Rebuilding Connectivity Matrices from Two-Atom Fragments Using the Genetic Algorithm[†]

Christopher Le Bret[‡,§]

Department of Research, PMSI, 52 rue Mouffetard, F-75005 Paris

Received September 14, 1995[⊗]

A method for reconstituting a connectivity matrix from a list of fragments is described. The Genetic Algorithm is used to obtain connectivity matrices that satisfy chemical correctness rules and match the given fragment list. A brief description of the genetic algorithm is made, and the encoding method and fitness evaluation function are described in detail. Performance of this system is described on eight molecules having from 8 to 25 backbone atoms. The software builds satisfying connectivity matrices for all of the examples, albeit with varying success rates and computation times. Simulated evolution, being a stochastic process, does not behave repeatably, and statistics over 500 runs are presented for each molecule. The software used is available freely to let readers replicate the results and test the method on the examples presented or on the fragment list of their choice.

1. THE PROBLEM

As part of a larger project, we have been exploring ways of rebuilding molecular structure (in the form of a connectivity matrix) from a list of **diatomic fragments**, e.g., 2 C═C, 5 C-C, 1 C-N, etc. The fragments we consider here consist only of the nature of the bond and that of the atoms at each end. They are mostly overlapping, as all nonterminal atoms take part in more than one skeleton bond and are mentioned in more than one fragment. Each bond other than C−H is mentioned once and only once in the fragment list. These fragments are small, purely "theoretical" and do not have a physical meaning the way NMR fragments do.

Connectivity matrices are a computationally convenient way of storing and processing 2-D chemical structures.[1] We use "ordinary", noncanonicalized connectivity matrices. The lack of chirality information is not a problem as the starting data we use does not contain 3-D information either.

The "puzzle-solving" exercise is very complex because much information is lost when breaking up the molecular formula into two-atom fragments. There are many ways to put the fragments back together again, most of which are chemically incorrect. Trying to rebuild the molecule from the fragments is like looking for a needle in a haystack.

To solve this problem, we used an approach that is well-known to make haystacks much smaller: the Genetic Algorithm. Use of this evolution-based method in problem-solving and optimization was pioneered in the 1970s, and it is now, thanks to advances in computer power, becoming a mature technique. A number of "primers"[2,3] and tutorial articles[4] are now available.

Problem-solving using the genetic algorithm is already quite common in the field of chemistry, where there are many complex problems that cannot be dealt with satisfactorily through classical, analytical solving. The genetic algorithm has been used for finding minimum energy configurations[5] or generating molecular active sites under multiple global and geometrical constraints.[6] Many approaches[7] use specific operators and/or special adjustments to the genetic algorithm. Our bias, contrastingly, is toward using a "standard" genetic algorithm library and concentrating on the encoding method (which requires some specific skill) and fitness function (which requires only chemical knowledge). The genetic algorithm kernel itself is taken for granted and will only be mentioned *en passant*. The "proof of concept" problem we set up was voluntarily unconstrained, to let us explore the limits of the freedom afforded us by the genetic algorithm.

2. A SHORT HISTORY OF STRUCTURE ELUCIDATION

The first attempts at automated structure elucidation were straight **lookups** in spectrum libraries.[8] Due to varying conditions and equipment, experimental and reference spectra were not always perfectly superimposable, and more sophisticated lookup methods had to be devised, using a variety of statistical techniques to retrieve the closest ("nearest neighbor") spectrum.[9] At the time the problems were compact storage and quick retrieval. There was no attempt at interpreting the spectra, nor would it have been feasible considering the then state of the art.

Advances in computing power and knowledge representation as well as the increasing user needs made it both possible and necessary to incorporate chemical reasoning in the structure elucidation process, starting with the STIRS system.[10]

The first large-scale, **knowledge-based** structure elucidation software was DENDRAL. It first used organized decision rules to deduce the required presence ("goodlist") or absence ("badlist") of certain chemical functions and structures from mass and other spectra.

In the next step, DENDRAL-CONGEN and DENDRAL-GENOA generate plausible isomers satisfying these constraints as well as topological and configurational constraints ("candidate generation"). The starting data may be in the

[†] Keywords: structure elucidation, connectivity matrices, isomer generation, genetic algorithm.

[‡] (33 1) 45 35 87 99, pmsi@altern.com.

[§] 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 specializing in artificial intelligence.

[⊗] Abstract published in *Advance ACS Abstracts,* April 1, 1996.

form of **nonoverlapping** fragments: these are actual molecule pieces such as those obtained from mass spectrometry and may be assembled like pieces from a **jigsaw puzzle**. The fragments may also be **overlapping**: such is the case with local structural information obtained from NMR spectra, where one nucleus will influence all its neighbors' offsets; then we have to deal with small fragments that must be assembled using partial matches, rather like **dominoes**.

The candidates are then evaluated using a spectrum predictor. Expected and predicted spectrum are then compared using techniques similar to the library lookup methods.

This **generate-and-test** approach was consistent with the prevailing approach of its time, essentially based on graph theory and tree exploration, like all of the expert system field. This approach ideally requires exhaustive tree exploration but this is computationally expensive; a variety of techniques are used to keep the combinatorial explosion under control: use of large fragments or "superatoms", interaction with the chemist (the CASE toolset[11]), and use of neighboring atoms' information (such as chemical shifts as in EPIOS[12]) to "sieve and guide" [13] the generation process.

The shift in advanced computing techniques (a.k.a. artificial intelligence) from symbolic, rule-based to self-building (or "adaptive") systems makes it the next logical step to replace systematic or guided tree exploration with stochastic, fully unassisted "solution space" search, where solutions "**evolve**" under the pressure of constraint instead of being **built** according to a predefined approach.

Meanwhile, NMR spectroscopy has made much progress too. Advances in digital signal processing let us obtain skeleton information as well as the hybridization states of carbon atoms thanks to analysis of $^{13}C-^{13}C$ and $^{13}C-^{1}H$ couplings. The advent of so-called "2-D NMR" provides us with a wealth of information that constrains much more strongly a structure elucidation problem, which is very helpful provided we do not get lost in a combinatorial maze.

3. THE GENETIC ALGORITHM

The genetic algorithm (GA) lets us obtain solutions to problems that cannot be solved analytically or algorithmically. It deals with **populations of solutions** rather than with a single one. These populations evolve to better satisfy the constraints given them. It bypasses the combinatorial problem, thus not requiring pruning techniques while allowing a more complete exploration of all possible solutions.

With this method, many thousands of more or less correct solutions ("genotypes") are submitted to a crude version of the **evolution of species**: survival of the fittest and evolution of successive generations through mutation and reproduction by crossover. The most successful members of the population are allowed to imprint the next generation more than others. Through "survival of the fittest", successive generations thus contain better and better solutions to the problem considered.

The genetic algorithm is a stochastic process, relying on random (although biased) events, and nonrepeatable by nature. Theoretical studies[14] give us guarantees of convergence toward the optimum. Practice indicates that this occurs most often in reasonable time.

In problem-solving applications of this method, each "creature" in the population embodies a possible solution to the problem considered (here, a connectivity matrix), and the better it solves the problem (the closer it is to the perfect solution), the fitter it is. Fitness is indicated by a number from 0 to 250—the higher the number, the better the creature, and the more correct its corresponding connectivity matrix. Fitness is a measure of the error (distance) between the connectivity matrix being considered and the fragment list; it allows us to assign each attempt a degree of closeness to the goal. A fitness value of 250 indicates a "perfect" connection table—a table describing a chemically valid molecule that also decomposes into the specified fragments.

While we cannot rebuild a connectivity matrix directly from the information in the fragment list, we can still, for a given connectivity matrix and a given fragment list, check whether the table yields the same fragment list as the given one.

The genetic algorithm lets us "explore solution spaces" (in our case, the space of all possible connectivity matrices) very efficiently by concentrating on the best parts of the search space while at the same time not completely ignoring the rest.

The number of all possible connectivity matrices is huge; this is a case of **combinatorial explosion**. We do not have a hope to chance across a correct connectivity matrix at random ("Monte Carlo"), but the genetic algorithm can guide a whole starting population of blank connectivity matrices, through successive mutations and crossovers, toward greater and greater correctness (fewer and fewer errors), making the most of a few beneficial chance mutations until a few elements of the population satisfy all constraints of chemical correctness as well as conformity with the fragment list.

3.1. The Encoding Method. The genetic algorithm requires us to encode each solution as a string of 0s and 1s. Maximal efficiency is attained when the encoding is as compact as possible, to reduce the number of possible strings and therefore the size of the genetic search space. For all the efficiency of the genetic algorithm, it is a good idea to "help" it as much as possible. This accelerates convergence.

The first 7 bits of the genotype encode an integer number between 1 and 128 that represents the total number of atoms other than hydrogen in the connectivity matrix. The next bits are read as the number of carbon atoms, and so on with all the atoms handled by the software, in our case O, N, S, F, Cl, Br, and quaternary N^+. There can be any number of these atoms from 0 to 127. The software uses the smallest necessary number of bits. When all atom numbers are set by the user, the numbers above are the same for the whole population and therefore do not need to be carried by (encoded in) the chromosomes.

Now we need to encode the connection matrix itself. We know the number n of all atoms in the connectivity matrix (it is the sum of the numbers of atoms just extracted). We then expect $(n*(n-1)/2)$ places to fill: that is the useful part of the matrix, one half excluding the diagonal (as shown in the software). Then $(n*(n-1)/2)$ groups of 2 bits are read from the string, each having a value from 0 to 3: 0 means no bond, 1 means single bond, 2 double bond, etc.

To make encoding as compact as can be, and avoid exploration around useless candidate solutions, only bond types present in the requested data may be encoded. Generally two bits per possible bond are enough, but if the triple bond is also used, three bits are required to accommodate these five possible values and chromosome length increases by 50%. Conversely, if only single bonds are necessary, only one bit per bond is needed and the length drops by half, accelerating convergence.

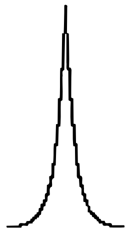


Figure 1. Hat function.

The genotype length for a connectivity matrix with 13 atoms of two different kinds will be $7 + 4 \times 2 + (13 \times 12/2) \times 2 = 171$ bits, that is 22 bytes. With 22 atoms this climbs to 60 bytes: genotype length varies as the square of the number of atoms and this is a problem. See the discussion about compact encoding in the conclusion.

3.2. The Fitness Function. The fitness function **gives each tentative solution a value that is a little better each time a solution has improved** (shows greater closeness to chemistry rules and the fragments specified). We need something subtler than giving a mark of 0 to an erroneous solution and 1 to a correct one. We need a fitness function that increases slightly when a solution becomes a little less bad. Most of the effort described in this paper went into designing a good and generic fitness function.

The fitness function first checks the now-decoded connectivity matrix for **chemical correctness**. On a scale from 0 to 250, no connectivity matrix will receive a mark higher than 150 if it is not chemically correct. Evaluation stops here if the connectivity matrix is not chemically correct, or else the connectivity matrix is further assessed for **consistency with the fragment list**.

We first compute a series of values concerning each atom: number of electrons engaged in bonds, number of number of aromatic bonds, as well as molecule-wide values: total number of atoms of each type, total number of fragments of each type, total number of aromatic bonds, etc. The number of hydrogens is deduced from the number and nature of skeleton bonds.

Most of the values can be compared against "good" values: no atom may have more than a given number of bonds; no aromatic cycle may have other than six bonds between six carbons; etc. We can then compare the values from the connectivity matrix being considered with the ideal; any difference means that the creature will receive less than the maximum mark for this given feature. The function of mark given against deviation takes the shape of a kind of hat (Figure 1). The perfect value, 1, corresponds to the top of the hat. There may be a "flat top" instead of a peak if there is a range of equally acceptable values. For instance, a carbon atom is allowed to engage one to four of its electrons in bonds equally. Most of the fitness function's fine-tuning is setting the width of the hat (how quickly it goes to zero on each side of the peak) for each kind of comparison.

Fitness computation consists of many individual comparisons. We use two running totals, one is the maximal number of points attainable, F_{Max}, and the other is the number of points obtained by the connectivity matrix being examined, F_{Curr}. For each test performed, we increase F_{Max} by one and F_{Curr} by *hat (*expected value − actual value*)*. F_{Curr} will be one if the actual value being considered is equal to the expected value (or within the range of legal values), less than one otherwise. The final fitness value is $250 \times (F_{Curr}/F_{Max})$. By simultaneously increasing the numerator and the denominator at each test, we may perform a variable number of tests without too much accounting. This flexibility proved useful when tuning up the fitness function.

The computation proceeds in three successive steps:

−Comparison of the number of atoms of each type with the plausible numbers (reckoned from the fragment list): If any number of atoms is outside the prescribed range, the connectivity matrix receives a total fitness value smaller than 25 and computation stops for this matrix. Options **/NC**, **/NO**, **/NN**, **/NH**, and **/NS** can be used to circumvent this step and directly dictate the number of atoms of each kind (thus saving time). All results presented here were obtained using full information—exact atom numbers as well as the fragment lists.

−Comparison of the number of bonds of each atom with plausible values (from chemistry rules): Atoms with no bonds or too many bonds for their available electrons incur a penalty. Only C, N, O, and S atoms may have bonds with H atoms. If there are bonds between atoms other than C, N, O, S, and H, there is a penalty.

−Connectivity matrices with too many hydrogen atoms (as compared with the maximum value of $C_nN_mH_{2n+m+2}$ for a fully saturated, acyclic alkane/amine) are discouraged, because this means that the connectivity matrix actually contains two disjoint molecules (there are too many hydrogens for all the backbone atoms to be all connected). However, this is not foolproof, as the connectivity matrix may evolve into containing two fragments, one or more of which having unexpected cycles, thus respecting globally the upper limit on the number of hydrogens.

−A specific check that the connectivity matrix embodies **a molecule all of one piece** is performed. There is a penalty if there is more than one independent fragment in the table.

This first part account for the first 100 units of the 0−250 fitness range. Connectivity matrices with atoms having more bonds than chemically allowed leave evaluation here, with fitness no higher than 100. We want to strongly encourage sparse tables.

The tables are then **further checked**:

−Atoms carrying other than zero or two aromatic bonds are penalized. Atoms having aromatic bonds together, but not forming a cycle or not being six, are penalized. The number of aromatic fragments (incomplete cycles) in the molecules is counted.

−Atoms others than nitrogen (when allowed by the fragment list) or carbon having aromatic bonds entail immediate exit with fitness smaller than 150.

−If the total number of aromatic bonds in the whole molecule is different from 6*n*, there is also a penalty. We take into account the case of many independent phenyls. For polyaromatics, the way we count electrons in connectivity matrices imposes that we regard them as one phenyl plus 2*m* double bonds, so there is no contradiction there.

−C═C═C, −C═CH2, and −C═NH groups are discouraged as being unlikely (one can push the genetic algorithm in any direction with such **arbitrary** rules). This corresponds to the classical **bad list**. Forbidding carbons carrying two double bonds is chemically justified, while the latter two groups are excluded because we wanted to encourage in-cycle double bonds.

Connectivity matrices with "forbidden" atoms in aromatic cycles or containing "badlist" fragments leave evaluation here, with fitness no higher than 150.

Chemically correct tables are **finally matched to the fragment list**. In this third and final stage, the expected and actual numbers of fragments are compared. Every difference is penalized, using the "hat" function in the same way.

A connectivity matrix satisfying every fragment constraint and chemical rule will receive a fitness value of 250, the maximum value.

These are basic, general rules. Anything else can be added to encourage molecules to have or not have any specific chemical group or feature.

It is to be noted that if there is the data supplied are **inconsistent**, or if the data as supplied correspond to no valid connectivity matrix, the algorithm will "do its best" and supply the most satisfying connectivity matrix—the one that satisfies most constraints. We witnessed this while finishing up the example software—there were typing errors in the predefined data and this behavior allowed us to detect them. The fitness curve will then hit a ceiling under 250, for instance 243. This is known as **robustness** and is not present in every problem-solving method.

The net result of all this is that **any connectivity matrix having a fitness value of 250 is guaranteed to**

—have correct chemical bonds on all its atoms;

—embody one solid molecule (not disjoint fragments)

—have well-formed aromatic cycles

—have no C═C═C, −C═CH2 or −C═NH groups

—yield fragments that match exactly the given list.

These are strong constraints, but working from two-atom fragments still leaves lots of room for isomerism.

One complete fitness calculation takes about 0.60 ms on a Pentium-100 processor for the largest molecule (buzepide).

3.3. Genetic Operators and Diversity-Enhancing Techniques. In our approach, the fitness function and the encoding scheme make up all of the necessary work to solve any problem using the genetic algorithm and are the only parts that change across applications; genetic manipulation of the encoded strings is problem-independent and makes only use of "regular" operators. For the rest of the software (genetic algorithm management, user interface, etc.) we used our pre-existing Galvano 2.9 Genetic Algorithm/Simulated Annealing library[15] to avoid duplicating effort. This software is written in C and runs in protected-mode DOS. This is one of the good points of the genetic algorithm: one does not have to write lots of code each time, just the encoding and fitness functions.

The tool used offers the following operators:

—Two-parent, two-point, two-children crossover: Starting from two parents, say P1 having the 000000000000 string and P2 having 111111111111, we randomly select two "crossover points" and cut, swap, and splice the middles of the parents' chains, giving two children, say 111000001111 and 000111110000. Here the crossover points were between the third and fourth and the eight and ninth bits (genes), respectively.

—Ordinary mutation and point inversion.

—Lamarckism: Starting from one parent, we randomly select two points and systematically try out all combinations of zeroes and ones between these points. The best combination is retained. This is a local optimization operator.

All operators are applied to members of the population selected through biased random. Likelihood of a chromosome's selection for taking part in crossover or being applied

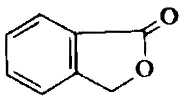


Figure 2. Phthalide.

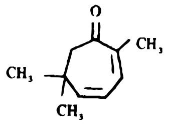


Figure 3. Eucarvone.

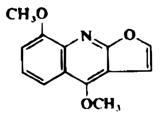


Figure 4. α-Fagarine.

an operation to is proportional to the exponential of the chromosome's fitness value.

We also use several population control methods:

—Elitism: We systematically keep the few best performers through the next generation, in addition to their taking part in the regular selection for operations.

—The rates of operations applied to the population are adjusted using Srinivas' and Patnaik's scheme.[16] More mutations and inversions are performed when population diversity is decreasing, and conversely the part of crossover is increased when the population is very diverse and concentrating and consolidating the good partial solutions is required.

—Also, each operator's performance (in terms of how often it improves the solutions it is applied to) is monitored, and the better performers are applied more often. Performance curves are visible using the software's **/DQ** option.

We did not use any problem-specific operators, as these do not justify the time spent on them and also defeat the intended (and observed in nature) perfect generality of the genetic algorithm.

4. RESULTS ON ACTUAL COMPOUNDS

We took compounds at random in a handbook.[17] We chose relatively complicated structures (polycycles, etc.). The structures were chosen so as to provide variety from the simple to the quite complex and give us a view of the possibilities and limits of the method, not to show the method in the best possible light.

The rules of chemistry described above and put to use here are general, so success does not depend on the choice of examples. Users can submit any fragment list to the software to test this claim.

It should be kept in mind that stochastic processes are nonrepeatable by nature and only lend themselves to statistical analysis. For each compound tested we made 500 trial runs using a population size of 2000 solutions and report statistics on the number of successes.

The evolution process does not always take the same time, and it may become "stuck" and not find a satisfactory solution within reasonable time. Evolutions where the best fitness had been stable (and smaller than 250) for 100 generations were stopped and deemed failures. "Successful evolution" means that the software did find a connectivity matrix that satisfied all chemical and fragment number conditions, with fitness 250. In many cases, this may mean different isomers from the one expected.

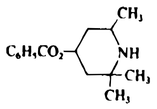
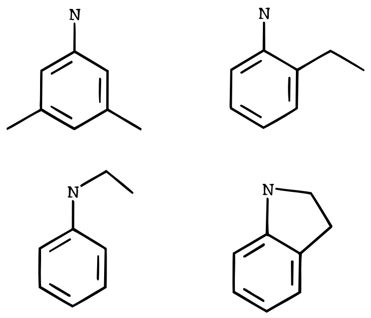


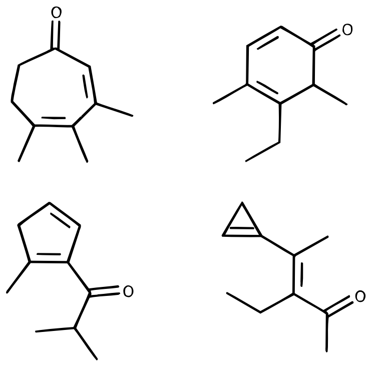
Figure 5. *β*-Eucaine.

For instance, in the case of **dimethylanilin**, we may obtain any of the positional isomers or even get an ethyl group on the phenyl instead of the two methyls. The two molecules on the left may be output by the software as being fully correct by the fragments; however, the two molecules on the right are incorrect, as they have an extra C-N fragment. The rightmost one also has the wrong number of cycles and therefore two missing hydrogens. The two top molecules will receive the maximum possible fitness value; the others will not.



All this is consistent with the fragment list as supplied.

Eucarvone as reconstituted may have its double bonds and methyl groups pretty much anywhere on the skeleton, not to mention cycle length. All the following molecules satisfied all constraints given, in terms of number of atoms and bonds of each kind, and number of cycles:



This is again consistent with what little information we supplied. These "correct" solutions reveal **a crying need for more information**: the hybridization states of the carbons would have helped a lot as well as information about how many hydrogens are connected to each. The only error we avoid is the presence of side $=CH_2$s: this has been specifically excluded as part of the "badlist".

The results are summarized in Table 1.

The statistical success rates shown are averages over 500 runs. For instance, 79 of the 500 runs involving buzepide finished with a completely correct solution (fitness 250). All other runs were interrupted after no improvement was observed for 100 generations. These rates estimate your chance of success on a single try. The time brackets indicated contain 80% of the number of runs of each kind.

The software used lets us perform multiple independent evolutions at the same time. With a compound having a 15.8% probable success rate, by launching it with 10 "families" (option **/f**) we would probably have at least one successful evolution among the 10. This is one way around the nonreproducibility of the genetic algorithm.

Table 1

name	formula	no. of successes	success rate (%)	time to solution (s)
dimethylaniline	$C_8H_{10}N$	500	100	1.5−8.3
phthalide (Figure 2)	$C_8H_6O_2$	403	80.6	2.7−40.3
eucarvone (Figure 3)	$C_{10}H_{14}O$	500	100.0	2.6−4.6
α-fagarine (Figure 4)	$C_{13}H_{11}NO_3$	355	71.0	8.2−97.7
β-eucaine (Figure 5)	$C_{15}H_{21}NO_2$	240	48.0	6.7−115.7
etizolam	$C_{17}H_{15}ClN_4S$	134	26.8	63.0−205.8
adinazolam	$C_{19}H_{18}ClN_5$	6	1.2	81.1−332.7
buzepide	$C_{22}H_{28}N_2O$	79	15.8	128.3−513.1

The structure is easily and quickly reconstituted under 15 or so backbone atoms. For more complex structures, finding a solution takes longer and success is less frequent. The solution is of course more easily found when the problem is simple (the search space is smaller), hence the increasing time and decreasing success rate as molecule complexity goes up.

However, the systems also works for nontrivial molecules. The molecular structures of some of the compounds tested are shown. The most complicated structures tested belong to the family of azepines. We met with less success with these, but mainly because we deliberately refused to over-simplify (see section 6 about the expression of phenyls).

All those example compounds are accessible in the software, using options **/x1** to **/x8**, respectively.

5. MAKING IT PRACTICAL

Having implemented the basic mechanism for reconstituting connectivity matrices, thus validating the concept, we considered what was needed to **find a solution more quickly**. We try to provide "hints" that help the population go to the good parts of the possibilities space straight away.

First, we cannot just let the population span all possible connectivity matrices: the increase in the number of possibilities would be frightening. The population would eventually migrate toward the best parts of the data space, but the users need fast answers. The fragment list is therefore analyzed and upper and lower bounds for the number of each atom (backbone atoms, halogen, and hydrogen atoms) are computed. Indeed, we can estimate what are the largest and smallest numbers of atoms of each kind necessary to realize the bonds—from each bond on a separate atom to as many bonds per atom as chemically possible.

Chromosome encoding is then adjusted so tables only decode into plausible numbers of atoms of each kind. Again, this narrows the search and facilitates the GA's work. Chromosomes may or may not all have the same length (total number of atoms).

We can also, more directly, let the user specify the exact number of Carbon, Oxygen, Nitrogen, Hydrogen, and Sulfur atoms if they are known. These are options **/NC**, **/NO**, **/NN**, **/NH** and **/NS**, respectively, in the software.

The exact number of cycles can be supplied (option **/NY**) if known.

The exact number of hydrogens can be supplied (option **/NH**); this is redundant if all the fragments and other atom numbers are supplied but serves as a consistency check.

These options add very strong constraints and turn out to be necessary for handling nontrivial molecules. They are applied in the examples supplied.

It is also possible (a recent development, not described here) to "preset" (option **/LCM**) part of the connectivity matrix and work the GA only on the remaining part. By filling parts of the connectivity matrix and supplying the molecular formula, we can constrain the GA to use known actual fragments from a mass spectrum or complex functions (taking up more than one of our fragments, such as esters, or phenyl groups), evidence for which is taken from IR information.

6. CONCLUSIONS

These experiments contribute to show that the genetic algorithm is usefully applied to chemical problems, thanks to its very scope—anything that can be coded as zeroes and ones and the performance of which can be measured in some way can be made optimal. Also, no problem-specific operators are necessary, and regular crossover, mutations, and inversion do the job quite well. The conditions set, beyond the regular chemistry rules, can be as strict or as lax as needed. Implementation of "goodlists" and "badlists" is straightforward.

Regarding the specific problem we handled, it should also be noticed that **two-atom fragments are the worst** (the ones carrying the least information). Three-atom fragments would have better combinatorics; the ratio of chemically correct connectivity matrices to all possible connectivity matrices for a given list is lower, making the search space comparatively smaller. Matching actual mass spectrometry fragments by supplying a partially-filled table (see section 5) is comparatively easier.

Possible improvements are numerous. First of all, there is an urgent need to integrate **steric constraints** into fitness evaluation, to discourage monstrosities like a single bond between two *para* carbons on the dimethylaniline molecule or an epoxy group on a small cycle. The chemistry rules enforced at present by the fitness function do not consider these aspects, and users of the software will notice that it sometimes yields chemically lawful, yet sterically grotesque solutions. We are also currently investigating strategies for cycle inventory and **short cycle** avoidance.

Taking into account **hybridization states** (option **/SP**) and **number of hydrogens attached** would also be very helpful, to prevent terminal methyls where they should not be, hinder wrong-length cycles, etc. Most of the mistakes described above would not have been possible using such information.

Besides, we need a smarter and less literal way to use connectivity matrices: the phenyl radical should be treated as an "honorary halogen" or **superatom**, rather than letting the genetic algorithm toil its way through realization that six isolated fragments indeed are a phenyl. Experience with complex phenylated molecules, like etizolam (**/x6**) shows that the genetic algorithm is often stuck with a seven-atom, six-bond aromatic fragment that takes a lot of time to close. This explains the very low success rate for this compound—because it has to "find" two phenyls while all others have at most one.

Also, it would probably be smart to make the **encoding** more compact. As it stands, encoding consists of many spaces—there are many more zeroes (nonbonds) than bonds, and this gets worse with molecule size. A simple, run-length-style encoding would make the chromosomes much smaller and definitely facilitate exploration of the search space. The compactness gain would facilitate scaling up to larger molecules as we would have chromosome length of the order of N to the power of 1.2 or 1.5 (fractal dimension of a worst-case, very branched molecule) instead of the order of N squared (what we have with a dumb, literal encoding of the connectivity matrix as it is displayed like we are using now, that becomes exponentially emptier as molecule size increases).

We generally plan to keep optimizing the software while making it closer to the concerns and information available to chemists.

We also started working toward automatic graphical representation of connectivity matrices.

ACKNOWLEDGMENT

The author is deeply indebted to Profs. Jean-Pierre Doucet and Annick Panaye of ITODYS, University of Paris, for their invaluable help in preparing this paper. Thanks to Dr. William Fisanick for his detailed, helpful assessment of a previous version of this paper. Thanks to Dr. Jean-Christophe DORÉ from the Paris National History Museum for his invaluable help. GalvaStructures, the software described here, is available from PMSI on request, at no charge.

REFERENCES AND NOTES

(1) Spialter, L. The Atom Connectivity Matrix and its Characteristic Polynomial: A New Computer-oriented Chemical Nomenclature. *J. Am. Chem. Soc.* **1963**, *85*, 2012−2013.
(2) Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning; Addison-Wesley: 1989.
(3) Holland, J. H. Adaptation in Natural and Artificial Systems. University of Michigan Press: Ann Arbor, MI, U.S.A., 1975.
(4) Lucasius, C. B.; Kateman, G. Understanding and using genetic algorithms. *Chemometrics Intelligent Laboratory Systems* **1993**, *19*, 1−33 and **1994**, *25*, 99−145.
(5) Mestres, J.; Scuseria, G. Genetic Algorithms: A Robust Scheme for Geometry Optimizations and Global Minimum Structure Problems. *J. Comput. Chem.* **1995**, *16*,6, 729−742.
(6) Glen, R. C.; Payne, A. W. R. A genetic algorithm for the automated generation of molecules within constraints. *J. Computer-Aided Mol. Design* **1995**, *9*, 181−202.
(7) Venkatasubramanian, V.; Chan, K.; Caruthers, J. M. Evolutionary Design of Molecules with Desired Properties Using the Genetic Algorithm. *J. Chem. Inf. Comput. Sci.* **1995**, *35*, 188−195.
(8) Zupan, J.; Munk, M. E. Hierarchical Tree Based Storage, Retrieval, and Interpretation of Infrared Spectra. *Anal. Chem.* **1985**, *57*, 1609−1616.
(9) Woodruff, H. B.; Snelling, C. R.; Shelley, C. A.; Munk, M. E. Computer-Assisted Interpretation of Carbon-13 Nuclear Magnetic Resonance Spectra Applied to Structure Elucidation of Natural Products. *Anal. Chem.* **1977**, *49*, 13, Nov. 2075−2080.
(10) McLafferty, F. W. Interpretation of Mass Spectra; Science Books: Mills Valley, CA, U.S.A., 1980.
(11) Shelley, C. A. et al. Interactive Structure Elucidation, in Computer-Assisted Structure Elucidation. Smith, H. D., Ed.; ACS Symposium Series 54, **1977**, pp 92−125.
(12) Dubois, J.-E.; Carabedian, M.; Dagane, I. Computer-aided elucidation of structures by carbon-13 nuclear magnetic resonance. *Anal. Chim. Acta* **1984**, *158*, 217−233.
(13) Panaye, A.; Doucet, J.-P.; Cayzergues, P.; Carrier, G.; Mathieu, G. Structure elucidation: from spectral data to molecular recognition. *L'actualité chimique*; Apr−May 1988; pp 103−112.
(14) Rudolph, G. Convergence Analysis of Canonical Genetic Algorithm. *IEEE Trans. Neural Networks* **1994**, *5*,1, 96−101.
(15) Galvano-GalvaStructures User's Guide; PMSI: 1995.
(16) Srinivas, M.; Patnaik, L. M. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Trans. Systems, Man Cybernetics* **1994**, *24*,4, 656−667.
(17) CRC Handbook of Chemistry, 67th ed.

CI950132P