Journal of Thermal Analysis and Calorimetry, Vol. 55 (1999) 925-947

DEVELOPMENT OF GENETIC ALGORITHMS IN ORDER TO PERFORM DATA PROCESSING OF FLAME TEMPERATURES

D. Brunel and L. Elégant*

Laboratoire de Thermodynamique Expérimentale, Université de Nice, BP 71, 06108 Nice Cedex 2, France

(Received January 17, 1998; in revised form June 4, 1998)

Abstract

A program is described which enables performing of genetic algorithms for the determination of two positive real parameters. These new types of procedures are tested on a software of determination of flame temperatures previously developed in a fully classic way. The genetic operators used are crossover and mutation. They perform operations on a binary coded form of the parameters. The goal of the present study consists in developing and optimizing a genetic determination of the parameters at a given temperature. We succeed in selecting the general architecture of the procedure and implementing it in our main software of calculation of flame temperature. We have chosen this pyrotechnic field of application because we knew the behaviour of the real parameters, so the debugging operations were easier.

Keywords: binary coding parameters, calculation of flame temperature, computer program, crossover, data processing, dissociative reactions, genetic algorithms, genetic operators, pyrotechnic reactions

Introduction

In the thermal analysis and calorimetry some algorithmic procedures are frequently used in calculations for the determination of real parameters. The strategies used for these calculations can be more or less classic (dichotomizing search, direct mathematic resolution...), so more or less suitable for the problem studied.

In some cases it is very interesting to use less academic methods. In previous studies we showed the great interest in using Artificial Neural Networks (ANNs) in the determination of kinetic parameters in Differential Scanning Calorimetry (DSC) [1, 2]. This is a direct application of Artificial Intelligence (AI). Another use of AI is represented by Genetic Algorithms (GAs).

* Author for correspondence: e-mail: brunel@unice.fr

1418–2874/99/ $ 5.00

Akadémiai Kiadó, Budapest
Kluwer Academic Publishers, Dordrecht

There are more and more fundamental studies on GAs; their real capacities are not fully appraised, contrary to neural networks, whose principles of application are now better known. We have not found publications describing the use of GAs in the domain of DSC. The final goal of our current activities will be in two steps.

1. Define, realize and set GAs adjusted for the determination of kinetic parameters issued from physical transitions in DSC.
2. Compare and appraise the efficiency of GAs and ANNs.

Nevertheless, in order to perform and control the behaviour of our GAs, we decided to test them on a problem concerning the estimation of parameters whose characteristics are well known and similar to those above. The present work describes the work involved in this study.

We chose to include genetic procedures in a software of calculations of flame temperatures of pyrotechnic reactions. An initial release was developed some years ago for an industrial society. The method chosen to perform the calculations of flame temperatures requires at a given time the determination of 2 parameters which have a well known behaviour.

In GAs there are no systematic methods for obtaining good solutions, but rather a set of solutions which are explored and appraised simultaneously. The production of this set of solutions requires a systematic approach. GAs can be considered as a parallel approach to the resolution of a given problem. From this point of view the methods of functioning involved by GAs are a specificity of the evolutionist domain.

Method for flame temperature data processing

There are several methods of performing such a calculation. They have different levels in complexity and ability. All of these methods use the theoretical equation of the reaction. The most modern adiabatic methods are founded on the minimization of the global Gibbs's energy and calculations of enthalpy balances for the system. Unfortunately there are not many references on softwares using this approach, we have noted that it is an optional possibility in more general softwares dealing with chemical equilibria. The features for these softwares (such as: Chemsage, Thermodata, Equilib, Thermo-Calc...) can be freely downloaded on the Internet network by using a search engine such as Alta Vista or Yahoo.

For the specific kind of reactions that we had to compute, we were given the following goals:

– Combustion of hydrocarbons with possible inclusion of some materials.
– Creation of a database of thermodynamic properties for the whole species involved in pyrotechnics (about 250).
– Taking into account the dissociation of CO, CO_2, NO, NO_2, H_2, H_2O from 298 up to 5000 K.

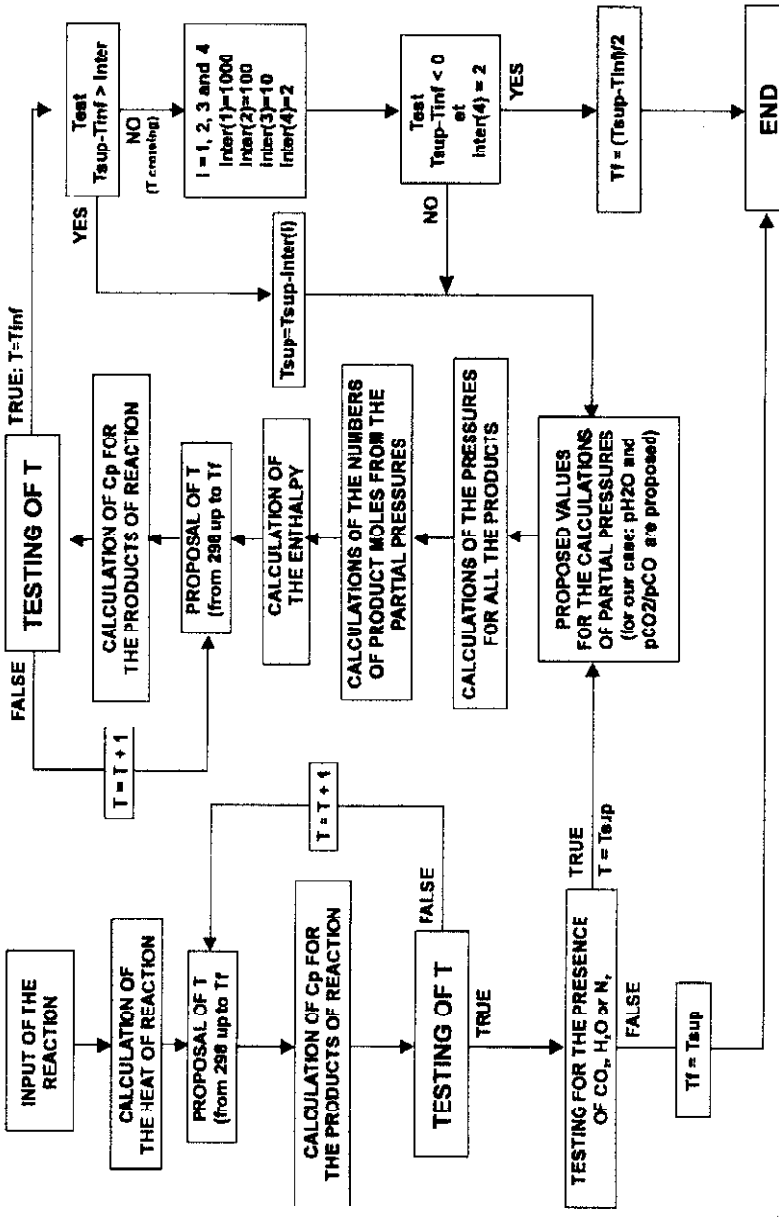

Fig. 1 General procedure for our calculations of flame temperature

For all these reasons we decided to develop a very simple method founded on the manual method of Damkohler and Edse [3, 4]. The original features of the improvements brought by our work were very satisfactory in comparison with the requirements.

The basic principle of the calculations consists in making the heat of the reaction equal to the global heat capacities of the products of the reaction. We will not give details of the calculations in this paper [3]. However, Fig. 1 presents a general view of the procedure:

The studied reaction

In order to test our genetic algorithms we have chosen the combustion of heptane in oxygen. The theoretical equation is:

$$C_7H_{16} + 11O_2 \rightarrow 7CO_2 + 8H_2O$$

In fact the species such as CO_2 and H_2O are dissociating and giving species such as O_2, O, H_2, H, OH, H_2O, CO and CO_2. The software knows the equilibrium constants of the possible dissociations between 298 and 5000 K and it is able to compute the partial pressures for the reaction products and thus the corresponding real mole amounts of dissociated species. For that procedure the values of some parameters must be set. In our example these values are the partial pressure pH_2O and the ratio pCO_2/pCO. In order to check the computed partial pressures two tests must be validated. For the above reaction the tests will be:

i) The sum of the computed partial pressures is equal to 1 (atmospheric pressure).

ii) The ratio between the computed number of oxygen and hydrogen atoms (nO/nH) is equal to the value given by user (1.375 for this reaction).

In the classic method of calculation initially developed the values of the two parameters were proposed systematically.

If these two relations are realized simultaneously, this will mean that the calculated partial pressures are true and that the values proposed by the software were right. For the reaction above the flame-temperature is 3138 K, the pressure pH_2O is equal to 0.269268 and the ratio pCO_2/pCO is equal to 0.589490. The one problem is in fact to be able to propose the right values for these two parameters. In order to better realise this problem we will represent as an example in a three-dimensional space (x, y, z) the four quantities pH_2O, pCO_2/pCO, nO/nH and P. The partial pressure pH_2O is put on the X axis, the ratio pCO_2/pCO on the Y axis and the values of nO/nH as well as those of P on the Z axis (Fig. 2).

The sets of values of pressures P and ratios nO/nH determine two distinct areas above the plane (x, y). We drew a small part of it on the above picture (hatched area). These two areas will cut the planes $z=P=P_t$ (P_t: total pressure 1 atm) and

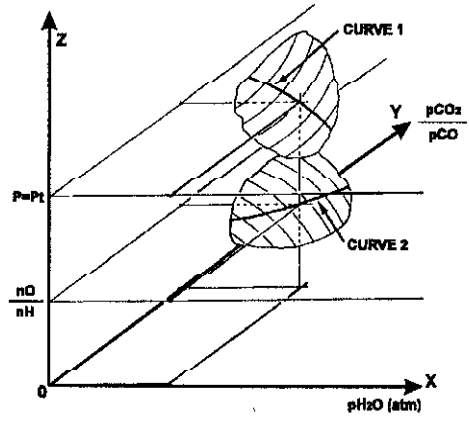

Fig. 2 Three-dimensional view of calculations on partial pressures and nO/nH

z=nO/nH (for our example nO/nH=1.375) giving two curves (curve 1 and 2) which will pass one above the other at a given time. This inevitable intersection point corresponds to the real values of pH$_2$O and pCO$_2$/pCO.

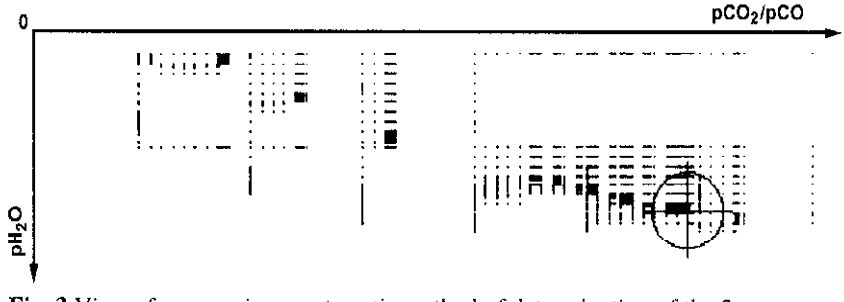

Fig. 3 View of our previous systematic method of determination of the 2 parameters

Our initial procedure scanned the plane $x0y$ by proposing incremental values for pH$_2$O and pCO$_2$/pCO according to a systematic methodology. The values of the increments decreased owing to the error computed for the above i) and ii) tests. This method was described in our previous work [3]. Figure 3 shows the development of this systematic methodology. The right couple pH$_2$O, pCO$_2$/pCO is indicated with an encircled cross.

The genetic approach

It is the above systematic methodology of determination of pH$_2$O and pCO$_2$/pCO which is now replaced by GAs that we are going to describe below. Basically GAs will run some processes of generation on a coded form of the above two parameters in order to obtain new parameters. The mechanisms for the

generation of the parameters draw one's inspiration from biologic genetics processes for the duplication of chromosomes. This explains the designation 'Genetic Algorithms' [5–9].

First, the GAs are going to code in an appropriate form ('pattern') the real values of the parameters. Secondly they must select into a set of coded parameters (we said 'population') some couples of them (so-called 'parents'), which will be subjected to some genetic processes (usually 'crossover' and 'mutation'). The new parameters obtained (the 'children') replace their corresponding parents in the population. The selection of the parents for the crossover is conducted as a function of the error obtained for the calculations of the total P pressure and the ratio pCO_2/pCO. The weaker the errors on these calculations, the greater the chance of the corresponding parameters of being selected. Thirdly, in order to avoid the disappearance of certain patterns, some genetic mutations can be performed on the whole population before starting calculations.

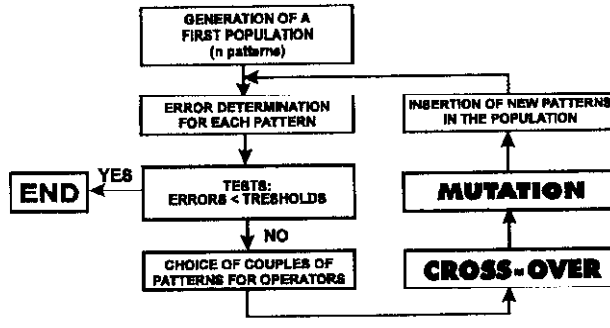

Fig. 4 Global chart for genetic processes

The totality of these processes represents a cycle, known as a 'generation', repeated until errors lower than thresholds given by user are obtained. Figure 4 shows the global chart of these processes.

Coding methods adapted to the problem studied

As we must work on an adapted form of the parameters, it is usual to code them as a binary pattern, so we have chosen this method. However we specify that it is not the only way to run genetic operators. This coding must be reversible and computerized. Basically the binary coding consists in transforming each real numeric value framed by a given interval, into a corresponding binary chain.

In a binary coding the lengths of the chains can be constant or not. For our approach we chose constant length which are easier to run. For current and future values of the parameters studied, we have framed the interval of values between –99 and +99, with an effective accuracy of 10^{-10}. So we can code any value from –99.9999999999 to +99.9999999999. This interval is oversized. The global number of distinct values which must be coded is given by:

$$r = 2I/10^d \tag{1}$$

where r: number of possible real values, I: width sought for of the coding interval (for us: 99), d: accuracy sought for (for us: 10).

Then our specifications will give $r=1.98 \cdot 10^{12}$ distinct real potential values. In a classical problem of binary coding we have the following scheme widely used in computer science (Fig. 5):

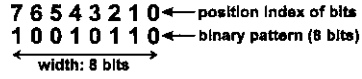

Fig. 5 Locating of bits in a binary pattern

The width of the binary pattern is given by the very simple relation:

$$w = i + 1 \tag{2}$$

where w: width of the binary pattern, i: position index of the respective bits in the pattern.

Then we can set the width of binary patterns with the relation:

$$2^i = r \tag{3}$$

$$i = \log_2 r = \frac{\log_{10} r}{\log_{10} 2} \cong 40.85 \tag{4}$$

then with Eq. (2)

$$w = 41.85 \tag{5}$$

Then we can round w to 42. This means that 42-bits-chains long can be used to code any value as defined with the relation (1). So as i=41 we can code up to $2^{41}=2.199 \cdot 10^{12}$ different values. We have developed procedures to perform decoding between real and binary patterns. As the values of the parameters are strictly positive we decide to shift the coding interval from [–99, +99] to [0, 198] with exactly the same amount of binary patterns generated. In Table 1 we show some real and corresponding binary values calculated by the coding procedure.

Table 1 Real and binary corresponding values

	Real value	Corresponding binary pattern
1	0.0000000000	0000000000 0000000000 0000000000 0000000000 00
2	0.0000000001	0000000000 0000000000 0000000000 0000000000 01
3	99.0000000000	0100000000 0000000000 0000000000 0000000000 00
4	197.9999999999	0111111111 1111111111 1111111111 1111111111 11
5	198.0000000000	1000000000 0000000000 0000000000 0000000000 00

The binary pattern n°5 (for 198 value) is very particular because it is the only one which begins with the bit 1 and has forty-one following bits 0. This is a particularity of our coding choice in order to allow in the future real values higher than 198 with minor modifications of the software. For the time being we will never generate a pattern beginning with a bit 1 and having any other bits 1 elsewhere in its chain.

The knowledge of the coding interval and the amount of binary patterns potentially generated leads to the coding accuracy a_c:

$$a_c = \frac{2|I|}{2^{41}} = 9.004 \cdot 10^{-11} \tag{6}$$

This accuracy means that the smaller interval between two consecutive real values is a_c: it is a satisfying value. To decode binary patterns into real values we use an adapted form of a well known relation to obtain the relation (7) which is used under a computerized form in our software:

$$R = 198 \frac{\sum_{i=0}^{41} m_i \cdot 2^i}{2^{41}} \tag{7}$$

where R: real value corresponding to a binary pattern, m_i: binary value of the bit whose position index is i (m_i=0 or 1), i: position index of the bit m_i.

The calculations work on two real parameters, so we had to choice between coding each parameter as a distinct binary chain (this supposes working with two independent 42-bits-long binary chains), or coding and concatenating parameters in a single 84-bits-long chain. For our experiments we never obtained convergent results with two independent binary chains, so we decided in accordance with a general opinion in this domain, to concatenate the two parameters in the same coded chain. Then the first 42 bits are used to code the partial pressure pH_2O, and the last 42 bits to code the ratio pCO_2/pCO. We have not found specific publications on this topic in the literature.

Size of population

GAs work on the evolution of a set of binary chains named 'population'. The number of chains per population is the population size. There is no mathematical law to determine this, owing to the kind and the amount of parameters. It is only an experimental appraisement founded on former work and personal test, which allows the setting of correct values. A good compromise is needed between a small population having a good representation of the totality of the pertinent binary chains and bigger populations, perhaps more representative but requiring more computer time. Usually, sizes of populations found in the literature are

about 30 patterns. This is the value we have used. We also tested bigger populations (50 patterns) but we never noted an improvement in performance. In both cases we sometimes had no convergence of the algorithms to a correct result. This behaviour is not linked to the size of populations. We have not tested population sizes lower than 30 patterns.

Generation of initial population

As presented above the proposed interval for the values of the parameters offered by our coding procedures is larger than the current one required by the two parameters. In order to start the first calculations an initial population of 30 patterns must be generated. As we have not at the very outset a knowledge of the domain for the 2 parameters values, we developed a random procedure for the generation of the first population. The 30 binary 84-bits-long chains are built by concatening the equiprobable out of a random toss. The out values of a bit m_i are 0 or 1 with the same probability p, so we have:

$$m_i = x, \quad \text{where } p(x = 0; x = 1) = 0.5 \tag{8}$$

From a computing point of view, binary patterns can be considered as a one-dimensional indexed array. So the building procedure of an initial binary pattern can be represented diagrammatically from an algorithmic point of view by

```
for j=1 to size_population
        for i=1 to 84
                pattern[j]·bit[i] = x
        end i
end j
```

where: size_population: size of the population (for us 30), j: number of the binary pattern in the population, i: position index of the bit being generated, pattern[j]·bit[i]: computerized structure allowing generation and storage of the binary patterns. This is a structure as defined in the C language. Then this structure is the computerized form of the bit m_i, x: binary value for the bit m_i defined such as $p(x=0; x=1)=0.5$.

The structure really implemented checks the conformity of the generated chains with the specifications presented above. From a probabilist point of view the first populations are globally constituted with the same amount of bits 0 and 1. As the real values of the parameters are smaller than the offered interval, we had to adapt a masking system when the chains have been generated.

The binary masking system

The Fig. 6 shows the distributions of 4000 real values randomly generated without mask by using our above procedure:

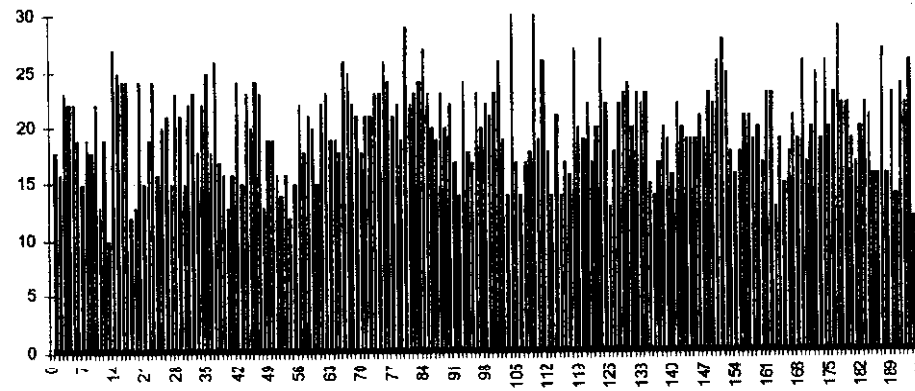

Fig. 6 Distribution for a set of 4000 real values randomly generated from 0 up to 198

The lowest value generated is 0.06939654 and the highest one is 197.986953. We note for this big set of values a relatively homogeneous distribution due to the great amount of values generated (4000) and that very small value close to 0 ($<10^{-2}$) will be very hard to generate. As we create populations with only 30 patterns, this phenomenon will be emphasized. In Fig. 7 we show the frequencies for one distribution of only 30 real values randomly generated.

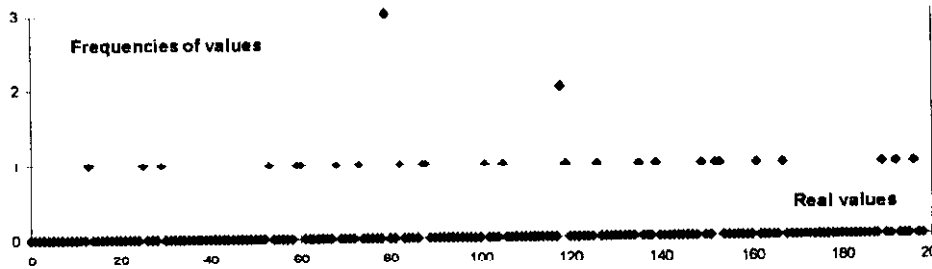

Fig. 7 Frequencies of real values generated for one distributions of 30 patterns

We note that several areas are less concerned than others. In this example there is no generated value close to 0, and there is only one value between 0 and 20. This phenomenon is normal owing to the width of the binary patterns generated and the small size of the population (30 patterns). In Fig. 8 we have superimposed the frequencies for 10 successive distributions of 30 patterns in order to show that on a global set of patterns (10×30=300) the values generated again becomes more homogeneous.

As the GAs explore the whole binary coding space offered, we added a procedure in order to voluntary limit for our case the area for number generation. For this we apply a mask directly on the binary patterns. Figure 9 schematically shows the masking procedure.

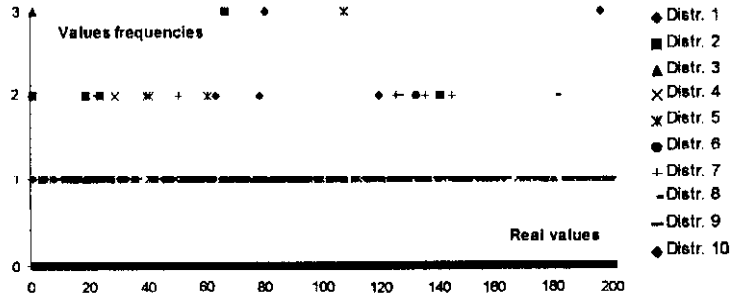

Fig. 8 Frequencies of real values generated for 10 distributions of 30 patterns

As soon as the binary patterns are generated, a procedure transforms two n-bits areas into the 0 bit (inverted areas in Fig. 9). The amount n can be set by the user in the source code. For the time being the masking width is static, but we think that a dynamic one will be more interesting and efficient.

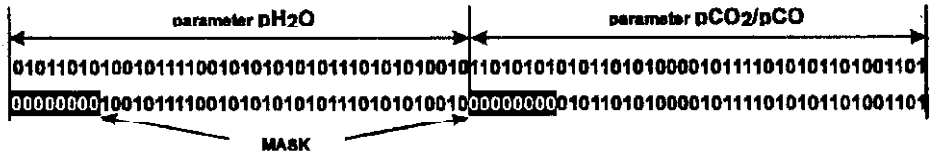

Fig. 9 Masking of binary patterns

The necessity of masking can be more sensitive if number close to 0 must be handled. In this case the mask must be wider.

The genetic operators

Operators are the most important part of genetic algorithms. They give an evolutionist behaviour to the calculations. They start to work on the population randomly generated such as those shown above. We used the two main operators: 'crossover' and 'mutation'. The main running of genetic algorithms is shown in Fig. 4 and concerns many genetic applications in which we can find the following basic steps:

1. Generation of a first random population
2. Performance testing for each pattern of the population (end of program if the performances are good enough)
3. Selection of couples of patterns owing to their good performances
4. Performing of genetic operators:
   - Crossover on selected patterns
   - Mutations on the entire population
5. Insertion of new patterns in the population and going to the step 2.

The crossover

This is the best known operator. Its name comes from an analogy with the biologic chromosomes duplication. Its fundamental steps are schematised in Fig. 10.

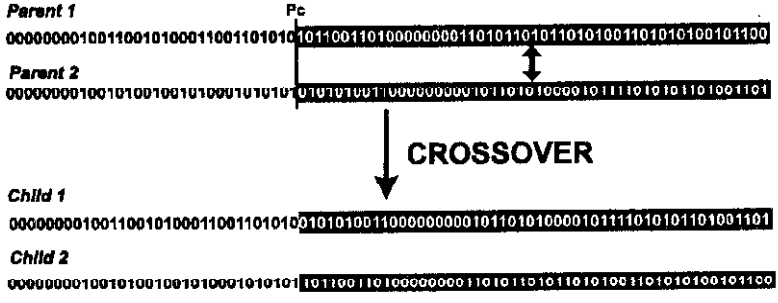

Fig. 10 Schematic performing on two parent patterns

The PC index represents the position from which the crossover is performed. The most important part for this genetic operator is the pertinent selection of the couples of patterns concerned by crossover. These couples of patterns are selected owing to their performances. To perform the evaluation of the pattern's respective performances, we consider the errors induced by the two parameters on the calculations on the total pressure P and the ratio nO/nH. We can postulate that the weaker the errors in these calculations, the greater the probability that the selection of the corresponding patterns will be stronger. So in order to materialize this probability we determine for each pattern j an accuracy factor named '$proba_j$' such as:

$$proba_j = \frac{1}{(errp + errn)_j probcum} \tag{9}$$

where $proba_j$: factor accuracy for a pattern j, *errp, errn*: respective errors induced by the pattern on the calculation of the total pressure P and the ratio nO/nH, *probcum*: cumulated probability of the patterns for the entire population. This quantity can be considered as a total fitness of the population, j: number of a pattern in the population.

The cumulated probability 'probcum' is defined as:

$$probcum = \sum_{j=1}^{\text{size pop}} \left(\frac{1}{errp + errn} \right)_j \tag{10}$$

To perform the selection of probabilist patterns the software generates a random series of size pop=30 numbers included in [0, 1], and then for each value of

the series a pattern of the population which has the value immediately above it will be selected. Consequently we generate a temporary population in which the better patterns have a greater appearance-frequency than others. The very 'bad' patterns will not appear in this temporary population. Next a second random series of size_pop probabilities is generated, then we select the corresponding pattern if its factor of accuracy '$proba_j$' is smaller that a given probability. Usually this threshold is 0.25; then we expect that (on average) 25% of patterns undergo crossover. As this selection must concern couples of patterns, the software will remove one pattern to obtain an odd set. The matching of patterns and the position PC for crossover are randomly performed. When the crossover is completed the generated 'child' patterns replace their respective 'parents' in the original population.

The mutation

As soon as the crossover is performed the mutation operator starts. It concerns the entire population. Its principle consists in randomly altering some bits of the population. Usually 1% of the bits is expected to mute. As for us a population is composed with 84·30=2520 bits, we will have a maximum of 25 bits which undergo mutation. Considering our specifications we decide not to allow mutation on the first bits of the patterns because this would involve the setting of all the others bits to 0 in order to generate the number 198.0000000000 (cf Table 1). The probability of generation of this exact quantity is very weak. The fundamental goal of this operator is to avoid definitive disappearance of binary patterns which could produce good 'children' is a future generation. Moreover the mutation has the ability to spontaneously create a very good pattern. This approach can be considered as the simulated annealing procedure which can be employed in other uses of the artificial intelligence in order to avoid local minima. This operator creates local disorders favourable to the appearance of solution patterns.

The masking procedure described above and applied to the initial randomly generated population is then also applied after the mutation operator in order to respect the global format of the binary patterns.

When the masking procedure is achieved, the new population obtained is ready for the next calculations of error. The entire cycle is named 'generation'.

Computing sights

Most of the currently existing softwares performing GAs are developed in an academic goal. It would be very time-consuming to adapt these kinds of products to our problem. So we decided to develop all of the computing procedures. The initial software for calculation of the temperature of flames was developed in BASIC some years ago. Then we had first to develop a C language release of this

product. We also had to develop a tool to manage and use the database of thermochemical properties initially created. All these developments were realized on PC (Pentium 133 MHz) computers with Borland C. The final C software was successfully tested and gave identical results with a better efficiency than the Basic release, as was natural. This release did not include genetic procedures, so it is this C software that we have modified in order to perform genetic algorithms.

The program handles coded parameters by means of structures plainly speaking in the strict sense of C language. It would be possible to use an object-oriented method in C++. In these C structures, which we named 'chromosome', there are different fields in order to memorize:

- The number of the binary pattern in the population
- The binary chain itself
- The numerical values corresponding to the 2 parameters
- The errors induced for each parameter
- The efficiency (accuracy) of the pattern
- The total efficiency for the entire population

So this structure defines a computing type. A patterns population is thus defined as a structure composed with n above structures and which represents a population (n is the size of the population, for us n=30). For each important step in the life of a binary-patterns population we handle a corresponding structure. So we have:

- An initial structure. This is the reference structure of the population used to perform the error calculations and run the genetic operators.
- A buffer structure. As its name indicates this structure is suitable to handle temporary populations during calculations.
- A storage structure used to perform crossover on couples of patterns.
- A temporary structure used to perform mutation and masking procedure.

The binary patterns can be stored on hard disk in order to trace the evolution of a population during the debugging phase.

To develop the genetic release of the initial program of temperature the new genetic procedures were included in the source code of the program of calculation of temperatures, so the prior classic ones have been removed. As the goal of the present work is to validate the procedure of resolution of the pH_2O and pCO_2/pCO parameters for a given temperature we do not perform the iterative 'dichotomic' process on proposed successive temperatures shown in Fig. 1. For the current study the software runs all the genetic operators for a single temperature given by the user in order to test their convergence on the combustion of heptane.

User interface

When the program is started the user specifies the features of the reaction studied by choosing in the database all the species concerned, entering their re-

spective stoechiometric coefficients, the reaction family (to be chosen among 6) and some additional numerical parameters such as for example atomic ratio nO/nH and total pressure P. When this text-form input is achieved the software passes into fully graphic mode, then starts the calculations.

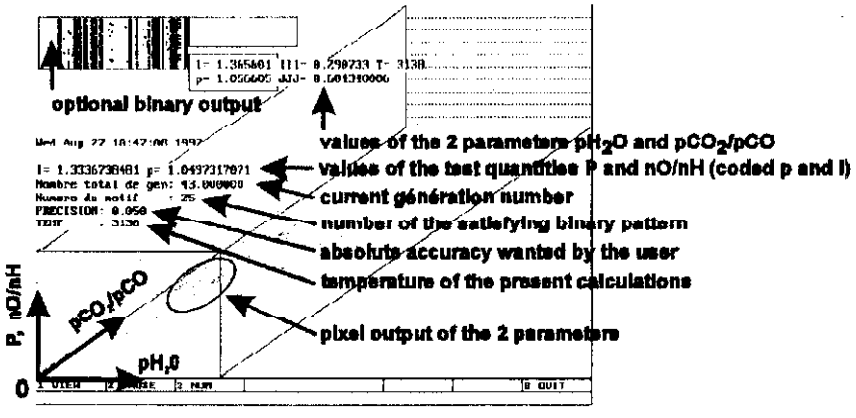

Fig. 11 Three dimensional output of the graphic interface

In order to monitor the evolution of calculations a user-graphic 3D-interfacing was fully developed. We show in Fig. 11 an output view of this interface when convergence of the GAs is obtained.

On the screen are displayed:

- The values of the two parameters being evaluated. They are noted III, JJJ and represent respectively pH_2O and pCO_2/pCO.
- The values of the test parameters which are the total pressure P and the atomic ratio nO/nH.
- Some parameters such as the number of the current generation, the number of the binary pattern which succeeds in the above tests, the precision required by the user, the temperature for which the calculations are conducted.
- The couples of proposed parameters pH_2O and pCO_2/pCO. They are graphically represented by pixels. The graphic procedures were developed in order to display a 3D output of the evolution of the calculations.

The user can stop the program at any time. Moreover in order to check the behaviour of the patterns the user can optionally display their binary structure in the upper left window of the display. As this procedure is very time-consuming it was not systematized.

Experimental results

To perform genetic algorithms there are numerous parameters to be taken in account, but well-known methods of choosing, determining or making an inven-

tory of them for a given problem do not yet exist. At the moment only experiments allow tests on the convergence of the genetic operators owing to the evolution of a population. In our very particular case of real positive and thresholded quantities determination we tested the GAs by studying some parameters:

The coding mode for parameters

As we said above, for the determination of several parameters it is possible to code them with distinct binary patterns or to concatenate them in a single binary chain. We conducted preliminary experiments with two separated 42-bits-long chains for each parameter. So the genetic procedures ran the operators independently on the two distinct populations. In that case we never obtained a convergence of the algorithms. We saw that it was impossible to have two accurate different populations simultaneously. This phenomenon can be explained because as we have no physical association of the binary forms of the two parameters, it is impossible to control the influence of the correction lead by the genetic evolution on each population. At a time t one population can given a good accuracy and the other population can have a less good one, so the genetic operators (such as the mutation operator) can randomly deteriorate the accuracy of the better population, and then the test will not succeed. As the GAs have a random behaviour it is not possible to lock together two accurate populations.

So, as described above, we coded the two parameters with a single 84-bits-long binary pattern, then we obtained convergence. In this case the patterns are physically associated, so the evolution processes are going to concern the totality. The optimization of their performances is linked.

The size of the population

This is an important parameter which defines the number of binary patterns in a population. No method exists to set this size according to a given problem. There is a compromise to be found between a good accuracy induced by a large population and the corresponding time consumed. In fact this size must determine a population large enough to be representative of the problem studied, and small enough to lead a reasonable data processing time.

According to the literature and to general opinion of searchers in this area, we set population size to 30 binary patterns. So the global binary size of a population is 84·30=2520 bits. We have conducted some tests with bigger sizes (50 patterns) but we have never noted better performances. We did not test smaller sizes. All the tests were conducted on a PC Pentium computer running at 133 MHz.

Behaviour of the genetic operators for different temperatures

The calculated flame temperature given by the first developed software is 3138 K. We have performed some tests in order to test only the capability of the

genetic algorithms to converge for a given temperature. For these tests the standard parameters were:

Size of the population: 30 patterns
Accuracy: 5%
Rate of mutation: 1%

Table 2 shows the number of generations performed according to a given temperature.

Table 2 Number of generations according to a given temperature

Temperature/K	Number of generation	Time-calculation/s
4500	168	6.05
3500	187	6.73
3138	43	1.55

As the GAs converge in this range of temperatures we decided to study the influence of some parameters on the behaviour of results for a given temperature.

Behaviour of the genetic operators for a given temperature

We studied the influence of the accuracy and the rate of mutation on the behaviour of the calculations for a given temperature. We selected the flame temperature 3138 K because our procedures must have a very good behaviour for this temperature. If no, it will not be useful to test GAs for other temperatures. There are two very interesting parameters:

- The accuracy: this quantity (which is set by the user) represents in fact the absolute error between the parameters calculated by the software and the user's ones. As these values are close to 1, the accuracy can be considered as an error percentage.
- The rate of mutation: this parameter is also set by the user. It represents the capability of the GAs to explore new parameters values on a probabilist way. We think that the influence of this parameter is very important on the behaviour of the GAs.

These parameters can be easily set in the source code before compilation. Two quantities characterize the efficiency of the procedures developed:

- The number of no-convergence: to determine this quantity we tested our algorithms in series with identical parameters. The number of tests in a series in 32 (2^5), then we count the no-convergence of the algorithms. For us there is no-convergence if the GAs need more than 10 000 generations to satisfy the tests (10000 generations represent about 6 min).
- The time-calculation: when we have convergence the time needed to satisfy the tests is a good indicator for the efficiency of the GAs.

Table 3 Number of no-convergence according to accuracy and mutation

N° experiment	Accuracy/%	Mutation/%	No-convergence
1	5	1	0
2	3	1	4
3	1.5	2	5
4	1.5	3	1
5	1.5	4	0
6	1	4	0
7	1.5	1	22
8	1	3	1
9	4	1	0
10	3	2	0
11	0.5	4	0
12	0.5	3	1

Table 3 summarizes at 3138 K the number of no-convergences of the AGs owing to the accuracy and the rate of mutation. The initial values were 5% for the accuracy (absolute error 0.05) and 1% for the rate of mutation. Our methodology was to explore couples of values (accuracy, mutation) in order to give a global sight of the efficiency. The experiments were numbered in their chronological order.

We saw from Table 3 that the best efficiency is for high values of accuracy or mutation. This is a normal phenomenon. The worst performance was for small values of accuracy (1.5%) and mutation (1%). As the goal is to obtain convergences with a good accuracy, we recommend to choose a correct accuracy (for example 1%) and try different rates of mutation from a small value (such as 1%) up to the smallest possible value which give 0 no-convergence. For our experiments 4% for the mutation gave 0 no-convergence with the wanted accuracy (1%).

When this procedure was correctly achieved we tried with the same mutation (4%), a smaller value for the accuracy (for example 0.5%). Doing that we also obtained 0 no-convergence. Then for the same value of the accuracy (0.5%) smaller values of mutation can be tested. We tried 3% for the mutation with 0.5% for the accuracy but we got one no convergence. So for our experiments the best couple of values for the accuracy and mutation was (0.5%, 4%). Note: In order to become finer it is possible to use no-entire values for the rate of mutation, but we have not tried that.

Other parameters such as the rate of crossover or the width of the binary masking could be studied, and will be the object of our future work.

With the best experimental conditions we have summarized in Table 4 the errors calculated for the calculation of the proposed parameters (pH_2O and pCO_2/pCO), and for the tested parameters (nO/nH and the total pressure P).

Evaluation of the performances

We note in Table 4 that the smallest minimum error (0.0023%) on a result is given for pCO_2/pCO with accuracy=1% and mutation=4%. The smallest average error (0.1951%) and the smallest maximum error (0.3593%) on a result are given for nO/nH with accuracy=0.5% and mutation=4%. These are very good values, and we note they are brought with the rate of mutation=4%. That is normal because this rate of mutation allows the choice of a very satisfying accuracy (0.5%).

Table 4 Errors on calculated parameters according to experimental conditions

Conditions	Error	pH_2O/%	pCO_2/pCO/%	nO/nH/%	P/%
Accuracy: 4%	average	0.7185	0.8872	0.3829	0.4641
Mutation: 1%	min	0.0367	0.0166	0.0405	0.0477
	max	1.9919	2.5642	0.7251	0.9935
Accuracy: 3%	average	1.6426	2.0586	1.4246	1.5536
Mutation: 2%	min	0.0154	0.0909	0.5934	0.0105
	max	4.4586	3.9023	2.1211	2.9124
Accuracy: 1.5%	average	1.0394	1.0235	0.4429	0.9130
Mutation: 4%	min	0.0265	0.0337	0.0044	0.0259
	max	2.1619	2.6047	1.0545	1.4893
Accuracy: 1%	average	0.8172	0.7393	0.4011	0.4780
Mutation: 4%	min	0.0149	0.0023	0.0552	0.0441
	max	1.8572	2.1026	0.7188	0.9543
Accuracy: 0.5%	average	0.5279	0.9193	0.1951	0.3051
Mutation: 4%	min	0.0484	0.2819	0.0140	0.0194
	max	1.2067	1.5744	0.3593	0.4856

The worst average error (2.0586%) and maximum error (3.9023%) are given for pCO_2/pCO with accuracy=3% and mutation=2%. The worst minimum (0.5934%) is given for nO/nH with accuracy=3% and mutation=2%. These are relative good performances.

For the next study about the time consuming according to the experimental conditions, we have chosen to work with smaller series of calculations, in order to reduce the global time-calculation. This will be necessary when our genetic procedures will be included in the real iterative procedure of determination of the flame-temperature.

We decide to work with new series of 8 tests (2^3), which represent 25% of the former series.

Time consuming according to the experimental conditions

We have generated 3 new series with mutation=4% and accuracy=1.5%, 1% and 0.5%. For each series we have measured the calculation time. As expected we had not no-convergences in these series. Tables 5, 6 and 7 summarize these features.

For these 8-test-series we note that the average of the calculation time for one generation is about 0.038 second, and the average of the global time for each test is between about 10 and 30 s. The longer calculation time (29.52 s) is given for the best accuracy (0.5%), but the faster time (10 s) is not given by the worst accuracy (1.5%). It is because in each series we can have very different numbers of generations. To improve the homogeneity of the values we can for example decide to select the six smaller numbers of generations in each 8-test-series. In this case Tables 8, 9 and 10 show the new series.

Table 5 Calculation time for accuracy=1.5% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	17	0.66	0.0388
2	13	0.49	0.0377
3	23	0.83	0.0361
4	14	0.55	0.0393
5	15	0.61	0.0407
6	2008	74.75	0.0372
7	30	1.1	0.0367
8	1164	43.33	0.0372
average	410.5	15.29	0.0380

Table 6 Calculation time for accuracy=1% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	35	1.32	0.0377
2	56	2.14	0.0382
3	1730	64.43	0.0372
4	106	3.96	0.0374
5	22	0.82	0.0373
6	24	0.88	0.0367
7	77	2.85	0.0370
8	97	3.63	0.0374
average	268.375	10.00	0.0374

In these new 6-test-series the average of the global time for each test is now between 0.71 and 2.39 sec. We can now note that the smallest average is associated with the worst accuracy and the largest one is associated with the best accuracy. That is a normal correlation.

Table 7 Calculation time for accuracy=0.5% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	12	0.5	0.0417
2	159	5.98	0.0376
3	165	6.21	0.0376
4	45	1.70	0.0378
5	82	3.07	0.0374
6	44	1.65	0.0375
7	5770	215.59	0.0374
8	38	1.43	0.0376
average	789.38	29.52	0.0381

Table 8 Calculation time for accuracy=1.5% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	17	0.66	0.0388
2	13	0.49	0.0377
3	23	0.83	0.0361
4	14	0.55	0.0393
5	15	0.61	0.0407
6	30	1.1	0.0367
average	18.67	0.71	0.0382

Table 9 Calculation time for accuracy=1% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	35	1.32	0.0377
2	56	2.14	0.0382
3	22	0.82	0.0373
4	24	0.88	0.0367
5	77	2.85	0.0370
6	97	3.63	0.0374
average	51.83	1.94	0.0374

Table 10 Calculation time for accuracy=0.5% and mutation=4%

	Number of generations	Time of calculation/s	Time/generation/s
1	12	0.5	0.0417
2	159	5.98	0.0376
3	45	1.70	0.0378
4	82	3.07	0.0374
5	44	1.65	0.0375
6	38	1.43	0.0376
average	63.33	2.39	0.0383

In order to use the values of the parameters calculated in the final iterative procedure of determination of the flame-temperature we recommend then to select in a series the test which give the smallest error.

Conclusions

In this work we successfully tested the ability of genetic operators for the determination of real parameters used in the data processing of flame-temperature of pyrotechnic reactions. Our goal was to validate the use of genetic operators at a given temperature for the combustion of heptan in air.

We obtained very good convergences of the genetic algorithms. The next steps of our study will consist in testing the iterative 'dichotomic' method for the proposed successive temperatures. As the scale of the parameter depends on the proposed temperature, we think it will be necessary to perform a dynamic masking of the binary patterns.

This original notion will be useful for the determination of parameters from DSC curves. For this experience we have developed a gaussian signal generator whose characteristics are compatible with our genetic procedures. Their implementation will be facilitated.

Our work validates the use of genetic operators in order to replace more classical determination methods. The random and probabilist behaviour of the genetic operators allows avoidance of possible local minima. Nevertheless an optimization of some features, as for example the size scale of the parameters generated is necessary to avoid some divergences of the algorithms more systematically.

References

1 N. Sbirrazzuoli and D. Brunel, Neural Comput. & Applic., 5 (1997) 20.
2 N. Sbirrazzuoli, D. Brunel and L. Elégant, J. Thermal Anal., 49 (1997) 1.
3 D. Brunel, J. Cassan and L. Elégant, J. Thermal Anal., 44 (1995) 969.

4 A. G. Gaydon and H. G. Wolfhard, Flames, 4th ed., Chapman and Hall, London 1979, p. 323.
5 D. E. Goldberg, Les algorithmes génétiques, Addison Wesley, Paris 1991.
6 L. Davis, Handbook of genetic algorithms, V. Nostrand-Reinhold, New York 1991.
7 J. C. Heudin, La vie artificielle, Hermés, Paris 1994.
8 A. Hunter, SUGAL: Sunderland genetic algorithm (Simulation software), University of Sunderland, Sunderland 1995.
9 H. M. Cartwright, Applications of artificial intelligence in chemistry: Genetic algorithms, Oxford University Press, Oxford 1993.