159	77	25	5		
C_7	356	920	1230	1031	575	222	56	9	
C_8	1804 1802	5308 5305	7982 7981	7437 7436	4679	2082	654	139	18

Table 6

The first discrepancy occurs with C_8H_2, where the are 1804 (according to MOLGEN) or 1802 (according to Galvastructures) isomers. Galvastructures finds fewer isomers than MOLGEN in a few cases, always large and very unsaturated molecules. Human checking is (again) barely possible and more work on automated comparison is needed. The nature of the observed Galvastructures degeneracy is described above, at the end of section 4. There is every reason to believe that the MOLGEN numbers are the correct ones.

The question of "**how many isomers are there actually ?**" is difficult to solve conclusively. Tables 1 to 6 indicate that problems appear sometimes only with numbers or molecule complexities that are beyond the range of the humanly verifiable. Even for a simple, saturated molecule such as $C_{14}H_{30}$, there is some doubt as to whether there are 1858 or 1859. DENDRAL sez 1859, Benecke & al. say 1858, Hu & Xu say either 1858 or 1859 (Ref. 20) depending on the paper ! We find 1858 isomers using GalvaStructures. GalvaStructures does, however, find the agreed-upon 4347 isomers for $C_{15}H_{32}$, and 10359 isomers for $C_{16}H_{34}$. For subtle differences visible only on complex molecules, it is very difficult to check by hand.

The fact that generate-and-deduplicate software and Galvastructures agree to a large extent is interesting because radically different methods were used.

I also tried the software on miscellaneous interesting problems. The 22 positional isomers of

dioxin are duly found, the correct form of anagyrin is found as well as 111 others. To save time, the `/BFP` option checks for the presence of a given fully-filled connectivity matrix among many isomers. Benecke & al.'s "BASF" problem (Ref. 21) is solved yielding 4500 isomers to choose from, no improvement at all over the 201 isomers exhibited by their method. GalvaStructures does not allow conditions on cycle lengths at the time, but we can exclude triple bonds (`/BD` option) and the possibility to specify exactly the number and type of hanging bonds decreases the number of candidates. Benecke's 201 candidates have conditions on bond type as well as cycle size. We probably need to be able to impose conditions on cycle lengths in GalvaStructures as a future improvement, using Panaye & Doucet's procedure (Ref. 22) and taking into account Figueras' remarks. But it is important to notice that, except for C_3 and C_4 rings that can be recognised on a ^{13}C NMR spectrum, a priori specification of ring size is more a **matter of chemist's habit** (not to say prejudice) than a reality.

6. The genetic algorithm and isomer generation

Even though the genetic algorithm is **admittedly very inefficient** at isomer generation, it does seem to be **exhaustive**. Our genetic algorithm/simulated annealing library seems to very exhaustively explore the possibilities space, while the fingerprint method is apparently quite competent at weeding out duplicates. Also, use of the genetic algorithm and its ability to handle various constraints will take all its meaning when a (neural network-based) ^{13}C NMR **spectrum simulator** is coupled to the software. Spectrum comparison will be a natural addition to the fitness function. The limitation to a few thousand expected isomers does not seem to be a problem for real-life tasks, especially when the number is limited to those isomers whose spectrum is close enough to the obtained spectrum.

We take pride in not special-casing our genetic algorithm library for any specific application. I also like to consider this a proof of good design. The Galvano library has an **elitist mechanism** that automatically prevents the best elements in the population from being lost. This has been shown to be a **necessary condition of convergence to the optimum** (Ref. 25).

The problem of many genotypes (bit strings manipulated by the genetic algorithm) mapping onto the same phenotype (here, connectivity matrices) is common to many genetic algorithm applications. It is important to avoid the population being swamped by what are actually variants of the same solution. Therefore, the fingerprint mechanism is a part of the basic library. If many genotypes have the same fingerprint, only one genotype is kept. Conversely, one of each differently-mapped genotypes is always kept. This mechanism thus "grabs" every newly-appeared isomer and ensures that it is kept to the end of the evolution. Thus, isomer numbers in the population never decrease.

Of course, each different application has a different way of computing the fingerprint; this is made possible by the use of a "callback function", called by the library but supplied by the application programmer.

7. Conclusion and directions for future work.

The system presented uses the unprecedented memory and processing capabilities now available to us to (apparently) exhaustively generate isomers using methods that would have been impractical only a few years back. By using somewhat crude methods, this one-man team could put together in less than six man-months a decently-working isomer generator and duplicate eliminator. The intricacies of graph theory were avoided. The difficult task of finding a molecule's symmetry group for duplicate elimination was eschewed entirely.

The point of this article is to prove that these newfangled methods can make profitable use of a computer's present processing power while releasing the user for more high-level work.

Of course the missing isomers have to be found and the problem (probably with the fingerprinting method) fixed. The very next step will be, now we have a validated generation method, to couple it with a spectrum generator. PMSI already has a neural network-based ^{13}C NMR spectrum generator. Modifying the fitness function so that an isomer's fitness depends on its closeness to the experimental spectrum puts to good use the genetic algorithm's flexibility.

It could also be interesting to explore chirality. It seems that exploration of each active carbon along the four bond directions will lead us to partial fingerprints of the same style, which may then be compared. If all four partial fingerprints are different, they can be ordered in two different ways for each active centre. However, fine software such as MOLGEN+ already does this.

The absence of mandatory lists of absent/present features (goodlist/badlist) is also a drawback, and must be corrected. This, however, like ring counting, can only be applied after candidate generation. But it does not pose any implementation problems.

I am deeply indebted to Profs. Jean-Pierre Doucet and Annick Panaye of ITODYS, University of Paris, for their guidance and learned advice in preparing this paper. Thanks to trainee Lamia Benabbas for checking some of the results.

GalvaStructures 3.2, the software described here, is available at no charge from PMSI.

Christopher Le Bret, born 1963, is a graduate of the Ecole Nationale Supérieure de Chimie de Paris and of the University of Paris. He is the technical manager of PMSI, a company specialised in artificial intelligence (or decision support, or data mining, or whatever is this week's buzzword).

References

Ref. 1

Le Bret, C., «Rebuilding Connectivity Matrices from Two-Atom Fragments using the Genetic Algorithm»
J. Chem. Inf. Comput. Sci., **36**, No 4, 1996, 678-683.

Ref. 2

Hibbert, D.B., «Generation & display of chemical structures by genetic algorithms»
Chemom. Intell. Lab. Syst., **20**, 1993, 35-43.

Ref. 3

Sedgewick, R. : «Algorithms in C»;
Addison-Wesley, 1990

Ref. 4

Chesneaux, Boisson : «Le Calcul Scientifique sur Ordinateur», Dunod, Paris, 1988, p.41

Ref. 5

Press, W, & al., «Numerical Recipes in C», Cambridge University Press, 1986, p. 16.

Ref. 6

Burden, F. : «Molecular Identification Number for Substructures Search»
J. Chem. Inf. Comp. Sci., **29**, 1989, pp. 225-227

Ref. 7

Hu & Xu, «On Highly Discriminating Molecular Topological Index»
J. Chem. Inf. Comput. Sci., **36**, No 1, 1996, 82-90.

Ref. 8

Balaban, A., «Numerical Modelling of Chemical Structures: Local Graph Invariants & Topological Indices»
in «Graph Theory & Topology» ; King, R. & Rouvray, D., Eds, Elsevier, Amsterdam, 1987, 159-176.

Ref. 9

Muller, W. & al., «The Walk ID Numbers Revisited»
J. Chem. Inf. Comput. Sci., **33**, No 2, 1993, 231-233.

Ref. 10

Weeg & Reed, «Introduction to Numerical Analysis», Blaisdell Publ. Co., 1966, p. 12.

Ref. 11

Carhart, R., «Erroneous Claims Concerning the Perception of Topological Symmetry»
J. Chem. Inf. Comput. Sci., **18**, No 2, 1978, 108-110.

Ref. 12

Shelley, C. & Munk, M., «Computer Perception of Topological Symmetry»
J. Chem. Inf. Comput. Sci., **17**, No 2, 1977, 110-113.

Ref. 13

Lederberg J., Sutherland G., Buchanan B, Feigenbaum E., Robertson A., Duffield A., Djerassi C. : «Applications of Artificial Intelligence for Chemical Inference. I. The Number of Possible Organic Compounds. Acyclic Structures containing C, H, O and N»
J. Am. Chem. Soc., 91:11, May 21, 1969, pp. 2973-2976

Ref. 14

Faulon, J.-L., «Stochastic Generator of Chemical Structure. 1. Application to the Structure Elucidation of Large Molecules»
J. Chem. Inf. Comput. Sci., **34**, No 5, 1994, 1204-1218.

Ref. 15

Cailey, «On the number of univalent radicals C_nH_{2n+1}»
Phil. Mag., Series 5, **3**, 1877, 34-35.

Ref. 16

Masinter, L. M., & al., «Exhaustive Generation of Cyclic & Acyclic Isomers»
J. Amer. Chem. Soc., 96:25, 21 Dec. 1974, 7702-7714, p.7708.

Ref. 17

Contreras, M. L. & al., «Exhaustive Generation of Organic Isomers. 3. Acyclic, Cyclic, and Mixed Compounds»
J. Chem. Inf. Comput. Sci., **34**, No 3, 1994, 610-616.

Ref. 18

Bohanec, S., & al., «Structure Generation of Constitutional Isomers from Structural Fragments»
J. Chem. Inf. Comput. Sci., **31**, No 4, 1991, 531-540.

Ref. 19

Hendrickson, J. B. & al., «Generation and Enumeration of Carbon Skeletons»
J. Chem. Inf. Comput. Sci., 31, No 1, 1991, 101-107.

Ref. 20

Hu & Xu, «Principles for Structure Generation of Organic Isomers for Molecular Formula»
Anal. Chim. Acta, **298**, 1994, 75-85.

Ref. 21

Benecke, C. & al., «MOLGEN+, a generator of connectivity isomers and stereoisomers for molecular structure elucidation»
Anal. Chim. Acta, **314** (1995), 141-147.

Ref. 22

Panaye & al., «Topological Approach of 13C NMR Spectral Simulation: Application to fuzzy Substructures»
J. Chem. Inf. Comput. Sci., **33**, No 2, 1993, 258-265.

Ref. 23

Fan, B. T. & al., «Ring Perception. A New Algorithm for Directly Finding the Smallest Set of Smallest Rings from a Connection Table»
J. Chem. Inf. Comput. Sci., **33**, No 5, 1993, 657-662.

Ref. 24

Figueras, J., «Ring Perception using Breadth-First Search»
J. Chem. Inf. Comput. Sci., **36**, No 5, 1996, 986-991.

Ref. 25

Rudolph, G., «Convergence Analysis of Canonical Genetic Algorithm»
IEEE Trans. Neural Networks, **5**, No 1, Jan. 1994, 96-104.