Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Late-Breaking Papers of EuroGP-99

Edited by: W.B. Langdon, R. Poli, P. Nordin, T. Fogarty

Software Engineering (SEN)

SEN-R9913 May 31, 1999

Report SEN-R9913
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Late-Breaking Papers of EuroGP-99

Edited by

W.B. Langdon, Riccardo Poli,

Peter Nordin, Terry Fogarty

PREFACE

This booklet contains the late-breaking papers of the Second European Workshop on Genetic Programming (EuroGP'99) held in Göteborg Sweden 26–27 May 1999. EuroGP'99 was one of the EvoNet workshops on evolutionary computing, EvoWorkshops'99. The purpose of the late-breaking papers was to provide attendees with information about research that was initiated, enhanced, improved, or completed after the original paper submission deadline in December 1998.

To ensure coverage of the most up-to-date research, the deadline for submission was set only a month before the workshop. Late-breaking papers were examined for relevance and quality by the organisers of the EuroGP'99, but no formal review process took place.

The 3 late-breaking papers in this booklet (which was distributed at the workshop) were presented during a poster session held on Thursday 27 May 1999 during EuroGP'99.

This booklet is available as a technical report SEN-R9913 from the Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-1098 SJ Amsterdam.

1991 ACM Computing Classification System: I.2.8

Note: The editors' affiliations are W. B. Langdon (CWI, P.O. Box 94079, 1090 GB, Amsterdam, The Netherlands, e-mail `W.B.Langdon@cwi.nl`); Riccardo Poli (The University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK, e-mail `R.Poli@cs.bham.ac.uk`); Peter Nordin (Chalmers University of Technology, S-41296, Göteborg, Sweden, e-mail `nordin@fy.chalmers.se`); Terry Fogarty (Napier University, 219 Colinton Road, Edinburgh, EH14 1DJ, UK, e-mail `t.fogarty@dcs.napier.ac.uk`).

TABLE OF CONTENTS

Controlling Code Growth in Genetic Programming by Mutation

Anikó Ekárt
Computer and Automation Research Institute
Hungarian Academy of Sciences
1518 Budapest, POB. 63, Hungary
E-mail: ekart@sztaki.hu
Tel: +36 1 209 6194

ABSTRACT

In the paper a method that moderates code growth in genetic programming is presented. The addressed problem is symbolic regression. A special mutation operator is used for the simplification of programs. If every individual program in each generation is simplified, then the performance of the genetic programming system is slightly worsened. But if simplification is applied as a mutation operator, more compact solutions of the same or better accuracy can be obtained.

1. INTRODUCTION

An important problem with genetic programming systems is that in the course of evolution the size of individual programs is continuously growing. The programs contain more and more non-functional code. When evaluating the genetic programs over the fitness cases, much of the time is spent on these irrelevant code fragments. Thus, they reduce speed and, in the meantime, make the programs unintelligible for humans.

[Koza, 1992] proposes the use of a maximum permitted size for the evolved genetic programs as parameter of genetic programming systems. Therefore, genetic programs are allowed to grow until they reach a predefined size. In the same work a so-called editing operation is proposed for (1) making the output of genetic programming more readable and (2) producing simplified output or improving the overall performance of genetic programming. The editing operation consists in the recursive application of a set of editing rules. However, it was shown that for the boolean 6-multiplexer problem applying the editing operation does not influence the performance of genetic programming in a notable way.

[Hooper and Flann, 1996] apply expression simplification in a simple symbolic regression problem. They conclude that the accuracy of genetic programming could be improved by simplification. Additionally, simplification could (1) prevent code growth and (2) introduce new useful constants.

[Langdon, 1999] introduces two special crossover operators, the so-called size fair and homologous crossovers. These operators create an offspring by replacing a subtree of one parent with a carefully selected similar-size subtree of the other parent. By using these operators, code growth is considerably reduced without affecting the performance of genetic programming.

There are several studies that suggest taking into account the program size when computing the fitness value.

[Iba *et al.*, 1994] define a fitness function based on a *Minimum Description Length* (MDL) principle. The structure of the tree representing the genetic program is reflected in its fitness value:

$$mdl = Error\_Coding\_Length + Tree\_Coding\_Length.$$

[Zhang and Mühlenbein, 1995] demonstrate the connection between accuracy and complexity in genetic programming by means of statistical methods. They use a fitness function based on the MDL principle:

$$Fitness_i(g) = E_i(g) + \alpha(g)C_i(g),$$

where $E_i(g)$ and $C_i(g)$ stand for the error and the complexity of individual i in generation g. The Occam factor $\alpha(g)$ is computed as a function of the least error in the previous generation $E_{best}(g-1)$ and the estimated best program size for the current generation $\hat{C}_{best}(g)$. Thus, their fitness function is adaptively changing from generation to generation.

[Soule *et al.*, 1996] compare two methods for reducing code growth in a robot guidance problem: (1) the straightforward editing out of irrelevant and redundant parts of code and (2) the use of a fitness function that penalizes longer programs. They conclude that applying the penalty outperforms *any kind* of editing out, so providing new evidence for [Iba *et al.*, 1994; Zhang and Mühlenbein, 1995].

Notwithstanding, other studies show that these seemingly irrelevant or redundant parts of code are useful because they shield the highly-fit buiding blocks of programs from the destructive effects of crossover.

[Angeline, 1994] calls these apparently useless fragments of code *introns*, in analogy with the introns contained in DNA. He points out that the formation of introns should not be hindered, since they provide a better chance for the transfer of complete subtrees during crossover.

[Nordin *et al.*, 1995] demonstrate through experiments that introns allow a population to keep the highly-fit building blocks and in the meantime make possible the protection of individuals against destructive crossover. They introduce the so-called *Explicitly Defined Introns* that are inserted in the code and serve as a control mechanism for the crossover probability of the neighboring nodes.

In the present paper we describe a method that takes both advices into account:

- Code growth in genetic programming should be limited in order to obtain a comprehensible solution in a reasonable amount of time.
- Introns should be preserved, since they shield the highly-fit building blocks from the harmful effects of crossover.

The paper is organized as follows: In Section 2 the biological evidence that inspired this work is presented. In Section 3 the method is described and in Section 4 the results of 800 runs of the system on two symbolic regression problems are shown. Then a real-world application using these results is discussed and conclusions are drawn.

2. BIOLOGICAL BACKGROUND

The DNA of living organisms contains:

- genes - the active DNA sequences;
- junk DNA - with no apparent effect on the organism; and
- control segments - that regulate timing and conditions of protein production.

Generally, the genes consist of:

- exons - base sequences that encode proteins or polypeptides; and
- introns - base sequences that do not participate in the production of proteins.

The DNA of procaryotes contains one continuous sequence of exons and no introns. However, the genes of most eukaryotic organisms are discontinuous, and the sequences of interrupting introns are much longer than those of exons. There is evidence suggesting that: *"introns were present in ancestral genes and were lost in the evolution of organisms that have become optimized for very rapid growth, such as eubacteria and yeast"* [Stryer, 1988].

Apparently introns play no role in the production of proteins, just like junk DNA. But they represent regions in which DNA can break and recombine without affecting the encoded proteins [Stryer, 1988].

[Gilbert, 1985] points out that *"introns have been used to assemble those genes that are the late product of evolution"*. At the same time, he brings evidence for the *loss* of introns during evolution.

Thus, introns play an important role in evolution, when shielding the exons from destruction through crossover. But they can disappear in the course of evolution.

On the other hand, the main source of variability is mutation. From the many existing types of mutation, we consider the following [Watson *et al.*, 1987; Banzhaf *et al.*, 1998]:

- change of one base pair to another;
- frameshift mutation - addition or deletion of one or more base pairs; and
- large DNA sequence rearrangement.

Usually, in genetic programming systems the analog of the first type is implemented. In addition, we also consider here the other two mutation types.

3. THE PROPOSED METHOD

The goal of this work is to reduce the size of genetic programs evolved in symbolic regression problems. However, the method could be applied to any genetic programming system after defining the corresponding rules that simplify the structure of programs.

We designed a special mutation operator that modifies only the structure of a genetic program; the interpretation and the fitness value remain the same. This modification is intended to eliminate the occasional introns and simplify the structure of the genetic program, without altering its accuracy (in a similar way to the editing operation of [Koza, 1992] and the expression simplifier of [Hooper and Flann, 1996]). Since the problem is symbolic regression, the mutation operator performs the algebraic simplification of the expression of a genetic program. The simplifier is implemented in Prolog and consists of approximately 250 clauses. Some of the simplification rules are shown in Table 1.

Table 1: Some Simplification Rules.

Original expression	Simplified expression	Binding
$0 + x$	x	
$K_1 * x + K_2 * x$	$K * x$	$K = K_1 + K_2$
$K_1 * K_2 * x$	$K * x$	$K = K_1 * K_2$
$(-1) * x$	$-x$	

Let us see two short examples:

$$f(x, y) = x * 2 - x * (3 - 1) + 3 * x/y \stackrel{\text{simpl}}{=} 3 * x/y$$
$$g(x) = x * (x - 2) + 3 * x \stackrel{\text{simpl}}{=} x + x * x$$

In the first example the non-functional part was removed and in the second one algebraic simplification was performed.

One can see the analogy of this simplification to the biological mutation:

- removal of non-functional code - frameshift mutation; and
- algebraic simplification - large DNA sequence rearrangement.

The removal of non-functional code is more restricted than frameshift mutation, since it is applied only to non-functional code and there is no addition, just deletion of this code.

The algebraic simplification is in fact more than a rearrangement. For a more precise analogy with biology we could have made just a simple transformation, such that:

$$g(x) = x * (x - 2) + 3 * x \stackrel{\text{transf}}{=} (3 * x - 2 * x) + x * x.$$

This transformation is closer to large DNA sequence rearrangement and could be the subject of later experiment.

Since the simplification of an algebraic expression involves the removal of non-functional code, we decided for a single mutation operator, that performs all possible simplifications on the selected expression (like the editing operation proposed by [Koza, 1992]).

We applied this mutation operator in addition to the usual recombination operators. We thought that applying the simplification in every generation might be too drastic and time-consuming and, therefore, we made the frequency of its application a parameter of the genetic programming system (also suggested by [Koza, 1992]).

4. EXPERIMENTAL RESULTS

Experiments were conducted on two symbolic regression problems. The goal was to evolve the programs that approximate the functions (1) $F_1(x) = x + x^2 + x^3 + x^4$ and (2) $F_2(x) = 1.5 + 24.3x^2 - 15x^3 + 3.2x^4$, respectively, in 100 data points, that were randomly selected in the $[0, 1]$ interval. The parameter setting is shown in Table 2.

Table 2: The Genetic Programming Parameter Setting.

Objective	Evolve a function that fits the data points of the fitness cases
Terminal set	x, real numbers $\in [-100, 100]$
Function set	$+$, $*$, $/$
Fitness cases	$N = 100$ randomly selected data points (x_i, y_i), (1) $y_i = x_i + x_i^2 + x_i^3 + x_i^4$ (2) $y_i = 1.5 + 24.3x_i^2 - 15x_i^3 + 3.2x_i^4$
Raw fitness and also standardized fitness	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(gp(x_i) - y_i)^2}$
Population size	100
Crossover probability	90%
Mutation probability	10%
Selection method	Tournament selection, size 10
Termination criterion	none
Maximum number of generations	50
Maximum depth of tree after crossover	20
Initialization method	Grow
Frequency of simplification	Every 1., 2. or 5. generation
Simplification probability	0-100%

In the fitness measure only the error is included, there is no term related to the

program size. We think that *a shorter program with more errors should not be preferred to a longer program containing less errors.* The mechanism for limiting code growth should be distinct from the selection mechanism. We introduced the simplification as a mutation operator in order to reduce code size without influencing the fitness-based selection mechanism.

We added two parameters: the frequency of simplification (F) and the simplification probability (P). Simplification is applied every F-th generation, on every individual program with probability P.

Table 3: Parameter Setting for Simplification.

Frequency F[gen.]	Probability P[%]
-	0
1	10, 20, 100
2	20, 40, 60
5	20, 25, 50, 75, 80, 100

In each parameter setting shown in Table 3 we performed 50 runs and recorded their average. The plots for the regression of F_2 are presented, since it is a more difficult problem. Nonetheless, the plots for F_1 have the same character.

In order to establish a good ratio between the probability and the frequency of simplification, we compared the results of the runs with the same overall simplification ratio P/F. ($P/F = 10\%$ is achieved when (1) 10% probability of simplification in each generation, (2) 20% probability of simplification in every second generation or (3) 50% probability of simplification in every fifth generation is applied). Particularly, we analysed the results for $P/F = 10\%$ and $P/F = 20\%$. The results for $P/F = 20\%$ are shown in Figures 1 and 2.

Considering program size, best results were obtained when simplification was applied in every generation: code growth practically stopped at the 30th generation. In the case of 100% simplification in every fifth generation, the form of the graphics (Figure 2 right) clearly reflects the alternating behavior of program size: after every fifth generation the average program size is reduced by simplification, then programs are allowed to grow for the next five generations. In fact, if we look at the average program size at every fifth generation (after simplification), we can see that in this case code growth is also moderated after the 30th generation. In the meantime, the fitness of the best program (Figure 1 left) was slightly lower when simplification was applied less frequently (and with higher probability, keeping P/F constant).

We made another comparison among the results of runs with different frequencies of simplification, keeping the probability constant at 20%. Figures 3 and 4 show the results for the cases: (1) no simplification (original genetic programming), (2) simplification in every generation, (3) every second or (4) every fifth generation. The results

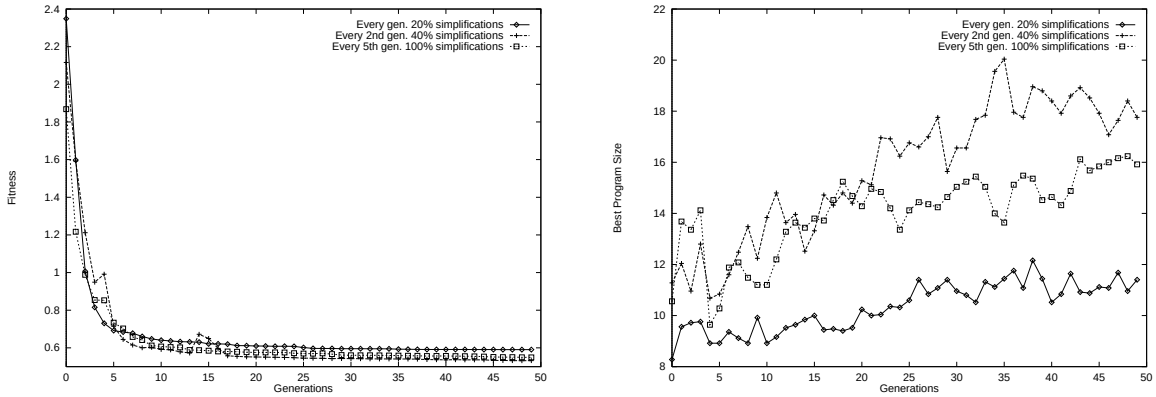


Figure 1: The Best Program Fitness and Size over Generations.

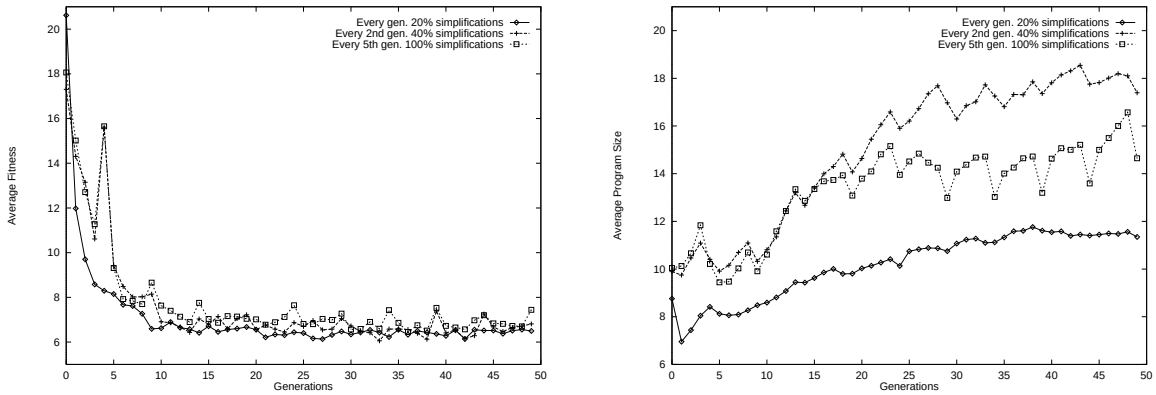


Figure 2: The Average Program Fitness and Size over Generations.

for simplifying every individual in every generation ($F = 1$, $P = 100\%$) are very close to those obtained when simplifying in every generation with probability $P = 20\%$ (since the corresponding graphics are practically undistinguishable, we do not show the graphics corresponding to total simplification). As we expected, in the case of genetic programming without simplification, the program size is continuously growing and in the case of simplification in every generation, the program size stabilizes at a low value after 25 generations. The fitness of the best program (Figure 3 left) is slightly worse when simplification is applied more often. The average fitness (Figure 4 left) of the cases when using simplification is better than that of the case with no simplification.

We also compared the results of runs, when the frequency of simplification was constant (every fifth generation) and the probability of simplification varied between 0-100%. While the size of programs was growing fast when no simplification was applied, it was quite stable when the probability of simplification was high. In the case of simplifying each individual program, the accuracy of the best program was slightly worse than for the other cases. Thus, we found again that applying simplification more often conducts to much shorter, but slightly worse solutions.

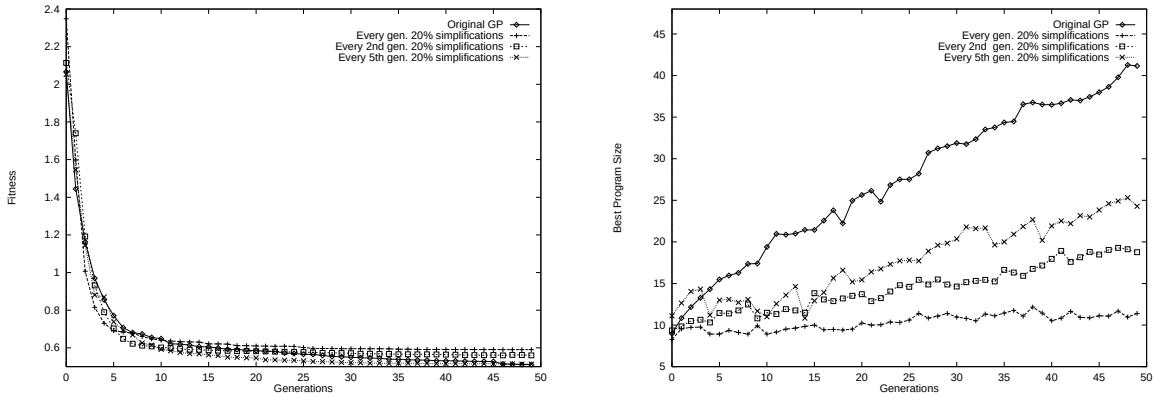


Figure 3: The Best Program Fitness and Size over Generations.

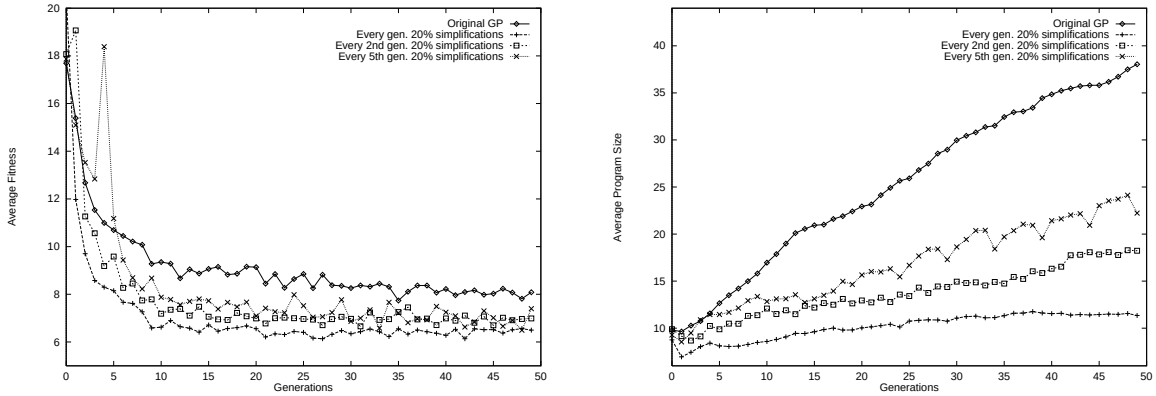


Figure 4: The Average Program Fitness and Size over Generations.

When choosing the probability of simplification, one has to make a trade-off between accuracy and program size:

- more accurate programs that grow moderately (less simplifications); or
- less accurate programs that do not grow (more simplifications).

5. APPLICATION

Our interest for limiting code growth in symbolic regression problems stems from the need for reducing the CPU time of our machine learning system [Ekárt, 1998]. The system is based on constructive induction, having the two components:

- learning engine - the C4.5 decision tree learning program [Quinlan, 1993]; and
- new feature generator - genetic programming.

We use this system in solving a mechanical engineering design problem, namely four bar mechanism synthesis [Sandor and Erdman, 1984]. The learning task is to discover the structural description of classes of such mechanisms. When generating the structural description, genetic programming creates the new features as algebraic

functions of the six structural parameters of four bar mechanisms (simple symbolic regression with a terminal set consisting of six variables).

Genetic programming is called at the creation of each node in the decision tree (about 50 times in each run), on the data that has to be classified at that node . The training data set (i.e., the fitness cases) contains more than 7000 items. Thus, evaluating irrelevant code on these data takes much CPU time.

6. CONCLUSIONS

In the paper a method for limiting code growth in genetic programming was presented. The mechanism for controlling the size of programs was distinct from the fitness-based selection mechanism. The control of code growth was realized by a special mutation operator, inspired by two forms of mutation in biology. The method was applied to symbolic regression, where the special mutation operator consisted in algebraic simplification. The evolution of programs had two alternating phases:

- classical genetic programming - allowing code growth and intron formation (several generations); and
- simplification of programs - eliminating introns and reducing program size by means of a mutation operator (one generation).

The experimental results show that code growth can be moderated or even stopped without the deterioration of performance by choosing the right frequency and probability for the application of simplification.

Acknowledgements

This work has been supported by grant No. T25471 of the National Research Foundation of Hungary and by the Eötvös Loránd University. The author is grateful to A. Márkus and J. Váncza for many helpful discussions.

References

[Angeline, 1994] Peter J. Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 75–97. MIT Press, 1994.

[Banzhaf *et al.*, 1998] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann, 1998.

[Ekárt, 1998] Anikó Ekárt. Generating class descriptions of four bar linkages. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 42–47, 1998.

[Gilbert, 1985] Walter Gilbert. Genes-in-pieces revisited. *Science*, 228:823–824, 1985.

[Hooper and Flann, 1996] Dale C. Hooper and Nicholas S. Flann. Improving the accuracy and robustness of genetic programming through expression simplification. In

John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, page 428, 1996.

[Iba *et al.*, 1994] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 265–284. MIT Press, 1994.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Langdon, 1999] William B. Langdon. Size fair and homologous tree genetic programming crossovers. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[Nordin *et al.*, 1995] Peter Nordin, Frank D. Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, 1995.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Sandor and Erdman, 1984] George N. Sandor and Arthur G. Erdman. *Advanced mechanism design: Analysis and synthesis*, volume 2. Prentice Hall, 1984.

[Soule *et al.*, 1996] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, 1996.

[Stryer, 1988] Lubert Stryer. *Biochemistry*. Freeman, 1988.

[Watson *et al.*, 1987] J. Watson, N. H. Hopkins, J. W. Roberts, J. Argetsinger-Steitz, and A. M. Weiner. *Molecular Biology of the Gene*. Benjamin-Cummings, 1987.

[Zhang and Mühlenbein, 1995] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

Parametric Coding vs Genetic Programming: A Case Study

Alain RACINE, Sana BEN HAMIDA, Marc SCHOENAUER
Centre de Mathématiques Appliquées
Ecole Polytechnique
91128 PALAISEAU, FRANCE

ABSTRACT

The goal is to design the 2-dimensional profile of an optical lens in order to control focal-plane irradiance of some laser beam. The numerical simulation of the irradiance of the beam through the lens, including some technological constraints on the correlation radius of the phase of the lens, involves two FFT computations, whose computational cost heavily depends on the chosen discretization.

A straightforward representation of a solution is that of a matrix of thicknesses, based on a $N \times N$ (with $N = 2^p$) discretization of the lens. However, even though some technical simplifications allow us to reduce the size of that search space, its complexity increases quadratically with N, making physically realistic cases (e.g. $N \geq 256$) almost untractable (more than 2000 variables). An alternative representation is brought by GP parse trees, searching in some functional space: the genotype does not depend any more on the chosen discretization.

The implementation of both parametric representation (using ES algorithms) and functional approach (using "standard" GP) for the lens design problem are described. Both achieve good results compared to the state-of-the-art methods for small to medium values of the discretization parameter N (up to 256). Moreover, preliminary comparative results are presented between the two representations, and some counter-intuitive results are discussed.

1. Introduction

Many optimization problems actually look for a function: unknown profiles can be seen as spatial functions, unknown commands can be described as time-dependent functions, ... Two approaches are then possible: standard numerical methods often discretise the definition domain of the unknown function, and the search space is then transformed into a space of fixed-length vectors of parameters (one value per discrete point). The original optimization problem is thus amenable to parametric optimization. But the accuracy of the solution is then highly dependent on the dicretization, and the finer the discretization, the larger the parametric search space. On the other hand, Genetic Programming (GP) offers an alternate approach where the search space is some functional space, independent of any discretization.

This paper presents a case study of such situation. The goal is to design a lens in order to control focal-plane irradiance of some laser beam. Irradiance control allows to concentrate laser beam energy on a particular region of the focal-plane of a focusing lens. Physicists use this technology to experiment a specific kind of nuclear fusion : *Inertial Confinement Fusion*. A tiny pellet of fuel is heated to a very high temperature by powerful beams of energy. Thus the laser inertial confinement process produces powerful bursts of fusion energy.

The best-to-date method to optimize the profile of the phase plate is simulated annealing [YLL96]. Nevertheless, this approach requires a significant computational time (120h of CPU time on a Cray YMP-2 computer). Others heuristical methods [SNDP96, fLEa, fLEb] can be used to provide an approximate phase profile. But these lenses are not efficient, i.e. they induce a drastical loss of energy outside the target: the spared computational time is balanced by a loss of precision.

We propose in this paper to use the *Evolutionary Computation* paradigm to handle the lens design problem. The choice of a representation is well known to be a crucial step in any Evolutionary Algorithm (EAs) – see e.g. the early debate on binary vs real encoding for real-valued parameters. On the other hand, EAs are flexible enough to be able to handle non-standard search spaces, such as spaces of variable length lists, graphs, etc. Along those lines, many representations have been designed to handle functions, among which Genetic Programming has proven very efficient on a number of problems (see e.g. [SSJ+96, Koz94] among others).

In the case of the lens profile design, the computation of the fitness of a given profile requires some numerical simulation of optical propagation of a laser beam, and the best-to-date methods for that involve Fast Fourier Transforms (FFTs), based on some $2^N \times 2^N$ discretization of the profile (see section 2). Hence a "natural" representation for a profile is the 2-dimensional matrix that will be used to compute those FFTs. But, even though working in the time domain allows us here to decrease the number of unknown parameters (see again section 2 for the details), the size of this parametric search space increases quadratically with the chosen discretization size N. So even without considering the computational cost of a single fitness evaluation, it is commonly acknowledged that the number of generations before convergence of any EA increases at least linearly with the size of the search space (see e.g. [TG93, Cer96]).

This is where functional approaches, among which Genetic Programming (GP), can take over: indeed, if the lens profile at a given point is represented as a function of that point, the size of the search space is independent of any discretization. Hence the number of generations to reach a given accuracy will hopefully be independent of any discretization, too – even though the computational cost of a single fitness evaluation will still depend on the discretization (i.e. the number of points where GP trees need to be evaluated still increases quadratically with the discretization parameter N).

Both approaches are presented in this paper: the parametric representation is handled by a $(\mu + \lambda)$ Evolution Strategy, and the non-parametric functional approach uses "standard" Genetic Programming to represent solutions by parse trees.

This paper first describes the application background and presents some feasibility constraints (section 2). The parametric representation, together with the associated ES algorithm, is introduced in section 3.1, while the functional approach based on Genetic Programming is detailed in section 3.2. The results of both algorithms are presented and compared in section 4. Both approaches give very good results compared to the state-of-the-art methods. But whereas GP quickly finds some good solutions (almost independently of the discretization, as expected), it experienced some difficulty in the fine-tuning of those solutions, and the ES method ultimately gives slightly better results. This suggest to build a hybrid algorithm that would take advantage of both methods: some further research directions are sketched in that perspective in section 5.

2. Focal-Plane Irradiance Problem

Consider a laser beam, specified by its scalar electrostatic field. The problem consists in designing a continuous distributed phase plate (the unknown lens shape) to produce a specified irradiance profile at the focal-plane of a given focusing lens.

Without any phase plate, the focal-plane profile corresponds to a single (very energetic!) peak at the focal point. In order to achieve the *inertial confinement fusion*, it is mandatory to be able to illuminate a given small circular target around the focal point of the focusing lens. Hence a phase plate is added before the focusing lens and its shape directly acts on the irradiance profile.

More precisely, the optical system (See figure 1) is composed by :

- a large pupil to control the complete illumination of the phase plate.
- a phase plate (to be determined) to modify the distribution of the phase along the section of the laser beam.
- a focusing lens
- a *virtual screen* at the focal-plane to measure the focal profile.

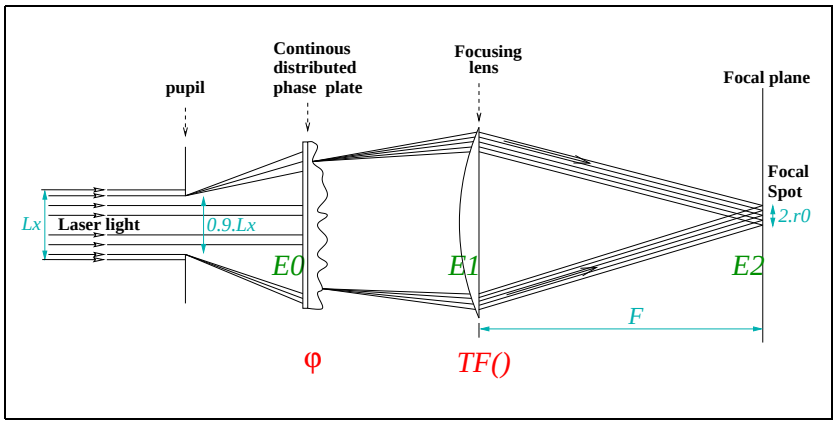


Figure 1: The optical system. All physical values have been normalized.

2.1 Analytic formulation

Figure 1 shows the experimental layout. First, the *pupil* produces a scalar electrostatic field:

$$E_0(x,y) = e^{-\pi\left(\frac{\sqrt{(x^2+y^2)}}{R}\right)^8}$$

The new field $E_1(x,y)$ generated by the phase plate $\varphi(x,y)$ is:

$$E_1(x,y) = E_0(x,y).e^{i\varphi(x,y)}$$

Thus, the field at the focal-plane is obtained by a simple Fourier Transform: $E_2(x,y) = \mathrm{FT}(E_1(x,y))$

Finally, the *relative* lighting intensity at the focal-plane is given by the intensity of the filed E_2, the Fourier transform of E_1:

$$I(x,y) = (E_2(x,y))^2 = (\mathrm{FT}(E_1(x,y)))^2$$

2.2 The optimization problem

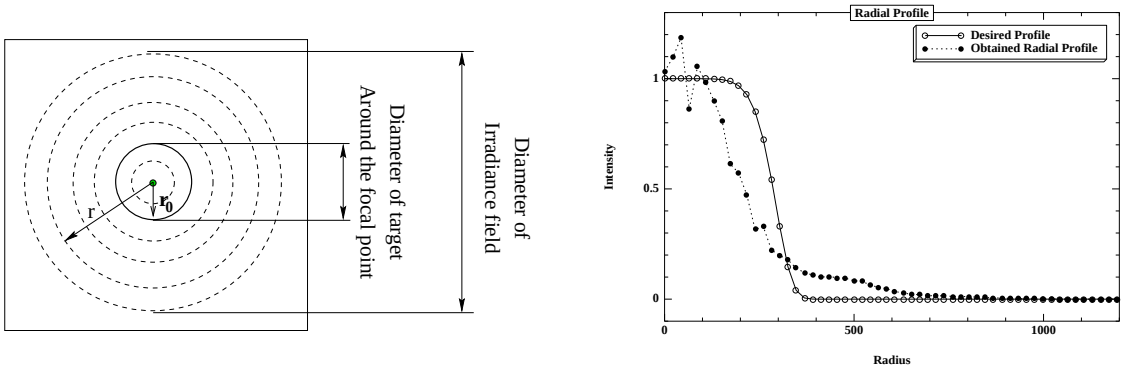
The goal is to design a phase plate such that a small circular target of radius r_0 on the focal plane is uniformly illuminated by the laser beam (see Figure 2-a): the perfect solution shows a radial step-like profile on the focal plane – which will be smoothed to a super-Gaussian profile of order 8 (i.e. $t(r) = \exp(-(r/r_0)^8)$ in the following.

The radial profile $\overline{I}(r)$ from $I(x,y) \equiv I(r,\theta)$ is given by:

$$\overline{I}(r) = \frac{1}{2\pi r} \int_0^{2\pi} I(r,\theta)\, d\theta$$

The goal is to design a phase-plate whose illumination profile on the focal plane will be as close as possible from the super-Gaussian profile (see Figure 2-b). The cost function (to be minimized) $\mathcal{E}rr$ becomes (after renormalization of $\overline{I}(r) \to \widetilde{I}(r)$) :

$$\mathcal{E}rr = \int_0^{\infty} \left[t(r) - \widetilde{I}(r) \right]^2 dr \tag{2.1}$$



(a) The target in the focal plane.

(b) Desired and sample radial profiles.

Figure 2: The optimization problem: fitting the desired radial profile.

2.3 Manufacturing Constraints

Some technological constraints have to be taken into account: manufacturers are not able to make very rough phase-plates. To ensure an acceptable smoothness of all designs, a common practice is to filter out the high frequencies of the phase-plate in the frequency domain (after applying Fourier transform), before transforming the filtered signal back into the space domain.

This leads to consider a new unknown field $Z(x,y)$, and to compute the phase plate φ by filtering out the high frequencies in Z as shown in Table 1 below.

Table 1: Smoothing the phase plate: the filter simply zeroes out high frequencies

$Z(x,y)$	$\longrightarrow$	$\tilde{Z}$	$\longrightarrow$	$\tilde{Z} \times Filter$	$\longrightarrow$	$\varphi(x,y)$
	FFT		$Filtering$		FFT^{-1}	

2.4 Working in Fourier space

Table 1 suggested a simplification of the search space: instead of working on Z, it is possible to consider directly the filtered Fourier transform $\tilde{Z} \times Filter$ as the unknown field. The number of non-zero terms in $\tilde{Z} \times Filter$ is much smaller than the number of terms in Z as all high frequencies correspond to a zero in the filter. Whereas it is clear that this will be an advantage for the parametric representation (see section 3.1 below), it nevertheless decreases the number of points where the unknown field $\tilde{Z} \times Filter$ has to be computed in the functional approach (section 3.2).

The number of non-zero values in $\tilde{Z} \times Filter$ is given in Table 2 for the different values of the discretization N used in the different FFT involved in the fitness.

Table 2: Non-zero terms in $\tilde{Z} \times Filter$ for different values of the discretization parameter N

$N = 32$	$\longrightarrow$	$nval = 32$
$N = 64$	$\longrightarrow$	$nval = 148$
$N = 128$	$\longrightarrow$	$nval = 560$
$N = 256$	$\longrightarrow$	$nval = 2284$

2.5 Fitness function

From the above consideration, the fitness function used thereafter can be graphed as in Figure 3, the comparison between both profile being given by equation (2.1).

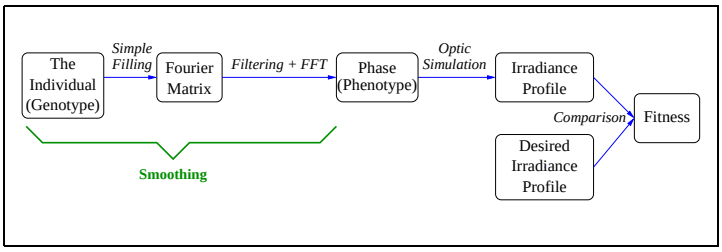


Figure 3: Fitness computation

The remaining open issue is the representation of the non-zero field $\tilde{Z} \times Filter$ - which will be thoroughly addressed in next section.

3. The evolutionary algorithms

3.1 Parametric representation

The most "natural" representation for the field $\tilde{Z} \times Filter$ of Table 1 is the direct encoding of all non-zero terms into a real-valued matrix - the size of the matrix is a function of the discretization parameter N as given in Table 2 (though it is not N^2, it is quadratically increasing with N).

We are then back into the familiar framework of parametric optimization, and have chosen a self-adaptive Evolution Strategy algorithm to address that problem (see [Sch81, B$\ddot{9}$5, BS93, BHS97] for more details on standard ES with self-adaptive mutation parameters).

Briefly, a standard deviation is attached to each design variable, and the mutation operator first mutate the standard deviation following a log-normal mutation before modifying the

design variable with a Gaussian noise using the new value of its standard deviation.

The recombination used is the standard ES "global discrete recombination" for both the standard deviations and the design variables. Note that a 2-dimensional crossover operator was also tried (the unknown parameters are indeed a 2-D matrix [KS95]) which did not give better results. All parameters are given in Table 3.

Table 3: ES parameters

Population size: μ	50
Number of offspring: λ	100
Selection Mode	$(\mu + \lambda)$
Mutation parameters	
Initial std. dev.: σ_0	0.3
Global update: τ	0.7
Local update: τ_L	0.3

3.2 Functional representation: GP

As discussed in the introduction, Evolutionary Algorithms offer alternate representations for functions – and the idea here is to represent directly the unknown field $\tilde{Z} \times Filter$ as a parse tree, using "standard" Genetic Programming as the optimization algorithm [Koz92, Koz94]. Table 4 is the standard GP tableau, giving all parameters.

Table 4: Tableau for the GP algorithm

Set of nodes	$\{+, -, *, sin\}$
Set of terminals	$\{x, y, \mathbb{R}\}$
Raw and stand. fitness	**Given by Equ. (2.1)**
Wrapper	**See Table 1**
`Population size`	**100**
`Selection Mode`	**Tournament (Size=10)**
`Replacement`	**Generational with elistism**
`Crossover probability`	**0.6**
`Mutation probability`	**0.4**
`Types of mutations`	Promotion of a branch Random Replacement of a branch Node and terminal permutation Constant terminals Gaussian mutation
`Maximal depth of an individual`	**14**
`Minimal depth at initialization`	**3**
`Interval for initialization of constant terminals`	[**-100,100**]
`Std. dev. for Gaussian mutation of constant terminals`	**10**

4. Results

Table 5 gives the average number of generations (over 11 runs) to reach a particular error level, while Figure 4 is a graphical view of the same results.

Table 5: Average number of generations to reach a specific error level. "??" means that error level was never reached, and a 0 error level indicates that the value is obtained at the first generation.

Discretization	32		64		128		256	
	ES	GP	**ES**	GP	**ES**	GP	**ES**	GP
Error=1.0	**51**	0	**220**	0	**416**	9	**799**	45
Error=0.5	**66**	0	**237**	3	**420**	48	**1021**	553
Error=0.05	**1216**	10	**608**	69	**1510**	??	**4741**	??
Error=0.025	**??**	698	**3896**	2175	**5277**	??	**??**	??

The first important result is that the results are very good indeed: an error of 0.025 is reached in 5277 generations for the 128×128 case – corresponding to a total computing time of about 30 hours of computing time for a Pentium II 350 Mhz, to be compared to the 120 hours of Cray YMP-2 for the 64×64 discretization. The best solution is given in Figure 5, showing a very good fitting of the target irradiance profile.

But when it comes to compare the parametric and the functional approaches, some surprising facts arise: as can be seen on Figure 4, GP is indeed more efficient than ES to reach quite large values of the error, but seems to have much difficulty to fine-tune the solutions and reach smaller errors. And this tendency is clearer as the discretization parameter N increases, which is the opposite of what was expected.

Figure 6 gives another point of view on this phenomenon: GP performs better than ES at the beginning of the evolution (until generation 630 for $N = 128$ on the figure.

It seems that in the early generations, ES has to optimize each coefficient one by one and takes a long time to reach reasonable performance while GP makes large jumps in the search space since a small modification of a tree generates a drastically different behavior. These results tallies with Angeline's work concerning the link between crossover and macro-mutation [Ang97]. In fact, both GP-mutations and GP-crossover are at this stage exploration operators.

In a second part, after ES has met GP performance, ES still improves the best fitness - thanks to self-adaptation. So even if the convergence of ES remains very slow, the fitness continues to decrease – and this is true up to 15000 generations, though the decrease becomes almost zero as the self-adapted parameters go to zero themselves.

These results reveal the lack of precision of our GP approach. GP roughs out very efficiently the exploration of the fitness landscape, but has a lot of difficulties to refine the solution. A possible improvement would be to also use self-adaptation to tune the constant terminals in GP rather than performing Gaussian mutation with fixed standard deviation – future work will investigate this possibility.

Another point that enforces this explanation is that the best results for GP are provided with a rather large tournament size (10 out of a population of 100). Indeed, this forces to concentrate the GP exploration around the best solution to improve the refinement process. However this approach rapidly reduces the diversity in the population, and GP is then searching in a quite narrow region of the search space.

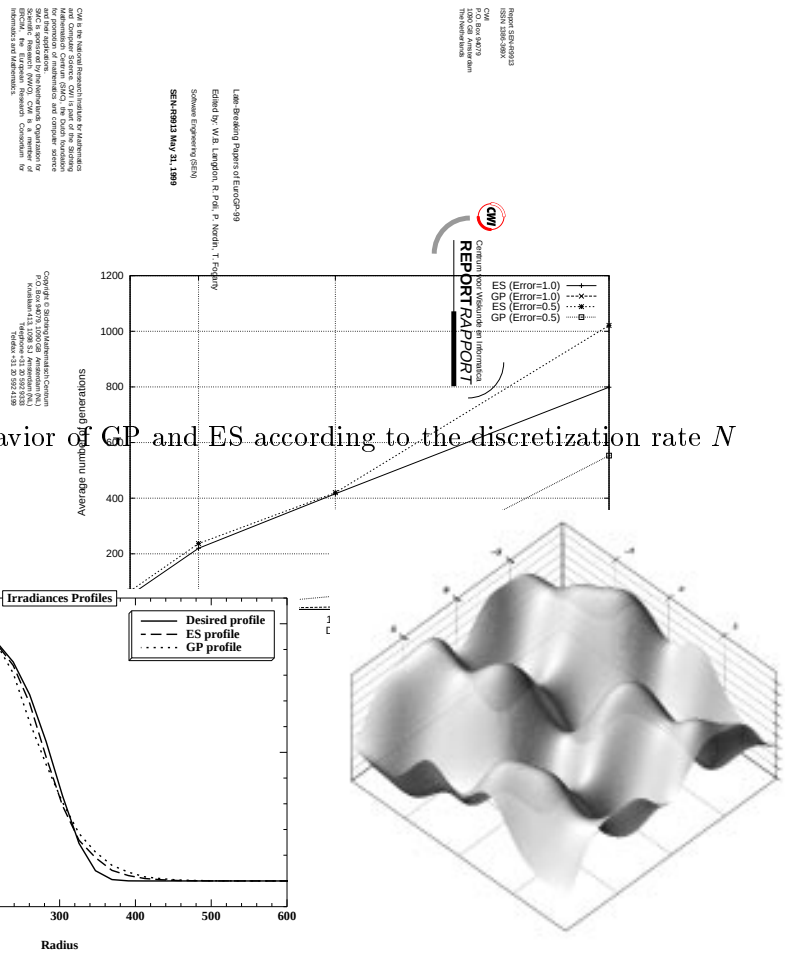


Figure 4: Behavior of GP and ES according to the discretization rate N

Figure 5: The best radial profiles (ES & GP) and the best phase plate (ES) for $N = 128$

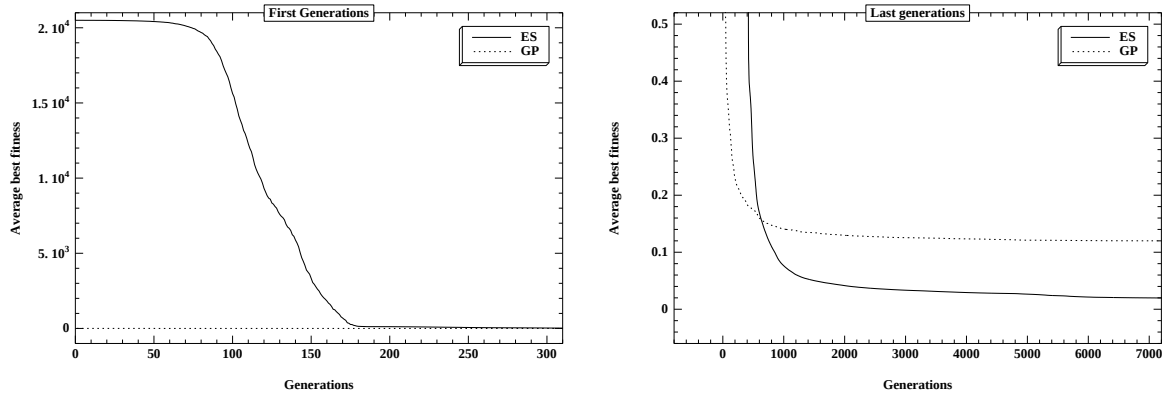


Figure 6: Average curves of the best individual with ES and GP (discretization rate=128)

5. A hybrid algorithm

From the evolution curves of GP and ES for $N = 128$ (Figure 6), it is clear that ES loses a lot of time at the beginning.

Hence in order to take advantage of both evolutionary approaches, we designed an hybrid algorithm. We start GP algorithm, and when it starts to stagnate, we transform the GP trees in the population into matrices of real values that are the initial population of an ES parametric run. This population at the resumption point naturally provides a genotypical diversity to the further optimization with ES.

Figure 7 shows first results (via semi-log scale) based on this two-steps strategy and underlines the increase of precision relatively to both algorithms alone a simple GP algorithm. Indeed, after the cross point, we notice that continued evolution with ES significally improve the final solution. Moreover, we can verify that the hybrid algorithm provides better results even than pure ES : solution at generation 5000 produces an error of 0.0172634 at the focal-plane (ES took an average number of 5277 generations to reach an error level equals to 0.025, see Table 5). However, no conclusion should ever be drawn from a single run ([Jr94]), and more experiments are needed to test the robustness of this hybrid approach.

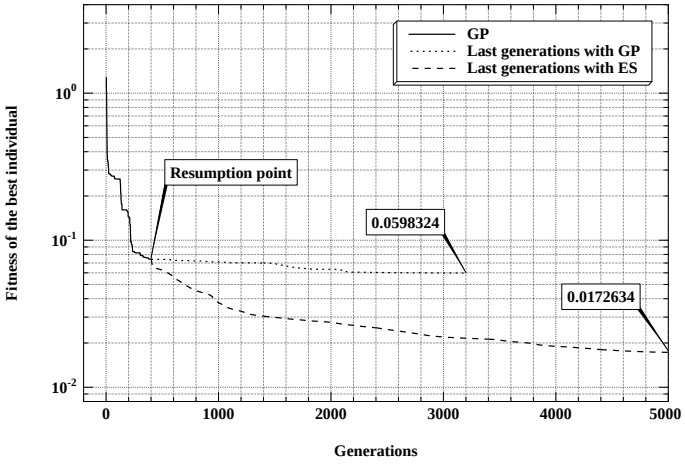


Figure 7: Comparison of evolution curves of GP and Hybrid algorithm

6. Conclusion

In this paper, a difficult real-world problem has been tackled by Evolutionary Computation using two different representations: the straightforward parametric representation by a matrix of real numbers, optimized using an ES+ algorithm, and the functional representation allowed by Genetic Programming.

Both algorithms gave good results compared to the state-of-the-art results, but the comparison of the parametric and the functional approaches lead to some surprising results. Whereas the GP functional approach was expected to be less sensitive than the ES parametric approach, the latter was able to obtain better-tuned results – whatever the discretization parameter. Our explanation lies in the inability of our GP implementation to fine-tune the

constant terminals – and hence locally optimize the good solutions it has found.

This was somewhat confirmed by the final experiment presented, which hybridized GP and ES, switching from the functional to the parametric representation to fine tune the solution – and reaching better solutions that both approaches alone. However, more experiments are needed to confirm the usefulness of that hybrid approach.

But the ultimate functional approach would be to use even further the analytic form of GP trees, postponing the numerical discretization as much as possible: We are presently working on a a pure symbolic system based on GP – using symbolic maths to compute all Fourier Transform whenever possible. This approach should provide a functional representation of the optimized phase plate instead of a discrete representation, hopefully avoiding most side effects generated by the discretization.

References

[Ang97] Peter J. Angeline. Subtree crossover: Building block engine or macromutation? In J. R. Koza and al., editors, *GP97: Proceedings of the* $2^n d$ *Annual Conf.* Morgan Kaufmann, 13-16 July 1997.

[B9̈5] T. Bäck. *Evolutionary Algorithms in theory and practice.* New-York:Oxford University Press, 1995.

[BHS97] Th. Bäck, U. Hammel, , and H.-P. Schwefel. Evolutionary computation: Comments on the history and current state. *Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[BS93] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[Cer96] R. Cerf. An asymptotic theory of genetic algorithms. In J.-M. Alliot and al., editors, *Artificial Evolution*, volume 1063 of *LNCS*. Springer Verlag, 1996.

[fLEa] Laboratory for Laser Energetics. Distributed phase plates for super gaussian focal-plane irradiance profiles. *LLE Review.*

[fLEb] Laboratory for Laser Energetics. High-efficiency distributed phase plate generation and characterization. *LLE Review.*

[Jr94] K. E. Kinnear Jr. A perspective on GP. In Jr K. E. Kinnear, editor, *Advances in Genetic Programming*, pages 3–19. MIT Press, Cambridge, MA, 1994.

[Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution.* MIT Press, Massachussetts, 1992.

[Koz94] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Massachussetts, 1994.

[KS95] C. Kane and M. Schoenauer. Genetic operators for two-dimensional shape optimization. In J.-M. Alliot and al., editors, *Artificial Evolution*, number 1063 in LNCS. Springer Verlag, Septembre 1995.

[Sch81] H.-P. Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, New-York, 1981. $1995 - 2^{nd}$ edition.

[SNDP96] M. D. Perry S. N. Dixit, M. D. Feit and H. T. Powell. Designing fully continuous phase screens for tailoring focal-plane irradiance profiles. *Optics Letters 21*, 1996.

[SSJ+96] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, and H. Maitournam. Evolutionary identification of macro-mechanical models. In P. J. Angeline and Jr K. E. Kinnear, editors, *Advances in GP II*, Cambridge, MA, 1996. MIT Press.

[TG93] D. Thierens and D.E. Goldberg. Mixing in ga. In S. Forrest, editor, *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms.* Morgan Kaufmann, 1993.

[YLL96] T. J. Kessler Y. Lin and G. N. Lawrence. Design of continuous surface-relief phase plates by surface-based simulated annealing to achieve control of focal-plane irradiance. *Optics Letters*, 1996.

Genetic Programming as an Analytical Tool for Metabolome Data

Richard J. Gilbert, Helen E. Johnson, Michæl K. Winson, Jem J. Rowland[†],
Royston Goodacre, Aileen R. Smith, Michæl A. Hall and Douglas B. Kell

Institute of Biological Sciences, University of Wales, Aberystwyth, Ceredigion SY23 3DD, UK
[†]Department of Computer Science, University of Wales, Aberystwyth, Ceredigion SY23 3DB, UK

rcg@aber.ac.uk, hej93@aber.ac.uk, mkw@aber.ac.uk, jjr@aber.ac.uk,
ars@aber.ac.uk, mah@aber.ac.uk dbk@aber.ac.uk

Corresponding Author: Richard Gilbert. Tel: +44 (0)1970 622353. Fax: +44 (0)1970 622354. http://gepasi.dbs.aber.ac.uk/rcg

Abstract

Genetic programming, in conjunction with advanced analytical instruments, is a novel tool for the investigation of complex biological systems at the whole-tissue level.

In this study, samples from tomato fruit grown hydroponically under both high- and low-salt conditions were analysed using Fourier-transform infrared spectroscopy (FTIR), with the aim of identifying spectral and biochemical features linked to salinity in the growth environment.

FTIR spectra are not amenable to direct visual analysis, so supervised machine learning was used to generate models capable of classifying the samples based on their spectral characteristics. The genetic programming (GP) method was chosen, since it has previously been shown to perform with the same accuracy as conventional data modelling methods, but in a readily-interpretable form.

Examination of the GP-derived models showed that there was a small number of spectral regions that were consistently being used. In particular, the spectral region containing absorbances potentially due to a cyanide/nitrile functional group was identified as discriminatory. The explanatory power of the GP models enabled a chemical interpretation of the biochemical differences to be proposed. The combination of FTIR and GP is therefore a powerful and novel analytical tool which, in this study, improves our understanding of the biochemistry of salt tolerance in tomato plants.

Introduction

The metabolome is a generic term for the total biochemical composition of a cell or tissue sample at any given time. Recent advances in DNA sequencing have lead to an explosion in the number of known gene sequences, but the majority of these new genes have never been characterised experimentally, and most have completely unknown functions within the cell. By investigating the changes in the metabolome of biological systems under different conditions, it is hoped that previously undescribed metabolic processes or pathways may be uncovered, leading to functional assignments for many of the newly-discovered genes within the genomic databases. This area of biology, termed functional genomics, will be a major focus of study over the next decade.

In order to study the metabolome of biological samples, new analytical techniques need to be developed. A typical metabolome study seeks to detect changes in the levels of a few specific biochemicals against a background of more than a thousand other cellular components. To address this, analytical instumentation is being developed which is capable of measuring biochemical signatures from whole-tissue or whole-organism samples. This typically results in datasets comprising measurements of many hundreds or thousands of variables. To complicate this task further, the identities of the particular biochemicals to be monitored are frequently unknown at the outset. The power of GP to select variables from high dimensional data and to form interpretable predictive models gives it a unique advantage in the analytical interpretation of metabolomic data.

Over the past two decades, the tomato as a crop has increased in popularity. Consequently, much research has been aimed at improving the economic viability of tomato production and post-harvest stability. Environmental stress, such as high salt concentration, is one of the main parameters limiting crop production. The tomato cultivar Edkawy is potentially salt-tolerant as it grows in the El-Bosaily area of North Egypt, where the soils are saline sands. Edkawy has already been studied in terms of salt tolerance and previous literature provides evidence that this tomato variety may have salt tolerant attributes[1,2]. In this study, Edkawy plants were cultivated using a hydroponic drip irrigation system, allowing precise control of the nutrient conditions within the root zone, including the salinity level. The aim of the study was to identify biochemical constituents (*biomarkers*) within the fruit tissue which are discriminatory for salt-grown tomato plants, and hence to contribute to the understanding of the fundamental biological mechanisms potentially underlying salt tolerance in tomato plants. This in turn may lead to rational improvements in the quality of tomato fruit grown in conditions of high salinity.

Fourier-transform infrared spectroscopy [3] is a physico-chemical analytical technique, which uses the vibrational characteristics of chemical bonds within molecules to obtain a 'fingerprint' spectrum with features defined by the functional chemical groups within the sample. This form of analytical technique is therefore able to give quantitative information about the total biochemical composition of a sample. A thin layer of the biological sample to be analysed is illuminated in the infrared to obtain an *interferogram* (produced by splitting an infrared beam of light, extending the path length of one half by reflecting it off a movable mirror, and recombining the beams optically). Chemical groups within the sample absorb specific frequencies of light within the interferogram due to 'resonance' with their vibrational motions, the precise frequencies absorbed being related to the energies specific to the vibrational modes of each chemical group. The information encoded in the reflected/absorbed light is then recovered by performing a Fourier-transform on the detected signal. The FTIR spectrum so obtained comprises 882 variables, each of which indicates the level of absorbance at a particular frequency of infrared light.

A readily accessible interpretation of such extremely high-dimensional spectra, also known as *hyperspectral data*, is often very difficult to obtain. Conventional analysis of data of this form falls into two types. The first type, *unsupervised* learning methods, includes principal components analysis (PCA), discriminant function analysis (DFA) and hierarchical cluster analysis (HCA), and seeks to form separable clusters in the data by performing mathematical transforms derived from the variables within the dataset without reference to known classes. The second type, *supervised* learning methods, includes partial least squares (PLS), multivariate rule induction (MRI), inductive logic programming (ILP) and artificial neural networks (ANNs), and seeks to refine a model based on the accuracy of its predictions for a set of examples with a known class structure. Although widely used, none of these methods provide models which are readily interpretable in a chemical sense.

Genetic programming [4-6] is an evolutionary technique which uses the concepts of Darwinian selection to generate and optimise a desired computational function or mathematical expression. GP is a

supervised learning method, and consequently requires a set of training examples to form predictive models that can then be applied to the classification of a set of previously unseen test samples. It has previously been shown that GP performs at least as well as conventional predictive modelling methods for analysing hyperspectral data [7].

Recently, hybrid GA-GP systems have been described [8,9] which are able to produce accurate predictive models whilst minimising their complexity by enforcing constraints on their functional form and expression length. However, a full GP system was chosen for use in this study because the precise mathematical form and complexity of predictive models able to classify tomato fruit tissue samples based on their FTIR spectra were unknown at the outset.

Methods

Plant Cultivation

The plants were grown in a hydroponic open-drip irrigation system, using perlite as an inert substrate. The use of a hydroponic system is ideal for studies into plant physiology as it allows complete control over the nutrients applied to the plants. The system was arranged to facilitate saline and control treatments. The capacity of this system was 120 plants with 60 replicates per treatment. All plants were irrigated with complete liquid fertiliser and supplementary sodium chloride (4000 ppm) was applied to the saline treated plants.

Fruit Tissue Preparation

Twenty fully ripe (at stage 10 on the OCDE tomato ripening chart) Edkawy fruits were harvested. Fruit were selected for uniformity to maximise homogeneity between samples. Ten fruit were taken from salt-grown plants, and ten from control plants. The seeds and skin were removed, the outer pericarp was crushed using a press, and kept on ice. The extract was homogenised using a Polytron blender at speed 5 for 1 minute. After homogenisation, 1ml aliquots of the sample were placed in Eppendorf tubes, and snap-frozen in liquid N_2. These were stored at -70°C until needed.

FTIR Spectroscopy

Ten replicate 5µl samples of each of the 20 fruit tissue samples were applied to wells drilled on a sandblasted aluminium plate, arranged to minimise the effects of artifactual trends in the data. Prior to analysis, the samples were oven-dried at 50 °C for 30 min. The plate was loaded onto the motorised stage of a reflectance thin-layer chromatography (TLC) accessory attached to a Bruker IFS28 FTIR spectrometer (Bruker Ltd.) equipped with a mercury-cadmium-telluride (MCT) detector cooled using liquid N_2.

The FTIR spectra were collected over a wavenumber range from 4000 cm^{-1} to 600 cm^{-1} under the control of an IBM-compatible personal computer using OPUS 2.1 software running under the IBM OS/2 Warp operating system. Spectra were acquired at a rate of 20 s^{-1}, and at a resolution of approximately 3.85 cm^{-1}. To improve the signal-to-noise ratio, 256 spectra were recorded and averaged for each sample. The complete dataset therefore comprised 200 averaged spectra, each containing 882 input variables. 100 spectra (50 from saline-grown and 50 from control fruit samples) were used by the GP as a training set to derive the models, and the remaining 100 spectra were used to test their predictive ability.

Genetic Programming

The GP implementation used in this study was capable of performing non-linear multivariate regressions with automatic variable selection. It was written in C, and was run on IBM-compatible PCs under Windows NT 4.0, and on DEC Alpha-based PCs under Linux 5.1.

The GP used the arithmetic operator functions *add*, *subtract*, *multiply*, and *protected divide* and a Boolean '*if greater than or equal to'* function. The *if* function returned a value of 1.0 if the first argument was greater than or equal to the second argument; 0.0 otherwise. To avoid possible numeric overflows, a protected *divide* function was used which returned a numerical value of 10^{15} for divisions with a denominator $\leq 10^{-15}$. Additional protection from floating-point errors was enforced by clipping the return value of each node into the range $\pm 10^{15}$.

Terminals comprised either floating-point constants (initialised randomly in the range –10.0 to 10.0) or input variables (corresponding to one of the 882 absorbance measurements which comprised each spectrum).

The GP generated initial individuals with random function trees of depth 2 to 6, and assessed their fitness using a scoring function that compared e_i (the model's estimate of the output for example i) with o_i (the experimentally-observed value) by calculating the root-mean-square error of prediction (RMSEP) for n training examples:

$$RMSEP = \sqrt{\frac{\sum_{i=1}^{n} (o_i - e_i)^2}{n}}$$

The fittest individuals were those which gave the lowest RMSEPs for the training set examples.

Since the dataset contains two classes (fruit from plants grown under either saline or control conditions), class membership was defined in the training examples by assigning a target output value of 1.0 to members of the saline-grown class, and 0.0 to members of the control class. A correct classification assigned when the output value of the GP-derived rule was within 0.01 of the target output for any given spectrum.

GP-generated rules, if allowed to evolve unchecked, tend to become longer and more arithmetically complex as the evolution proceeds, a phenomenon known as *bloat* [10]. This increase in complexity reduces the ready interpretability of the expressions generated. To combat this, a penalty of $0.01 \times N$, where N represents the number of nodes in the function tree, was added to the fitness calculation. This ensured that, for a given RMSEP, a shorter tree would be chosen over a longer one. In addition, a maximum tree depth of 10, and a maximum node count of 100 was enforced during the evolution.

The size constraints on the GP rules meant that even the longest rules could use only a small subset of the available input variables comprising the dataset. The GP was therefore compelled to perform an automatic variable selection, resulting in predictive models with significantly lower dimensionality (*i.e.* using far fewer variables) than the dataset as a whole. The automatic variable-selection ability of the GP approach is one of the main benefits of using this as a predictive modelling method [11]. Since the GP-derived models are readily-interpretable, analysis of the selected variables can lead to a rationalisation of the mechanism underlying the model.

The GP used five demes (sub-populations) each of 7500 individuals. Every 10 generations, the best 5% of the individuals in each of four satellite demes replaced the worst 5% in a central deme. The best 5% from this deme then replaced the worst 5% in the satellite demes. This divergent evolution and migration strategy has been shown to be more effective at solving high-dimensional problems than a conventional single-population GP [12].

During each generation, 1500 new individuals were created by single-point mutation, and 3000 by single-point crossover. Parental selection was proportional to fitness, and new individuals were retained in the deme if their fitness was higher than that of the current worst individual.

Partial Least Squares Modelling

Partial least squares (PLS) modelling is a widely-used supervised learning technique which reduces the dimensionality of multivariate data by using *a priori* knowledge of which spectra were derived from plants grown under saline or control conditions to produce mathematical models comprising linear combinations of variables. For this study, we used a PLS modelling system written in house by Dr. Alun Jones.

Variable Analysis

An analysis was performed to investigate the correlation between the GP-selected input variables and the known class structure of the data. *Product moment correlation* (PMC) is a method which uses linear transformations to quantify which variables (x) are most strongly related to the output data (y) being modelled.

The PMC (R) ranges from -1 to +1, indicating a perfect negative to a perfect positive correlation R takes the sign of C_{xy}. A value of 0 indicates that x is uncorrelated with y. R for n examples be calculated as follows:

$$R = \frac{C_{xy}}{\sqrt{C_{xx} \cdot C_{yy}}}$$

where

$$C_{xy} = \left(\sum_{i=1}^{n} x_i \cdot y_i \right) - n \cdot \left(\bar{x} \cdot \bar{y} \right)$$

$$C_{xx} = \left(\sum_{i=1}^{n} x_i^{\,2} \right) - n \cdot \left(\bar{x} \right)^2$$

$$C_{yy} = \left(\sum_{i=1}^{n} y_i^{\,2} \right) - n \cdot \left(\bar{y} \right)^2$$

Quantum Mechanics and Infrared Spectral Analysis

The semi-empirical quantum mechanics program PM1, part of the HyperChem 5.1 molecular modelling package (HyperCube, Inc.) was used to calculate infrared vibrational spectra and molecular vibrational modes for potential metabolites identified during the analysis of the data. The infrared spectral analysis

package IR Mentor Pro 2.0 (Bio-Rad Laboratories) was used to suggest candidate chemical groups responsible for the particular spectral features selected as discriminatory by the GP models.

Results and Discussion

After a sufficient number of reproductive generations, the GP was able to derive expressions capable of correctly classifying the examples in both the training and test sets to an average accuracy approaching 90%. The final GP model was produced after fewer than 400 generations. The PLS model was also to separate the classes to a similar accuracy (Figure 1). However, the PLS model, in common with the other widely-used statistical modelling methods, does not provide readily-available information about which variables have been selected.

The rules from 30 independent GP runs used 112 of the 882 input variables. The dataset as a whole contained variables with PMC values ranging from 0.000332 to 0.4401. The GP-selected variables had PMC values ranging from 0.000642 to 0.4296, indicating that the GP selected variables with both high and low correlations with the known class structure. The two most widely-used variables (each found in five models) had a PMC value of 0.3055 and 0.2876, both reasonably well-correlated with the class structure. Although the most-correlated variable in the data set was not used, the GP selected an adjacent variable on two occasions.

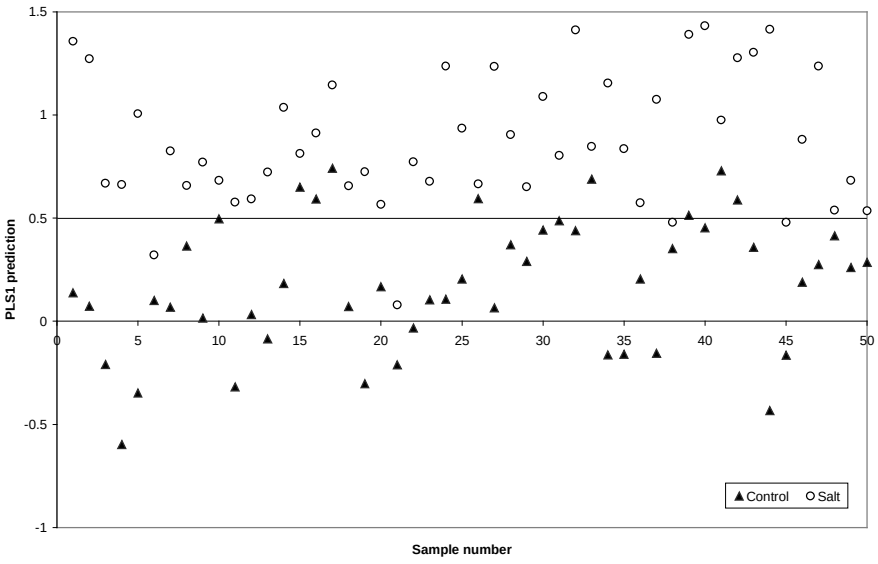


Figure 1

Partial Least Squares 1 (PLS1) was used to classify the samples. The PLS1 model was trained on 50 spectra. The chosen model used 11 components, which gave the minimum RMS error in the prediction values (0.3302) for a validation set of 50 previously unseen spectra. This model was then applied to 100 previously unseen spectra, comprising 50 control and 50 saline-grown samples. The plot below shows the model's predicted output values for this test set.

The GP-derived models used, on average, five variables in their predictive rules. No single variable could be used to classify the spectra with an accuracy approaching the 90% value of the GP-derived rules. Despite the reasonably high PMC values for most of the variables within the dataset PLS was unable to completely separate the two classes based on the spectral data (Figure 1). This indicates that the dataset does not contain enough information to allow a high degree of separation of the two classes to be made without using non-linear combinations of variables. The *if* operator was used in every GP rule, a clearly essential function for a classification problem which is not readily available to neural networks and the conventional statistical modelling methods.

The GP models were all different. The prediction accuracy ranged from 84.5% to 94% correct. Runs 9, 13, 18 and 27 produced remarkably similar models, with the same logical structure and using very similar variables:

Run 9: **IF (A_{2164}-A_{2245}) ≥ (A_{2060}-A_{2098}) THEN [Saline] ELSE [Control] (89% correct)**

Run 13: **IF (A_{2171}-A_{2245}) ≥ (A_{1963}-A_{2106}) THEN [Saline] ELSE [Control] (88% correct)**

Run 18: **IF (A_{2168}-A_{2257}) ≥ (A_{2029}-A_{2114}) THEN [Saline] ELSE [Control] (89.5% correct)**

Run 27: **IF (A_{2179}-A_{2230}) ≥ (A_{2025}-A_{2118}) THEN [Saline] ELSE [Control] (90.5% correct)**

In the above rules, A_n represents the measured absorbance at n wavenumbers. These rules may be indicative of the nature of the globally-optimal rule derivable from this dataset. The best performing rule, with a 94% predictive accuracy, used a similar logical construct and selected similar variables. However it included an additional term associated with a spectral feature at 2480 wavenumbers which enabled a better class prediction for some of the saline-grown samples which were incorrectly classified by the simpler rules:

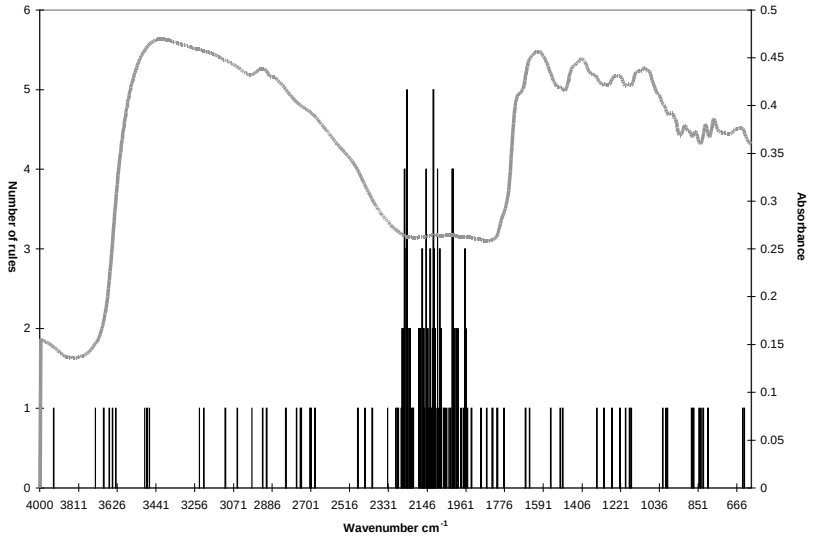
Run 28:

$$\mathbf{IF}\left[\left(\frac{A_{3476}}{A_{3499}}\right)\times \mathbf{IF}\left(A_{2480} \geq A_{883}\right)\right] \geq \left[\left(A_{2268}-A_{2133}\right)\times\left(A_{1558}+A_{3638}\right)+A_{2017}-A_{2110}\right]$$

THEN [Saline] ELSE [Control] (94% correct)

An analysis of which input variables (in terms of absorbances at particular wavenumbers) were selected showed that there were a few regions of the spectra that were consistently being used to form the different models (Figure 2a). In particular, the spectral region covering 2270 to 1960 cm^{-1} was used by most of the rules, and all of the best-performing models were based on a few small but distinct features within this critical region. The absolute differences between saline and non-saline grown samples in this region are relatively small, and so would not have been selected in a direct visual analysis, for example by using a difference spectrum.

(A)



(B)

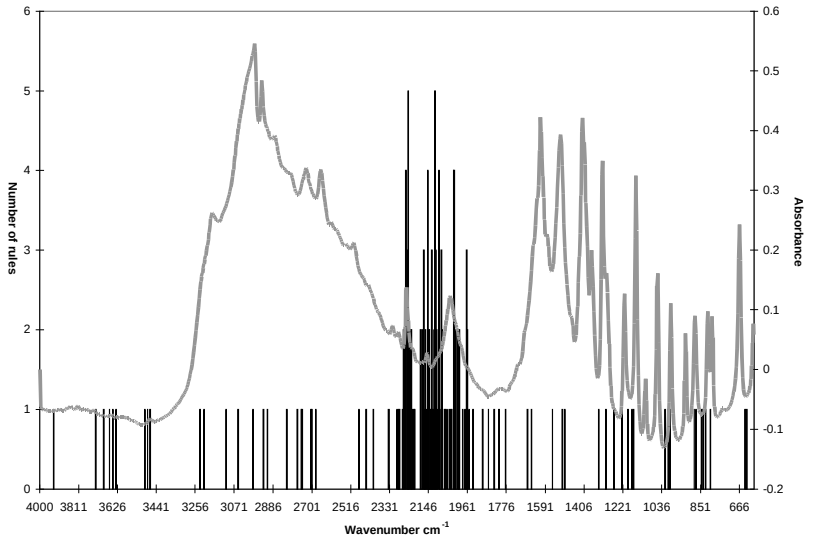


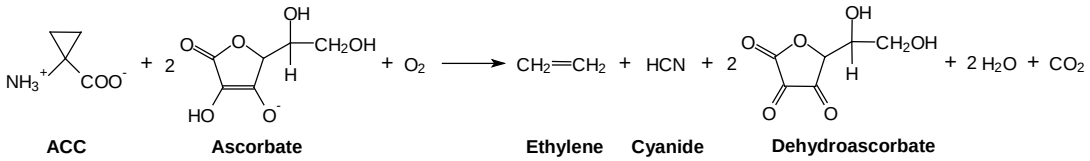
Figure 2

(A) The wavenumbers selected by the 30 GP rules are shown in reference to a spectrum averaged from the whole data set. The vertical lines represent the number of GP-derived rules that use particular wavenumbers to form a predictive model. The region from 2270 to 1960 wavenumbers is clearly important for producing good predictive models.

(B) The wavenumbers selected by the 30 GP rules are shown in reference to the FTIR spectrum of 0.1M β-cyanoalanine. The highly characteristic region identified by the GPs corresponds to the two distinctive peaks for the cyanide group in β-cyanoalanine.

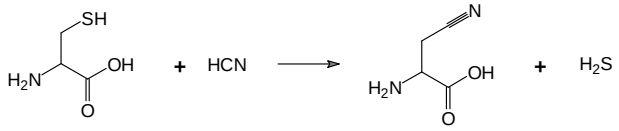
Quantum mechanics calculations showed that the only biochemically-reasonable functional groups that absorb strongly in this critical part of the IR spectrum are acetylenes ($R-C\equiv C-R'$) and cyanides or nitriles ($R-C\equiv N$), with the absorption due to a periodic stretching motion of the triple-bond. Acetylenes have a second characteristic vibration at approximately 3300 wavenumbers, a region unused by any of the GP-derived rules. If an acetylene group were responsible for the characteristic spectral features, this region would also be expected to be used by the rules. Therefore, the most likely candidate chemical moiety being identified by the predictive models as characteristic for tomatoes grown under saline conditions is a cyanide or nitrile group.

Cyanide, in the form of HCN, is formed in plants as a by-product during the conversion of the precursor 1-aminocyclopropane-1-carboxylic acid (ACC) to ethylene by the enzyme *ACC oxidase* [13]:



ACC **Ascorbate** **Ethylene** **Cyanide** **Dehydroascorbate**

Increased ethylene biosynthesis is known to occur in plants in response to stress and during climacteric fruit ripening [14]. It has previously been reported that tomato plants grown under saline conditions show enhanced ethylene production [15]. Hydrogen cyanide is an extremely toxic molecule, and plants have evolved a protective mechanism for its conversion to a less-harmful form. The cyanide produced during ethylene biosynthesis is rapidly converted to the less-toxic compound β-cyanoalanine by the enzyme *cyanoalanine synthase* (CAS):



Cysteine **Hydrogen cyanide** **Cyanoalanine** **Hydrogen sulphide**

The samples presented to the FTIR instrument were oven-dried in order to reduce the adverse effects of water in the spectra collected. This would also have driven off most of the other volatile compounds within the sample, such as free HCN. It is therefore a possibility that the compound being detected by the FTIR-GP analysis is β-cyanoalanine, or a related metabolite, which would be expected to remain in the dried samples. Quantum mechanical predictions of the spectral regions to which the cyanide group of β-cyanoalanine would contribute show a very high degree of correspondence with the critical regions selected by the GP models (Figure 2b). This is by no means conclusive evidence that β-cyanoalanine is the actual metabolite being detected as a discriminatory biomarker for salt-grown tomatoes, but it is consistent with this hypothesis.

From the observations and computational analyses, it seems reasonable to propose that tomato plants grown in conditions of high salinity produce enhanced levels of cyanide as a result of an increase in the production of the stress hormone ethylene. This is being detected by the analytical method described here, potentially in the form of β-cyanoalanine. It is possible that many of the toxic effects observed under saline-induced stress conditions are caused not by the salt *per se*, but by the concomitant increase in cyanide or a cyanide-containing compound as a result of increased ethylene biosynthesis. This proposed biochemical explanation is now subject to experimental verification using conventional biochemical techniques.

This study has shown that FTIR, in combination with GP, is a powerful new tool for the analysis of whole-tissue biological samples at the metabolome level. The technique is sensitive enough to detect changes in the levels of a single metabolite against the background of the entire cellular components, and can provide chemical information which can lead to the identification of the biochemicals which may be involved in metabolic processes under investigation. This method has the promise of becoming an extremely sensitive and discriminatory analytical tool which may be of crucial importance in the emerging field of functional genomics, and so help to advance the understanding of metabolic processes as yet unexplored by biological science.

Acknowledgements

RJG, JJR and DBK thank the UK EPSRC for financial support. MKW, JJR and DBK thank the UK BBSRC for financial support. HEJ and ARS and MAH acknowledge the financial support of the European Union INCO-DC programme. We thank Pat Causton and Dave Summers for their help in plant cultivation. We also thank Dr. Gary Salter for his assistance and encouragement.

References

[1] Mahmoud, M.H., El-Beltagy, A S, Helal, R M, Maksoud M A (1986) Tomato variety evaluation and selection for salt tolerance. Acta Horticulture 190, 559 - 565.

[2] Mahmoud, M.H., Jones, R A, El-Beltagy, A S (1986) Comparative responses to high salinity between salt-sensitive and salt-tolerant genotypes of tomato. Acta Horticulture 190, 533 - 543.

[3] Winson, M.K., Goodacre, R., Woodward, A.M., Timmins, É., Jones, A., Alsberg, B.K., Rowland, J.J. and Kell, D.B. (1997) Diffuse reflectance absorbance spectroscopy taking in chemometrics (DRASTIC) A hyperspectral FT-IR based approach to rapid screening for metabolite overproduction. Analytica Chema Acta 348, 273 - 282.

[4] Koza, J.R. (1992) Genetic Programming: On the Programming of computers by Means of Natural Selection., pp. 819 MIT Press, Cambridge, MA.

[5] Koza, J.R. (1994) Genetic Programming II: Automatic Discovery of Reusable Programs., pp. 746 MIT Press, Cambridge, MA.

[6] Koza, J.R. (1995) Survey of Genetic Algorithms and Genetic Programming. In: Wescon® 95 : E2. Neural-Fuzzy Technologies and Its Applications, pp. 589 - 594 IEEE, San Francisco, California, USA.

[7] Gilbert, R.J., Goodacre, R., Woodward, A.M. and Kell, D.B. (1997) Genetic programming: A novel method for the quantitative analysis of pyrolysis mass spectral data. Analytical Chemistry 69, 4381-4389.

[8] Taylor, J., Winson, M.K., Goodacre, R., Gilbert, R.J., Rowland, J.J. and Kell, D.B. (1998) Genetic Programming in the Interpretation of Fourier Transform Infrared Spectra: Quantification of Metabolites of Pharmaceutical Importance. In: Genetic Programming 1998 (Koza, J.R. et al., Eds.) Morgan Kaufmann, Madison, Wisconsin, USA.

[9] Taylor, J., Goodacre, R., Wade, W.G., Rowland, J.J. and Kell, D.B. (1998) The deconvolution of pyrolysis mass spectra using genetic programming: application to the identification of some Eubacterium species. Fems Microbiology Letters 160, 237-246.

[10] Langdon, W.B., Poli, R (1998) Fitness causes bloat: mutation. In: EuroGP '98, Vol. 1391, pp. 37 - 48 (Banzhaf, W., Poli, R, Schoenauer, M, Fogarty, T C, Ed.) Springer, Paris, France.

[11] Gilbert, R.J., Goodacre, R., Shann, B., Rowland, J.J. and Kell, D.B. (1998) Genetic Programming-Based Variable Selection for High-Dimensional Data. In: Genetic Programming 98: Proceedings of the Third Annual Conference, pp. 109 - 115 (Koza, J.R., Banzhaf, W,

Chellapilla, K, Deb, k, Dorigo, M, Fogel, D B, Garzon, M H, Goldberg, D E, Iba, H, Riolo, R L, Ed.) Morgan Kaufmann, Madison, Wisconsin, USA.
[12] Whitlock, M.C., Barton, N H (1997) The effective size of a subdivided population. Genetics 146, 427 - 441.
[13] Peiser, G.D., Wang, T.T., Hoffman, N.E., Yang, S.F., Liu, H.W. and Walsh, C.T. (1984) Formation of Cyanide From Carbon-1 of 1-Aminocyclopropane-1- Carboxylic Acid During Its Conversion to Ethylene. Proceedings of the National Academy of Sciences of the United States of America-Biological Sciences 81, 3059-3063.
[14] Hulme, A.C. (1970) The Biochemistry of Fruits and their Products. In: Food Science and Technology, Vol. 1 (Stewart, G.F., Chichester, C O, Galliver, G B, Morgan, A I, Mrak, E M, Scott, J K, von Sydow, E, Ed.) Academic Press, London.
[15] Mizrahi, Y. (1982) Effect of Salinity on Tomato Fruit Ripening. Plant Physiol. 69, 966 - 970.