

Evolutionary optimisation: a tutorial

Ron Wehrens, Lutgarde M.C. Buydens*

Laboratory of Analytical Chemistry, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Evolutionary algorithms such as Genetic Algorithms and Evolutionary Strategies have gained wide popularity in the last decades. They have shown their ability to solve complex optimisation problems in a number of fields, including chemistry. In this tutorial, the main ideas are illustrated using a simple example, and general guidelines for the application of these algorithms are given. Pointers to available resources on the World Wide Web are given in the appendix. ©1998 Elsevier Science B.V.

Keywords: Global optimisation; Evolutionary strategy; Genetic algorithm

1. Introduction

Optimisation problems are abundant, in chemistry as well as in everyday life. For example, when packing a suitcase usually some consideration must be given to the way in which all items are put in, first of all to make sure that everything fits into the suitcase, and second to prevent clothes from being wrinkled and other items from being damaged. This is a typical optimisation problem: we know what situation we want to achieve (a neatly packed suitcase) but we do not know if it is possible at all, let alone how to do it. In chemistry, well-known optimisation problems include finding the molecular conformations with low energies, finding optimal reaction conditions or instrument settings, and selecting the most discriminating set of wavelengths in a multivariate calibration experiment.

Many optimisation problems are so complex that mathematical optimisation [1] or exhaustive enumeration of all possible solutions is not feasible. Especially local optima that may be present make it very hard to find the global optimum. The metaphor of a

mountain landscape is often used: the multitude of peaks makes it difficult to determine which one is actually the highest. In such a case, search or optimisation strategies must be used. The simplest strategy, usually not leading to very good results, is the random search. One just tries a few possibilities and selects the one that appeared best. Its only field of application is in problems of the needle-in-a-haystack class, where there is no evidence whatsoever of 'getting hot', or of a direction of gradually improving solutions. An alternative is the heuristic search, in which rules of thumb are used to guess a solution that is at least acceptable. The suitcase problem is an example where this kind of strategy is useful (we do not want to try many different options and ruin fifteen suits and three suitcases before we have a combination that we like). More often, strategies are used which start from a trial solution and through a small modification go on to a next solution, perhaps after a small assessment of the best direction. Methods like gradient descent and simulated annealing [2] belong to this category.

Of all the methods mentioned so far only simulated annealing can be expected to perform well in the complex problems mentioned earlier. It is able to escape from local optima by accepting moves to a (temporarily) worse state with a small probability, whereas the others are likely to find the peak nearest to the starting point (if any peak at all), instead of the highest. Another class of optimisation methods that can be applied to this kind of problem is formed by evolutionary algorithms such as Genetic Algorithms (GAs) [3–5] and Evolutionary Strategies (ESs) [6]. Their principal characteristic is that they consider populations of solutions rather than one solution at a time. By a reproduction process that is biased towards better solutions (hence the name) the next population is formed, containing new and hopefully better solutions.

Not only does population-based search have different search characteristics compared to individual-based search, providing a complementary approach

*Corresponding author.

to e.g. simulated annealing, it also has a number of other advantages. If one is interested in more than one solution, it is difficult to prevent individual-based approaches from finding the same optimum over and over again; in evolutionary optimisation one can force diversity in a population (see below). Another advantage, in some cases perhaps even more important, is that the machinery of the algorithm determines the next population of trial solutions. In individual-based approaches, the user has to define how to proceed from one state to the next. Especially in cases where there are 'forbidden areas' in the search space this may not be easy. Evolutionary algorithms have much simpler ways of coping with this kind of problem.

In the remainder of this tutorial, the main ideas behind evolutionary algorithms are presented, using a simple example as an illustration. The individual components of evolutionary algorithms are then described briefly. Finally, we list some applications where evolutionary algorithms have been successful and a few general guidelines on how and when to apply them. Details on available software, applications and papers are collected in Appendix A.

2. Example problem: Packing gas atoms into a box

Consider the situation where instead of fitting clothes and other things into a suitcase, we would like to put rare-gas atoms into a box in such a way that the energy of the ensemble is minimal. Let us assume for simplicity that we have two-dimensional atoms (circles) and a square box of adjustable size. The interaction energy of an ensemble of N atoms is given by

$$E_{\text{tot}} = \sum \left(Ae^{-aR} - \frac{C}{R^6} \right) \quad (1)$$

where the summation runs over all pairs of atoms, R is the distance between two atoms, and A , a and C are constants, depending on the nature of the gas atoms [7]. The energy curves for He-He pairs, Ne-Ne pairs, etcetera, are depicted in Fig. 1.

Since the optimal interaction between two atoms gives a negative contribution to the overall energy, the problem boils down to putting as many atoms as possible into the box, taking care that no atoms overlap. For a system consisting of one type of atom, this is achieved with the well-known hexagonal packing,

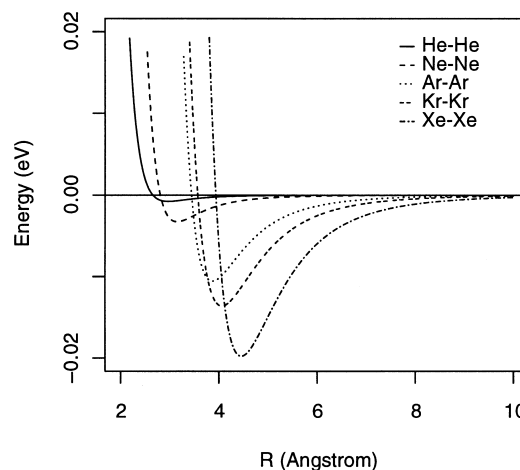


Fig. 1. Van der Waals energies of homogeneous pairs of rare-gas atoms.

where each atom is surrounded by six others. This problem already poses a considerable challenge to general-purpose optimisation algorithms, and it can be complicated further by including several types of gases in the box. In three dimensions, the problem is even more complex. Many more configurations are possible, and to go from one configuration to the next often requires a major reshuffling of several atoms. Evolutionary algorithms have been able to find all energy minima of clusters up to 75 atoms [8].

In this tutorial we consider a square box of size 20 Å. Atoms in the outer rim of 1 Å of the box are not considered in the energy calculations. Because they can be placed in that region without overlap penalties, this allows the algorithm to find out the maximum number of atoms in the inner space of 18 Å without the need to specify this number in advance.

A typical genetic algorithm (GA) to tackle this problem might proceed as follows. The first step in the optimisation is to randomly initialise a *population*, a set of trial solutions. Each of these consists of a set of coordinates for the atoms in the box and is represented as a *string* of numbers. If we try to fit in 40 atoms, the string therefore contains 80 parameters. The energy of a solution can be calculated in a process called *evaluation*. This is done by a function that calculates the energy of an ensemble of atoms according to Eq. 1, and in principle is the only part that is specific to the problem. To proceed to a next *generation*, parents are selected with a bias to the low-energy strings. This is the principle of the survival of the fittest, which gave evolutionary algorithms their name.

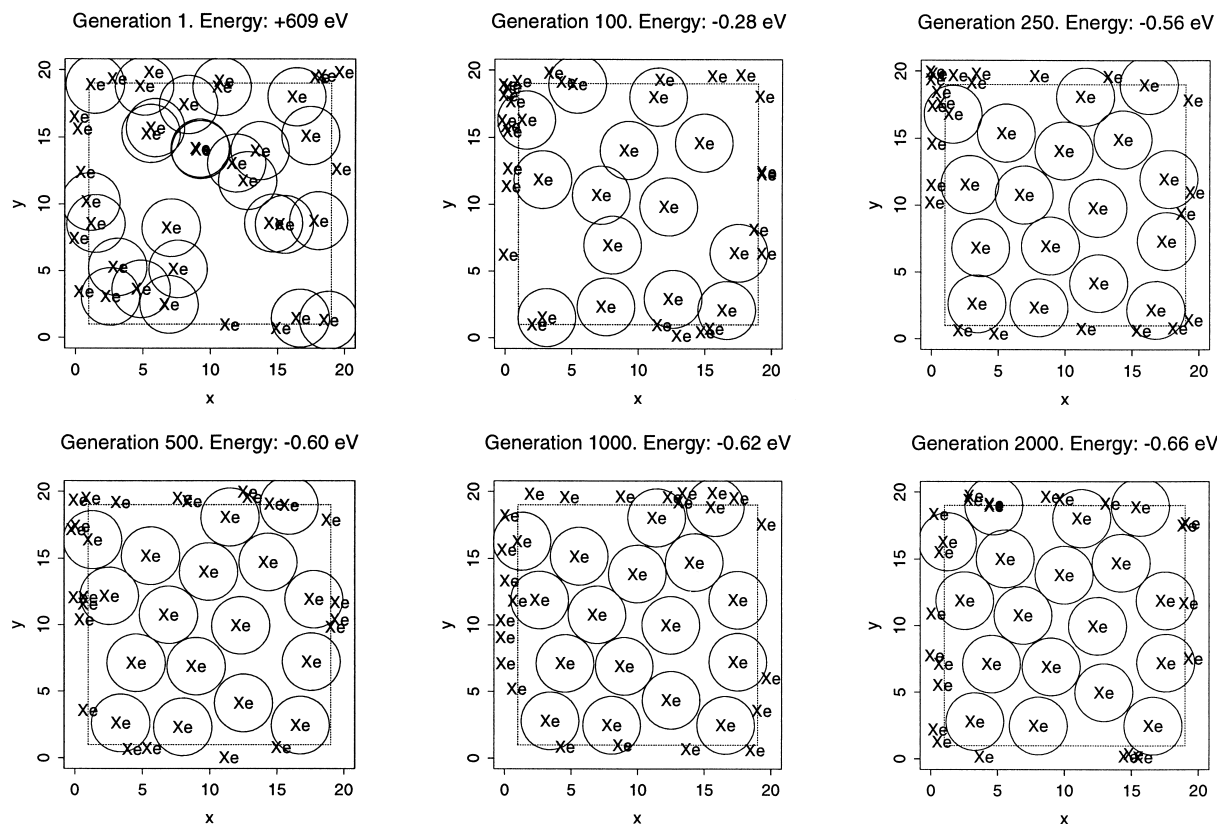


Fig. 2. The best configurations of 40 Xe atoms at different generations. A population of 30 strings is employed. The circles indicate the van der Waals radii of those atoms that actually contribute to the evaluation function, i.e. that are inside the outer rim of 1 Å. Atoms outside this region are indicated by their symbol only.

The next step is the sexual reproduction called *crossover*, in which two parents exchange genetic material, in this case coordinates of atoms. Thus, two children are created that are partly equal to their father and partly to their mother. This reproduction process continues until the next generation is filled

with children. Finally, random *mutations* are applied in which the value of one or more coordinates in a string are changed. Then, solutions in the new population are evaluated.

The cycle goes on until some termination criterion is met. In Fig. 2 the best solutions of different genera-

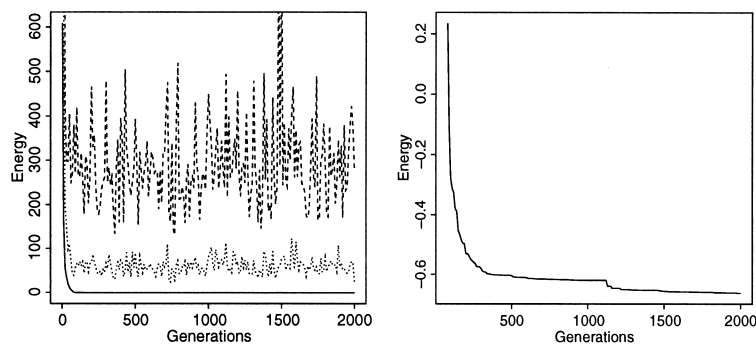


Fig. 3. The energies of the best (solid line), average (dotted line) and worst strings (dashed line) at each generation. The right plot zooms in on the low-energy part of the figure.

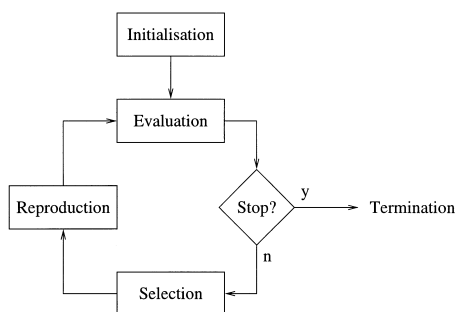


Fig. 4. The basic cycle of evolutionary algorithms.

tions are depicted. In this case 2000 generations were calculated. In the first generation, the randomly initialised solutions, there is clearly much overlap resulting in large positive energies. Already after 100 generations, the energy of the best solution in the population is negative. This is largely achieved by reducing the number of atoms in the inner region and thus avoiding overlap. During the remainder of the optimisation, more and more atoms are added. Still, the final solution is not perfect: one overlap is still visible and in the right half of the box the hexagonal packing is only roughly recognisable.

Often, it is not the best configurations in different generations that are plotted during an optimisation, but their energies (cf. Fig. 3). Clearly, most improvement is made in the first few generations. After that, improvement is slow. The variation in energies within a population is quite high. There does not seem to be a downward trend in the energies of the average and worst strings after approximately 100 generations, indicating that the diversity within the population is maintained throughout the optimisation.

3. Evolutionary algorithms: the machinery

The principles of evolutionary algorithms are easy to grasp and intuitively appealing, which may account for their recent popularity. Since the algorithms are problem-independent except for the evaluation function, toolboxes are available which contain all necessary constituents. Only the evaluation function has to be provided by the user. Simple application of standard settings and operators in many cases then suffices to find acceptable solutions. However, in more complicated applications performance may increase drastically if problem-specific characteristics are utilised.

There are many ways to do this at all levels of the algorithms.

In the next sections, a brief overview of the standard evolutionary operations in GAs and ESs as they are present in most toolboxes is given and the differences between the two main forms are highlighted. The basic cycle for evolutionary algorithms is depicted in Fig. 4 for easy reference.

3.1. Setup and initialisation

The purpose of the initialisation is to generate a population of trial solutions for the subsequent optimisation. This is usually done randomly, i.e. each member of the population is a random, and therefore usually very bad, solution to the problem. It is also possible to read the initial population from a file, or generate it otherwise, perhaps guided by prior information. Several aspects merit further inspection.

3.1.1. Representation of individuals

The first question is how to represent the trial solutions in the population. For our purposes, a string may be an array of real numbers. These are interpreted as coordinate pairs that must be optimised. It is also possible to divide the range for each parameter into discrete steps and use integers or even bits to indicate the locations of the atoms. This is illustrated in Fig. 5. Historically, ESs utilised real numbers while the GA community used bit representations. However, in the last few years this difference has largely disappeared.

The advantage of using real numbers is that any position in the box can be found by the algorithm, whereas using the other two representations only discrete points can be found. However, in many cases this is all that is required, and no efforts are wasted to

10.0	3.5	2.1	5.9
21	8	5	12
10101	01000	00101	01100

Fig. 5. Two pairs of coordinates, (10.0, 3.5) and (2.1, 5.9), in three different representations. Top: real values; middle: integer values (values between 0.0 and 0.5 indicated by the value 1, values between 0.5 and 1.0 indicated by 2, etcetera.); bottom: bits (each coordinate is represented by five bits, yielding 32 different values).

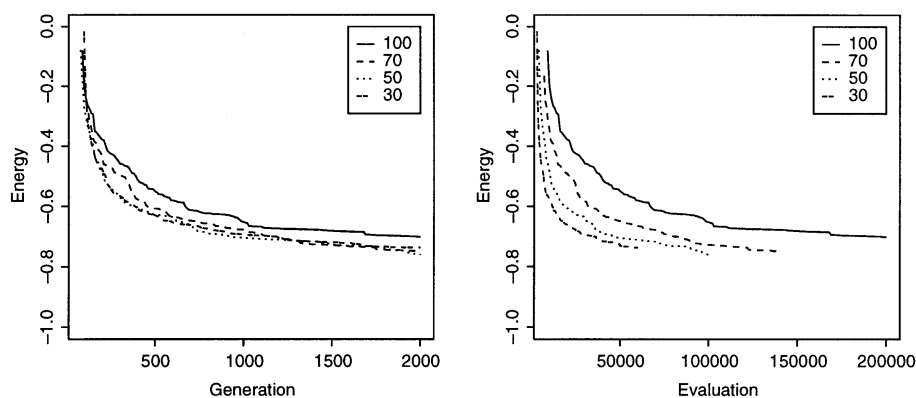


Fig. 6. Energy of the best string in the population during the optimisation for different population sizes. Left: against the number of generations. Right: against the number of evaluations. Plotted curves are the averages of five replicate runs.

provide unnecessarily precise answers. The latter two representations also use less computer memory. The drawback is the extra interpretation step that is required to convert integers or bits to meaningful values for the coordinates. The main advantage of using bits is that no checking is needed to see whether coordinates outside the box are generated during the optimisation. In the other two cases, mutation (see below) may introduce invalid values, e.g. coordinates outside the box region. Special care then has to be taken to avoid this. Although simple arrays are most often used as representations in evolutionary algorithms, there is no reason not to use other, perhaps more complicated structures.

The most important difference between the representations is that they define different search space topologies. If we represent positions in the box by real numbers, then the distance between two points is simply the Euclidean distance. A large distance indicates that two points are far from each other. If we use an integer representation where each point is represented by one number, however, two points with very different numbers may still be located quite close to each other. This affects the search characteristics of evolutionary algorithms profoundly.

A significant difference between GAs and ESs is that the latter also code a set of strategic parameters which determine the character of the search. As an example, consider the situation where a new individual is created by adding a random number to one of its parameters (mutation, see below). The standard deviation of the distribution from which the random number is drawn may be one of these strategic parameters. One could envisage a large standard deviation in the beginning (coarse search) and a small one at the end of the optimisation (fine search). This means that the

search itself is optimised on-line. Note that the snake may bite its own tail: bad search settings may prevent good search settings from ever being found.

3.1.2. The population size

The population size is determined by the user. Sizes of 20–500 strings are common. In general, the more parameters are being optimised, the larger the populations although no clear-cut rules are available. Often, there is a limit to the number of evaluations (i.e. population size times number of generations) that can be carried out realistically. In such a situation, one must choose between a large number of generations and a small population size, or vice versa. In cases where one is interested in many 'good' solutions instead of one single 'best' solution, it may even be better to use repeated short runs with small populations.

This trade-off is illustrated in Fig. 6. In this case, the average energies of the best configurations of five repeated runs are plotted. The quality of the final solutions is not too different, but the runs with the smaller population use far fewer evaluations. The lowest average energy is found with a population size of 50 strings.

Care must be taken not to base conclusions on one single run. Because of the stochastic nature of the search, it is preferable to look at a mean value of different runs. Indeed, in the first of these repeated runs the opposite picture was found: there the largest population seemed to give the best results. The positions of the atoms in the box as indicated by the best solution of these first runs are plotted in Fig. 7. The better energy of the run with the larger population is the result of the larger number of atoms (22) in the region where the energies are evaluated. The smallest population largely manages to prevent overlap but places four

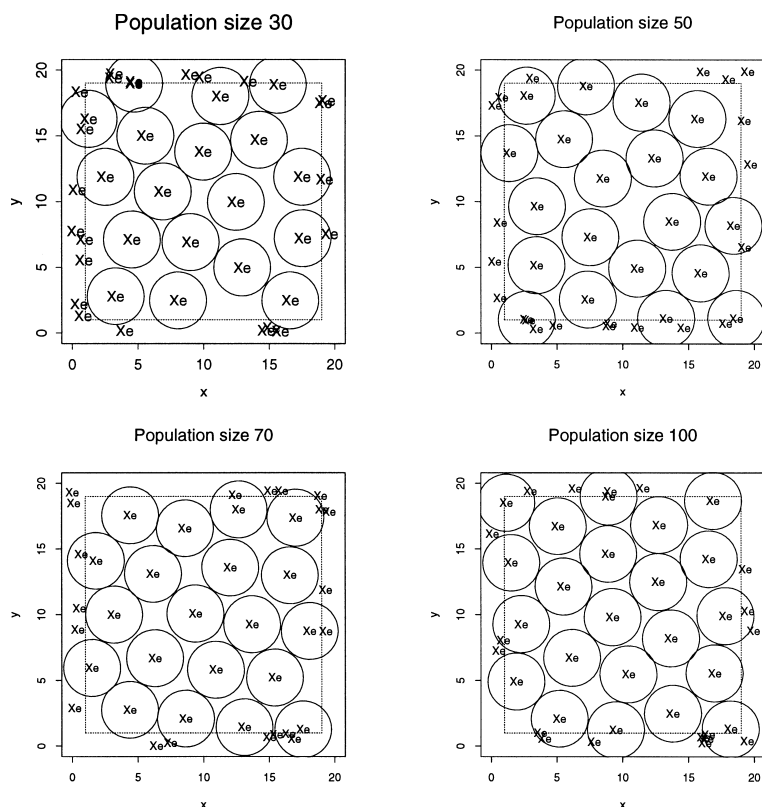


Fig. 7. Best configurations of 40 Xe atoms in a 20 Å box as found by GAs with populations of 30, 50, 70 and 100 strings (2000 generations), respectively. Energy of these solutions: -0.66 , -0.76 , -0.84 and -0.95 eV.

atoms less in the box. The other population sizes both place 20 atoms in the box and differ only in the degree of overlap and the distances between atom centres.

3.2. Evaluation

In the evaluation function, the numbers on the strings get their meaning. If we wanted to optimise coordinates in a three-dimensional box instead of a two-dimensional one, we could use exactly the same representation as in the example. However, then the third number would not mean the first coordinate of the second atom but rather the third coordinate of the first. As mentioned already, this is the fundamental reason why evolutionary algorithms are so general: it is only the evaluation function that makes the actual link with the problem. The rest of the algorithm is problem-independent.

To avoid confusion between minimisation and maximisation problems, often the term *fitness* is used. By definition, better solutions have a higher fitness. In many minimisation problems, the fitness is the inverse of the criterion that is minimised. Sometimes the

fitness values are scaled in such a way that the best solution in the population has a fitness near 1, and the worst has a fitness near 0. Another application of the scaling function is that differences between, say, very good and very, very good strings can be emphasised. In cases where the majority of the strings in the population are quite good and a few bad strings are present, this can be an advantage.

The evaluation function is usually the most time-consuming part of the optimisation process. Since in evolutionary optimisation thousands of evaluations are performed, significant speed gains may be obtained by careful design.

3.3. The termination criterion

At some point, the optimisation cycle must be stopped. One reason for this might be that the optimal solution is found. This is sometimes recognisable from the value of the evaluation function. When fitting a function, an error value of zero for the fit is ideal and cannot be further improved. More often, if no improvement has been observed during a certain num-

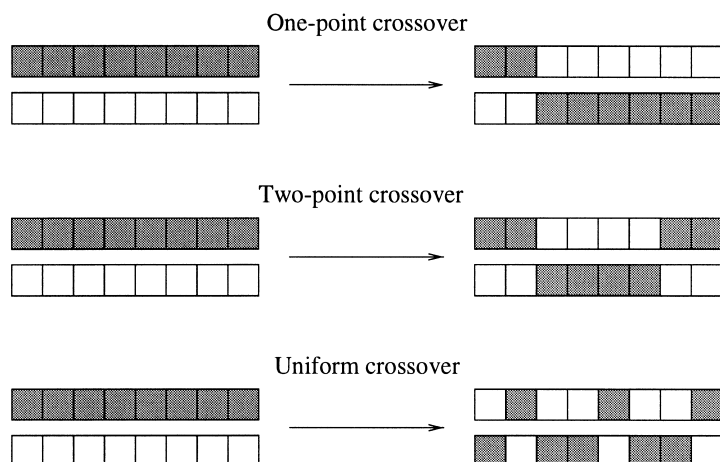


Fig. 8. Crossover schemes.

ber of generations, this is taken to be an indication that the optimum has been reached. A final possibility is to do only a specific number of generations. In practice, this last termination criterion is the one that is used most.

3.4. Selection

The main function of the selection step is to prevent bad solutions from producing offspring. Several selection mechanisms are available. The ones most often used are roulette selection, tournament selection and rank-based selection. In roulette selection, the chance of being selected is proportional to the fitness of a string. In tournament selection, repeatedly the best of a small, randomly picked subset of strings from the current population is selected. If the size of the subset is two, we speak of binary tournament selection. Finally, in rank-based selection the strings in the population are sorted in order of fitness value; from a user-defined threshold onwards, say the best 15%, strings are selected randomly. Thus, good strings will probably be selected multiple times and have a large effect on the composition of the next generation, while bad strings will eventually die out.

3.5. Reproduction

The aim of the reproduction step is to produce a new generation from the parents that were selected from the old generation. Two types of reproduction exist: sexual reproduction, in which parts of the parents are exchanged to produce two children, and asexual reproduction, in which each parent undergoes some form of mutation to produce a child very much like itself. The

former is called crossover and produces children that in general are different from both parents but still contain large parts of each. Historically, Evolutionary Strategies did not use crossover and relied completely on mutation. Both genetic operators are discussed briefly below.

3.5.1. Crossover

The crossover has always been regarded as the main operator in genetic algorithms, although recently its importance has been disputed. Its purpose is to combine *building blocks*, parts of the parent strings that correspond to favourable subsolutions. It is regulated by a user-settable parameter, the crossover probability. Usually, this is chosen in the region 0.5–0.9; it indicates that for a selected pair of parents there is a chance of crossover between 50 and 90%. If no crossover is performed, the children are equal to the parents.

The simplest form is the one-point crossover, which randomly selects a break point and exchanges all parameter values on one side of the break. A theoretical objection is that two parameters that are located at the end points of the string will never be propagated to the child together. Thus, two-point and multi-point crossovers are devised. The limiting form is the uniform crossover, where for each number on the string a coin is tossed to determine whether to exchange values or not¹. The different crossover types are illustrated in Fig. 8.

¹ A chance of 0.5 maximises the amount of mixing of the two strings. A lower or higher chance keeps more of the individual strings.

Whereas with real- and integer-coded solutions the crossover points are always between two parameters, with binary-coded strings this need not be the case. Since boundaries between parameters are only relevant in the evaluation phase, crossover points may be selected anywhere at the bit level. This means that a crossover point within a parameter also introduces a mutation (or rather, two), since the values on the child string, when interpreted in the evaluation function, are different from the values of both parents. Of course, crossover points may be forced to coincide with parameter boundaries.

In some cases, more complicated forms of crossover are used. This is usually a result of the chosen representation of the trial solutions or peculiarities of the problem. For instance, in the problem of wavelength selection one tries to find a small set of wavelengths that contains as much spectroscopic information as possible, usually almost as much as in the complete spectrum. When using evolutionary algorithms with crossover, one has to take care that the same wavelength is not selected twice. Special operators for these and other problems have been developed [9,10].

3.5.2. Mutation

The main task of mutation is to provide new solutions that cannot be generated otherwise. It introduces an element of random search (sometimes termed *exploration*), where the selection and crossover processes focus attention on promising regions of the search space (*exploitation*). Again, the occurrence of this operator is determined by a user-settable parameter, the mutation probability. This probability is usually much lower than the crossover probability to prevent too much random search. Values of 0.001–0.05 are common.

In the case of binary strings, a mutation may be the flipping of a randomly chosen bit. In the case of integer- or real-coded strings, it may consist in replacing a number on the string by a new random value within the permissible range, or adding a random value from some distribution to that number. In the latter case care must be taken to map the new value back into the permissible range, if necessary.

3.5.3. GAs versus ESs

In GAs usually a complete population is replaced by the next generation. Another possibility is to replace one or two strings after each reproduction step (called *steady-state* replacement). In general, the population

size is constant. With ESs, several other variants are possible. In so-called $(\mu+\lambda)$ ES μ parents generate λ offspring (where $\lambda > \mu$) and the next generation is selected from both parents and children. In (μ, λ) ES, however, selection is only performed from the children [11].

3.6. Extensions to the basic scheme

Numerous extensions are possible to this basic scheme. First of all, the main problem with an ill-configured evolutionary algorithm is premature convergence: one very good individual has dominated the population to such an extent that all members of the population are very much alike. It is clear that this is an undesirable situation. Several special operators are developed to prevent this, such as *sharing* in which the fitness of an individual is lowered if it is too much like another individual, forced mutations if this is the case, or *crowding*, where an individual is not inserted randomly in the next generation but replaces the individual which is most like itself. Also parallelisation can be applied for this purpose: instead of one big population several small (sub)populations can be maintained which may at some timepoints exchange genetic material. Because the influence of a very good individual then remains limited to one subpopulation the chance of premature convergence is decreased.

If the best strings in a population are copied unchanged to the next, this is called *elitism*. It makes sure that no valuable information is lost. If elitism is applied, the best string in the last generation is by definition also the best string encountered during the optimisation.

A further important possibility is sometimes termed *hybridisation*: it usually consists of a quick-and-dirty local optimisation for each trial solution in the evaluation phase. In the atoms-in-a-box example, one could envisage this as follows. Instead of an evaluation function returning the energy of an ensemble of atoms at the given positions, one could start a steepest-descent optimisation from this configuration for a few steps and return the energy of the new configuration. The coordinates of this new configuration may be back-coded on the string: in that case the evaluation not only calculates a fitness measure but also changes the values coded on the string. Since evolutionary algorithms are usually not very precise in their assessment of the global optimum, local optimisation of the final solution is always a good idea.

Old population:

0	0.417	12.4	2.1	19.3	0.5	5.7	9.2	4.6	6.2	18.9	0.6
1	-0.000	13.8	0.9	1.7	19.6	9.6	1.3	18.4	11.7	19.4	3.6
2	-0.003	7.6	3.7	9.4	11.7	18.4	4.4	12.3	18.1	0.9	4.9
3	-0.019	10.7	16.5	1.2	16.9	4.9	10.0	10.2	10.5	8.5	19.0
4	0.505	16.9	6.4	0.7	17.0	15.4	5.7	6.6	18.2	11.6	12.2
5	3.600	8.1	15.3	1.4	0.3	18.2	8.7	14.7	7.1	18.9	11.1
6	12.131	13.1	17.4	13.4	8.7	15.9	13.7	1.2	0.2	15.5	9.4
7	-0.010	19.3	13.1	2.5	13.0	12.0	3.1	16.9	15.1	17.0	5.3
8	99.414	4.4	8.6	1.8	3.8	14.2	4.9	4.0	9.9	1.0	14.0
9	0.462	16.6	12.8	15.7	1.3	17.6	9.9	10.8	3.4	4.4	3.2

Selected parent pairs by binary tournament selection (mother - father):

(**6** 3), (**1** 3), (**2** 7), (**3** 2), (**1** 9)

New population:

0	-0.029	13.1	17.4	13.4	8.7	15.9	13.7	1.2	0.2	8.5	19.0
1	-0.031	10.7	16.5	1.2	16.9	4.9	10.0	10.2	10.5	15.5	9.4
2	-0.000	13.8	0.9	1.7	19.6	9.6	1.3	18.4	10.5	8.5	19.0
3	-0.028	10.7	16.5	1.2	16.9	4.9	10.0	10.2	11.7	0.9	3.6
4	2.724	7.6	3.7	9.4	11.7	18.4	3.1	16.9	15.1	17.0	5.3
5	-0.000	19.3	13.1	2.5	13.0	12.0	4.4	12.3	18.1	0.9	4.9
6	9.416	10.7	16.5	1.2	16.9	4.9	10.0	12.3	18.1	0.9	4.9
7	71.06	7.6	3.7	9.4	11.7	18.4	4.4	10.2	10.5	8.5	19.0
8	-0.010	13.8	0.9	1.7	19.6	9.6	1.3	18.4	3.4	4.4	3.2
9	0.470	16.6	12.8	15.7	1.3	17.6	9.9	10.8	11.7	19.4	3.6

Fig. 9. The basic element of the GA cycle. A population of 10 strings is evaluated, and undergoes selection, one-point crossover and mutation to yield the new population. The first number is the string index and the second number is the result of the evaluation function, i.e. the energy of the configuration. The mutation location is underlined. For easy retracing, father string numbers and values inherited from the father strings are indicated in normal face, while mother string numbers and values inherited from them are in bold face.

3.7. Numerical example

A numerical example comprising selection, crossover and mutation using 10 strings is depicted in Fig. 9. To simplify tracing the crossover and mutation operators, mother strings are indicated in bold face. Internally there is no difference between both parent strings. Note that the quality of the new population is significantly better than that of the first.

4. The application of evolutionary algorithms

Many reviews of chemical applications of evolutionary algorithms have been published (see, e.g., [9,10,12,13]). GAs are more popular in chemistry, when looking at the number of papers, although there does not seem to be a deep reason for this.

Although it is not too difficult to program an evolutionary optimisation method from scratch, toolboxes are often used. They already provide many of the standard operators and the basic tools for manipulating populations. The only thing that should be programmed is an evaluation function (which may be quite complicated in itself). Still, the user has a lot of freedom in deciding which operators to use and in what way. With most toolboxes, standard settings are indicated and these may work very well with many problems. However, when they do not, it can be difficult to find settings that do work, since no general strategy exists to tune evolutionary algorithms. In general it is beneficial to include as much problem-specific knowledge as possible without sacrificing the simplicity of the representation.

The question now is when to use evolutionary algorithms. Obviously, their most important distinction is the use of a population. This provides possibilities to

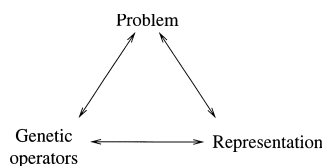


Fig. 10. The relation between problem, representation and genetic operators that are used determines the success of the optimisation.

prevent multiple solutions that are too much alike, something that may be hard to achieve when using, e.g., 50 parallel runs of simulated annealing. This makes evolutionary algorithms attractive for fields like molecular modelling, where all chemically relevant structures should be found.

A second area in which the use of a population can be advantageous is a class of problems in which there are 'forbidden areas', i.e. areas that are known not to contain viable solutions. An example of such problems is the optimisation of molecular structures. To go from one conformation to another one sometimes has to climb a very steep energy barrier, e.g. when two groups of atoms are very close or one group has to 'cross' another group. When using individual-based search, there is usually little choice but to go around these areas, which in practice may be very difficult. In population-based search, especially when the crossover is applied, much larger jumps through conformational space are possible. Moreover, solutions that are really bad can and will be excluded from the mating process without many further consequences. Therefore the forbidden areas do not have as large an impact. Whereas in individual-based search the representation is of minor importance and the main point is how to go from one solution to the next, in genetic algorithms the situation is reversed. The algorithm takes care of how to get a new trial solution and it all works when the representation is right. This dependence between problem, representation and evolutionary operators is depicted in Fig. 10.

A final field where evolutionary algorithms may be applied with success is the area of multicriteria optimisation. Often, an evaluation function combines several criteria into a single number. It is also possible to use the original set of criteria to rank the solutions in a population, something that is much more difficult when one considers individual-based search. This has the advantage that no explicit combination of criteria is necessary, something that can be difficult when they are of a different nature.

5. Conclusions

It has been nearly a decade since the first applications of evolutionary algorithms in chemistry appeared in print. The number of papers on the subject since then has shown a dramatic increase and application to a broad range of chemistry problems has been reported. One of the reasons might be the appealing familiarity of many of the concepts behind evolutionary algorithms. The survival of the fittest, selection, random mutation, parents and offspring, all are easily explained to others. Furthermore, the results often prove that it all actually works, and that these algorithms can be applied in a large range of chemical optimisation problems. Because of their inherently different search characteristics they form a complementary approach to individual-based searches. Finally, evolutionary problems are not too difficult to program. In fact, when using a standard toolbox all one needs is a function that accepts a trial solution and returns a measure for the quality of that solution.

Acknowledgements

The authors thank A. Dane for constructive comments on a draft version of the manuscript.

Appendix

Software and resources

All programs used in this paper were written in ANSI C using the toolbox PGAPACK and run on UNIX workstations. PGAPACK can be obtained by anonymous ftp from

```
ftp://ftp.mcs.ano.gov/pub/pgapack/pgapack.tar.Z.
```

The source code for the examples in this paper can be obtained from the Internet at the address

```
http://www-sci.sci.kun.nl/cac/rwehrens.
```

A wealth of information on evolutionary algorithms can be obtained from the Internet. The newsgroup comp.ai.genetic contains monthly postings of FAQs (Frequently Asked Questions) and features lively discussions of users. The ftp addresses of the ENCORE Evolutionary Computation repository are

ftp.germany.eu.net:/pub/
research/softcomp/EC
and
alife.santafe.edu:/pub/
USER_AREA/EC

for Europe and the United States, respectively. Both software and papers, including a huge database (2500 references) on Evolutionary Computation, can be retrieved from them. For subscription to the GA mailing list send a message to

ga-list-request@aic.nrl.navy.mil.

Searching on the Web will no doubt yield many more relevant sites.

Glossary

population	number of trial solutions
string	trial solution
evaluation	assessment of the quality of the trial solution
fitness	(scaled) measure of that quality
generation	population at some point during the search
crossover	exchange of information between trial solutions
mutation	random change of trial solution
building blocks	information-containing parts of trial solution
sharing	decrease in fitness if trial solutions are too much alike
crowding	non-random insertion in the population of a new trial solution
elitism	keeping the best solution(s) so far for the next generation
hybridisation	combination with other (local) search techniques

References

- [1] T. Schlick, in K.B. Lipkowitz and D.B. Boyd (Editors), *Reviews in Computational Chemistry*, Vol. 3, Chapter 1. VCH Publishers, New York, 1992.
- [2] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, *Science* 220 (1983) 671.
- [3] D.E. Goldberg, *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, New York, 1989.
- [4] L. Davis (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
- [5] J.H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1992.
- [6] I. Rechenberg, *Evolutionsstrategie. Optimierungsstrategie technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzberg, Stuttgart, 1973.
- [7] M. Karplus and R.N. Porter, *Atoms and Molecules*. Benjamin Cummings, London, 1970.
- [8] D.M. Deaven, N. Tit, J.R. Morris, K.M. Ho, *Chem. Phys. Lett.* 256 (1996) 195–200.
- [9] C.B. Lucasius, G. Kateman, *Chemometr. Intell. Lab. Syst.* 19 (1993) 1–33.
- [10] C.B. Lucasius, G. Kateman, *Chemometr. Intell. Lab. Syst.* 25 (1994) 99–145.
- [11] T. Bäck, F. Hoffmeister and H.-P. Schwefel, in R.K. Belew and L.B. Booker (Editors), *Proc. Fourth Int. Conf. Genetic Algorithms*, 1991.
- [12] D.B. Hibbert, *Chemometr. Intell. Lab. Syst.* 19 (1993) 277–293.
- [13] D.E. Clark, D.R. Westhead, *J. Computer-Aided Mol. Design* 10 (1996) 337–358.

Ron Wehrens obtained his Ph.D. at the University of Nijmegen (The Netherlands), and Lutgarde Buydens at the Free University of Brussels (Belgium). Both are now at the Laboratory of Analytical Chemistry at the University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands.

Internet: <http://www-cac.sci.kun.nl/cac>

TrAC/Internet column

In order to inform analytical chemists about the Internet and the role it could play in their lives, Dr. Stephen Heller was invited to become a Contributing Editor of TrAC. The Internet Column has now become a feature of the journal. The column can also be found on the World Wide Web.

Anyone interested in contributing to this column is invited to contact Michael Guilhaus at:

M.Guilhaus@unsw.edu.au

The Internet Column articles of TrAC can also be found on the Web. If you have a browser, to access the TrAC column on the Web simply point to:

<http://www.elsevier.nl/locate/trac>