Using Genetic Algorithm to manipulate polynomial expressions

Andrzej J. Pindor
Information Commons
University of Toronto
130 St. George Street
Toronto, Ont. M5S 3H1
Canada
Email address: andrzej.pindor@utoronto.ca
Phone: (416) 978-5045

Abstract

It is shown how the Genetic Algorithm, together with the Computer Algebra System Maple can be used to manipulate and simplify large polynomial expressions. Various parameters of GA: the shape of the fitness function, the size of the initial population, mutation and mating rates and others are determined by numerical experiments.

1 INTRODUCTION

In analyzing large mathematical expression the simpler the better is usually the case. The simplest form of an expression makes it easier for us to see and appreciate it's structure. The rewriting process requires often many manipulations and transformations, using various mathematical relationships. Nowadays, Computer Algebra Systems are a great help, but of course they have their limitations.

For polynomials it is often desirable to present them in the form of a product of factors (if possible) or at least a sum of a small number of terms each of which is itself a product of factors. Computer Algebra Systems usually contain a function *factor* which can present a polynomial as a product of factors (if possible) over the domain specified by coefficients. However, if the polynomial is a sum of two or more products of factors, the user has to determine how to group the terms before using the command *factor*. For instance, if we have a polynomial

$$a^2 - 2bxa - bx + 4x^2 - 4xa$$

it takes a certain amount of effort to notice that it can be presented as

$$(2x - a)^2 - bx(1 + 2a) = \\ factor(a^2 - 4xa + 4x^2) - factor(bx + 2bxa)$$

The simplest way of tackling the problem would be to consider all possible groupings of terms of a polynomial, to apply the command *factor* to each and determine which gives the most desirable outcome. The problem with this approach is that the number of possible groupings grows very rapidly with the number of terms. For 9 terms the number of relevant groupings is almost 10,000 , for 10 terms - nearly 52,000 and for 12 terms - nearly 2 million. Although these number do not seem particularly large, one has to take into account that the procedure *factor* has to be executed several times for each possible grouping. The brute-force approach requires 1300 sec on a Pentium machine running at 133Mhz for an expression with 9 terms and over 10,000 secs for an expression with 10 terms. Clearly, the brute-force approach becomes infeasible for larger expressions and below I present the use of the Genetic Algorithm to solve the problem. The Computer Algebra System used here is Maple and the program is written in the Maple language.

2 SETTING UP THE PROBLEM IN GA TERMS

The problem we want to solve consist then of finding a grouping of terms of a polynomial (from all possible, relevant groupings) which, when *factor*-ed, will produce the simplest, in some sense, form of the polynomial. The question of course arises: what is "simplest"? Is, for instance, $x^2 - y^2$ simpler than $(x + y)(x - y)$ or the other way around? On the other hand we would all probably agree that $(a - 2b)^3$ is simpler then $a^3 - 6a^2b + 12ab^2 - 8b^3$. Clearly, what is "simplest" is both a matter of personal taste and of the context - what we need the expression for. In the particular case discussed here I have decided on a certain way of calculating a number which would be smaller for "simpler" forms of an expression. The details are perhaps irrelevant here, but an example might be in order. For

$$expr0 = (x - 2a)^4 - (2ay - xb)^3$$

this number is 17 and for an expanded form of this expression, i.e. for

$$expr1 = x^4 - 8a^3y^3 + 12a^2by^2x - 6ab^2yx^2 + b^3x^3 - 8ax^3 + 24a^2x^2 - 32a^3x + 16a^4$$

this number is 65. This number (which I shall hereafter refer to as the "expression size") is used to calculate the fitness of an organism (see below) and the way it is calculated can be adjusted to reflect the preferences of the user.

Individual groupings of the terms of the fully expanded form of the polynomial will be the GA "organisms" and they are coded in the following way: the terms of the polynomial are numbered (starting from one) and a specific grouping is represented as a set of sets. All numbers have to be present in the set, distributed over the subsets and none of the number can appear twice. Due to the nature of the problem the set has to contain at least two subsets. Here are a few examples of the groupings (organisms) for the case of nine and twelve term polynomials:

{{5, 7}, {6, 9}, {3, 4}, {1, 2, 8}}

{{9}, {5, 7, 6}, {1, 2, 3}, {4, 8}}

{{10, 11, 12}, {1, 2}, {3, 6, 9}, {4, 5, 7, 8}}

It may be worth noting here that in the above formulation the problem discussed turns out to belong to the family of grouping problems and that there has been a number of attempts to apply GA to this area. Difficulties which arise in such applications, as well as a suggestion how to avoid many of them, can be found in (Falkenauer 1994). The representation outlined above inherently takes care of the redundancy which plagued many previous applications of GA to the grouping problems (thanks to properties of Maple's *set* data structure). Also, the crossover operator proposed below addresses concerns raised in (Falkenauer 1994) in connection with the crossover operators used in other applications of GA to grouping problems. It may even do it better than the operator proposed there.

To reconstruct an expression corresponding to a given 'organism' above, the command *factor* is applied to each subgrouping of terms represented by a subset of integers in the 'organism' and the results are added together.

Mutations implemented here come in two varieties - splitting mutations which cause one of the subsets to split randomly in two and coalescing mutations which cause two randomly selected subsets to coalesce into one. Devising a mating scheme is somewhat more complex. It was set up here in the following way - two organisms can mate in either of two cases: in the first case one of the candidate organisms has to have a subset with the same terms as two subsets in the other; in the second case two subsets in one of the organisms contain the same terms as two subsets in the other, although perhaps distributed differently. Mating consists of an exchange of the subsets containing the same terms. For instance:

{{8}, {5, 7, 6, 9}, {1, 2, 3, 4}} +

{{5, 7}, {6, 9}, {3, 4}, {1, 2, 8}}

goes into

{{8},{5, 7}, {6, 9} , {1, 2, 3, 4}} +

{{5, 7, 6, 9}, {3, 4}, {1, 2, 8}}

or

{{5, 7}, {6, 9}, {3, 4}, {1, 2, 8, 10}, {11}} +

{{9}, {5, 7, 6,}, {1, 2, 3}, {4, 8}, {10, 11}}

goes into

{{9}, {5, 7, 6,}, {3, 4}, {1, 2, 8 10}, {11}} +

{{5, 7}, {6, 9}, {1, 2, 3}, {4, 8}, {10, 11}}

It is clear that not every two organisms are able to mate. For instance, the two following ones have no material which could be exchanged in the manner described above:

{{1, 4, 5, 6}, {2, 3, 7, 8, 9}},

{{2, 6}, {3, 4, 5, 8}, {1, 7, 9}}

Fine tuning of the parameters of the GA, like the relative ratios of the contributions of the three outlined processes (to creating a successive population of the organisms) and the form of the fitness function, was performed by applying the scheme to polynomials with a relatively small number of terms which had known concise forms, for instance the expression **expr1** above.

The problem was cast as a maximization problem by defining the fitness of an 'organism" (a grouping of terms resulting in a certain form of the expression) as the square of the difference between the expression size for the expanded form and for the form corresponding to the actual grouping of elements specified by the 'organism'. Organisms for which the difference above was negative were taken to have zero fitness.

In the process of fine tunning the form of the GA to our particular case it was also discovered that the following additional features had to be present:

- all organisms in the population should be kept different. Without this requirement the population quickly becomes quite homogeneous and there is not enough variety to ensure an exhaustive search for the absolute maximum. The importance of enforcing uniqueness has already been stressed by Whitley (Whitley 1989); see also (Davis 1991).

- the mutation rate has to be quite high, unlike in typical GA applications, where it is usually kept very low (Kinnear 1994);

- in light of the high mutation rate it is important that the "best fit" organism is always copied to the new population; otherwise, even if the "best" organism is found, it could be lost in the next generation. This is usually termed 'elitism' (Davis 1991). Note that in our case a single mutation may drastically change the fitness of an organism.

Another important consideration is the size of the population with which one works. If the population is too small, there may not be enough variation in it to let the program explore all parts of the solution space. If the population is too large one life-time cycle (replacing an old generation with a new one) takes a very long time and the program runs very slowly. In our case a population of size n^2 (where n is the number of operands in the expression) worked quite well.

3 STRUCTURE OF THE PROGRAM

The logical structure of the program in a simplified form is presented in the form of a flow diagram in Fig.1 .

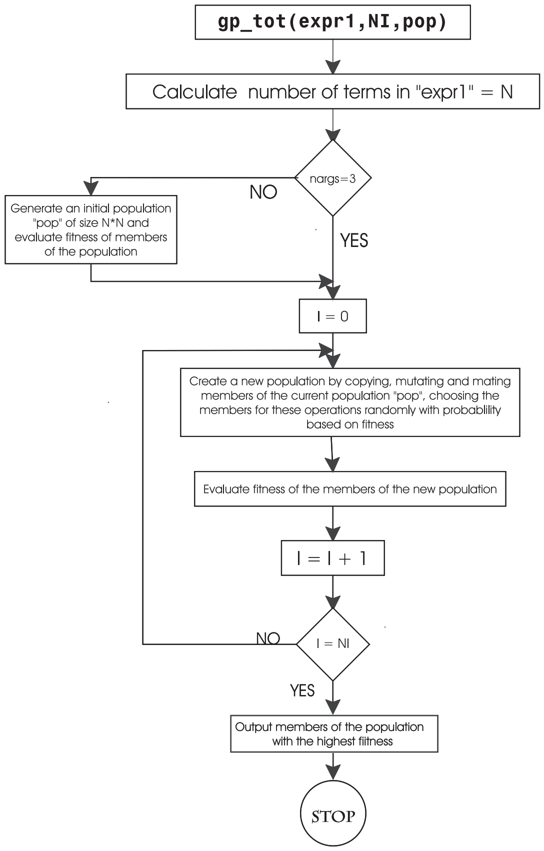

Figure 1: Flow diagram of the GA program

Input to the program is the expanded form of the expression, a number of iterations to be performed and the starting population. If the program is called with two arguments only (the internal variable "nargs" gives the number of parameters), then the initial population is generated internally. The program outputs the organisms with the highest fitness. The last population is also saved and can be used to perform more iterations.

4 THREE EXAMPLES

The results of three runs of the **gp_tot** program are presented below. Please note that in order to better monitor the convergence process, at the end of each iteration a number of parameters characterizing the population is printed out. These parameters, in the order in which they are printed, are:

- "best" organism (see below);

- iteration number;

- number of organisms in the population (this number may slightly exceed the number of operands of the expression, squared);

- average fitness of the population;

- lowest value of the expression size in the population;

- highest value of the expression size in the population;

- average number (over the current population) of subgroups in an organism;

- number of new organisms, as compared with the previous population;

The first two examples show that independent runs on the same problem may differ drastically and it is always advisable to repeat runs. We also find that even for a polynomial with 9 elements, the program based on the genetic algorithm may be more economical than the brute force approach of going through all possible groupings of terms. For larger polynomials it is of course the only practically feasible approach.

The Maple code for the procedure **gp_tot** can be found on the anonymous ftp site *ic-unix.ic.utoronto.ca* in the subdirectory *pub/scientific/maple/GA*, together with other files containing the code for auxiliary procedures.

The runs presented below come from a pentium machine, 133MHz, running Maple V R5 under linux. For the sake of brevity, portions of the output in the runs below are removed.

$$expr0 = (x-2a)^4 - (2ay - xb)^3$$

$$expr1 = x^4 - 8a^3y^3 + 12a^2by^2x - 6ab^2yx^2 + b^3x^3 - 8ax^3 + 24a^2x^2 - 32a^3x + 16a^4$$

This is a Maple command to execute the program

$$fff := gp\_tot(expr1, 5);$$

And here is the output

```
{{2,3,4,5}, {8}, {6,9}, {1,7}}
  0, 81, 95.198, 37, 65, 3.9

   Time of one cycle 28.560

{{2,3,4,5}, {6,9}, {1,7,8}}
  1, 83, 151.69, 36, 64, 4.0, 63

   Time of one cycle 35.350

{{2,3,4,5}, {1,6,7,8,9}}
  2, 84, 228.80, 17, 63, 3.9, 52
........................................

   Time of one cycle 32.810

{{2,3,4,5}, {1,6,7,8,9}}
  5, 83, 400.33, 17, 65, 3.5, 43
```

The three best forms of the expression found by the program are

$$(x-2a)^4 - (2ay - xb)^3$$

$$-(2ay - xb)^3 + x^4 - 8a(-a + x)(2a^2 - 2xa + x^2)$$

$$-(2ay - xb)^3 + 16a^4 + x(-4a + x)(8a^2 - 4xa + x^2)$$

Here we have been very lucky - the procedure found the optimum form of the expression in the second iteration. Since the starting population is created randomly, another run might require a greater number of iterations to find the same solution.

$$fff := gp\_tot(expr1, 5);$$

```
{{6}, {4,5,2}, {1,3}, {7,8,9}}
 0, 81, 96.914, 48, 67, 3.9

   Time of one cycle 25.420

{{1,3}, {7,8,9}, {4,6}, {5,2}}
 1, 84, 145.82, 42, 62, 3.9, 62

..........................................
   Time of one cycle 36.379

{{8,9}, {4,1,3}, {6,7}, {5,2}}
 5, 82, 279.88, 41, 61, 4.0, 44
```

Here it seems that the procedure has not converged after 5 iterations. We can add more iterations, noting that the most current population is stored in the global variable **new_pop**.

$$fff := gp\_tot(expr1, 5, new\_pop);$$

```
{{8,9}, {4,1,3}, {6,7}, {5,2}}
 0, 82, 279.88, 41, 61, 4.0

  Time of one cycle 36.370

{{4,6,7}, {3}, {8}, {5,9,1,2}}
 1, 82, 296.04, 34, 61, 3.8, 39

  Time of one cycle 35.880
```

```
{{4,6,7,8}, {3}, {5,9,1,2}}
 2, 82, 326.99, 30, 60, 3.8, 35

  Time of one cycle 37.830

{{4,6,7,8,3}, {5,9,1,2}}
 3, 83, 384.39, 17, 65, 3.9, 44
```

....

```
  Time of one cycle 29.580

{{4,6,7,8,3}, {5,9,1,2}}
 5, 82, 446.15, 17, 63, 3.8, 43
```

It may be worth noting here that solving the same problem using the brute-force approach (testing all relevant groupings of the terms) took almost 1300 secs on the same machine. The GA program took less then 600 secs to run through 10 iterations.

Now we try the procedure on a more ambitious case - an expression with 12 terms. As noted previously, for 12 terms there are nearly 2 million combinations which would have to be tried using the brute-force approach. The computing time required make this approach infeasible, since the estimated time for brute-force search is over 500,000 secs. As can be seen below, the genetic algorithm program can solve the problem in a reasonable time.

$$expr0 := (x-2a)^2 - (2ay - xb)^3 + (b^2y + a^2x^2)^2 + x^2y^2(a-b);$$

Expression size for this form is 36

When expanded, this expression has the form (I have deliberately shuffled the terms around):

$$expr1 = ay^2x^2 - by^2x^2 + a^4x^4 - 8a^3y^3 + 12a^2by^2x + 2b^2a^2yx^2 - 6ab^2yx^2 + b^3x^3 + b^3x^3 + b^4y^2 + x^2 - 4ax + 4a^2$$

and the expression size is now 89.

$$fff := gp\_tot(expr1, 20);$$

```
{{1,7}, {2,5}, {4,12}, {8,10,11}, {3,6,9}}
     0, 144, 164.75, 55, 89, 4.7

     Time of one cycle 119.790

{{1,7}, {2,5}, {4,12}, {8,10,11}, {3,6,9}}
     1, 146, 246.02, 55, 87, 4.9, 113

     Time of one cycle 159.680

{{1,7}, {2,5}, {4,12}, {8,10,11}, {3,6,9}}
     2, 147, 306.07, 55, 83, 5.0, 100

     Time of one cycle 206.690

{{1,5}, {2,7}, {4,12}, {8,10,11}, {3,6,9}}
     3, 145, 393.97, 53, 84, 5.1, 91
```

....

```
     Time of one cycle 245.049

{{4}, {10,11,12}, {1,5}, {3,6,9}, {2,7,8}}
    10, 145, 687.72, 52, 85, 5.1, 79
```

....

```
     Time of one cycle 214.830

{{1}, {10,11,12}, {2}, {3,6,9}, {4,5,7,8}}
    15, 146, 744.89, 42, 89, 5.0, 85

     Time of one cycle 206.881

{{10,11,12}, {1,2}, {3,6,9}, {4,5,7,8}}
    16, 147, 812.26, 36, 83, 5.2, 83
```

....

```
     Time of one cycle 211.109

{{10,11,12}, {1,2}, {3,6,9}, {4,5,7,8}}
    20, 145, 896.01, 36, 89, 4.9, 78
```

The three best forms of the expression found in this run are:

$$(x-2a)^2 - (2ay - xb)^3 + (b^2y + a^2x^2)^2 + x^2y^2(a-b)$$

$$x^2y^2\,(a-b) + x^2 - 4\,a\,(-a+x) + (b^2\,y + a^2\,x^2)^2 - (2\,a\,y - x\,b)^3$$

$$x^2\,y^2\,(a-b) + 4\,a^2 + x\,(x - 4\,a) + (b^2\,y + a^2\,x^2)^2 - (2\,a\,y - x\,b)^3$$

I have run the program 18 times for a 12 term expression and the average number of iterations required for convergence was 25. This corresponds to an execution time of about 8,000 secs. As mentioned above, the estimated time for the brute-force approach is about 500,000 secs.

5 GENERALIZATION OF THE PROGRAM

As described above, the program can only work in situations in which none of the terms from the factorized groups add up or cancel when these groups are expanded. For instance, the program is unable to find that

$$8x^6 - 12x^4by + 12x^4 + 6x^2b^2y^2 - 12x^2by + \\ 6x^2 - b^3y^3 + 3b^2y^2 - 3by - 1$$

can be written as

$$(2x^2 - by + 1)^3 - 2$$

because of the cancellation of the constant terms.

In case of addition or cancellation of the constant terms, the program can be easily modified to take this into account, to some extent. Any constant term (we only consider factorization over integers) can be represented as a sum of 1's and the expression can be padded with a certain equal number of +1's and -1's. How many would be up to the user to decide. Too few may not be enough but too many results increases considerably the number of terms in the expression and consequently the program consumes more time. However, this is the price to pay for solving the more difficult problem.

The program modified as described above had no problems finding the simplification above. I have run this case 7 times (padding the expression with integers 1,1,1,-1,-1,-1 which leads to 17 terms) with an average 15 iterations required for convergence. Finding that

$$8a^3x3 - a^3x^2 - 12a^2x^2 + a^2b - \\ ax^2 - 2a^2 + 6ax + b - 3$$

can be written as

$(2ax - 1)^3 - (ax^2 - b + 2)(a^2 + 1)$

turned out to be more difficult, in the sense that on a few occasions the program would get stuck in a certain region of the solution space. This was presumably due to the fact that the range of fitness values for this expression was only 529, whereas for the former expression it was 2500. When the way of calculating the expression size was modified so that the range of fitness values increased to 2209 in the above case (and to 10404 in the previous case), there was no problem with convergence. This new way of calculating the expression size worked also very well for all cases considered previously. One might extend the program in a similar way to the case of addition or cancellation of other (than constant) terms.

6 CONCLUSIONS

It has been shown here how the Genetic Algorithm can be used within a Computer Algebra system to perform additional simplifications of polynomials. GA was also used previously inside a different CAS, Mathematica, (Nachbar 1995) in a different context. It can be expected that GA can provide users of CAS systems with additional tools for dealing with certain mathematical problems. Data structures built into the Computer Algebra System make the CAS environment very convenient for applying GA to grouping problems (Falkenauer 1994).

Acknowledgments

I would like to thank an unknown referee for bringing my attention to the Falkenauer's paper (Falkenauer 1994)

References

E Falkenauer (1994). A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems, *Evolutionary Computation* **2**(2):123-144.

Kenneth E. Kinnear, Jr. (ed) (1994): *Advances in genetic programming*, MIT Press, Cambridge, Mass.

Lawrence Davis (ed) (1991): *Handbook of Genetic Algorithms*, Van Nostrand, New York.

Robert B. Nachbar (1995) . Genetic Programming, in *Mathematica Journal* Vo l5, Issue 3, p.36

D. Whitley, (1989) The GENITOR Algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Schaffer, J.D. (ed), *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufman.