A Sharp Fitness Function for the Problem of Finding Roots of Polynomial Equations Systems

Cruz E. Borges[1], Jose L. Montaña[2] and Luis M. Pardo[2]
[1]*DeustoTech Labs, Universidad de Deusto, Deusto, Spain*
[2]*Department of Mathematics, Statistics and Computing, Universidad de Cantabria, Santander, Spain*

Keywords: Non Linear Equations, Polynomial Equations, Evolutionary Algorithms.

Abstract: We experiment with several evolutionary algorithms for solving systems of polynomial equations with real coefficients. As a main difference with previous work, our algorithms can certify the correctness of the solutions they provide. This achievement is made possible by incorporating results from the field of numerical analysis to their fitness functions. We have performed an experimental comparison between the various proposed algorithms. The results of this comparison show that evolutionary and other local search algorithms can deal with the problem of solving systems of polynomial equations even for systems having many variables and high degree. Our main contribution is a nontrivial fitness function adjusted to the problem to be solved. This function is not based on any heuristics but on the fundamentals of numerical computation.

1 INTRODUCTION

Solving systems of non linear equations constitutes an important challenge both in Mathematics and Computer Science. Many problems in a variety of fields such as engineering, mechanics, medicine, chemistry, and robotics can be reduced to the problem of solving a system of non linear equations. Several methods and strategies have been proposed to address this problem, but so far one of the simplest cases, solving systems of polynomial equations with real coefficients, is still an open problem. Namely, the following remains open:

Problem 1 (Root finding problem for Systems of Polynomial Equations)**.** *Let $f_1 = 0, \ldots, f_n = 0$ be a system of n polynomial functions of degree bounded by d:*

$$\begin{aligned} f_1 &:= \sum_{i_1,\ldots,i_n} a_1^{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n} \\ &\vdots \\ f_n &:= \sum_{i_1,\ldots,i_n} a_n^{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}, \end{aligned}$$

where $\sum_{j=1}^{n} i_j \leq d$ and $a_j^{i_1,\ldots,i_n} \in \mathbb{R}$ for all j and $i_1, \ldots i_n$.

The Root finding problem for Systems of Polynomial Equations *is to find a point $\zeta \in \mathbb{R}^n$ such that $f_i(\zeta) = 0$ for all i, $1 \leq i \leq n$.*

The following is an example of this problem with three equations and three variables:

Example 2.

$$\begin{aligned} f_1 &= 5.38 \cdot 10^8 x_2^6 + 5.03 \cdot 10^8 x_2^5 + 8.95 \cdot 10^7 x_2^4 - \\ &\quad - 6.25 \cdot 10^{13} x_1^2 x_3 + 5.78 \cdot 10^6 x_2^3 + 1.07 \cdot 10^5 x_2^2 + \\ &\quad + 6.17 \cdot 10^2 x_2 + 1 \\ f_2 &= -5.03 \cdot 10^8 x_2^5 - 1.79 \cdot 10^8 x_2^4 + 6.25 \cdot 10^{13} x_1^2 x_2 + \\ &\quad + 1.88 \cdot 10^{14} x_1^2 x_3 - 1.73 \cdot 10^7 x_2^3 + 2.03 \cdot 10^7 x_1 x_2 - \\ &\quad - 4.29 \cdot 10^5 x_2^2 - 3.09 \cdot 10^3 x_2 - 6 \\ f_3 &= -5.55 \cdot 10^{15} x_1^2 + 1.11 \cdot 10^{16} x_1 x_3 + 1.8 \cdot 10^9 x_2 + 1. \end{aligned}$$

The case in which the system has the same number of equations as variables is one of the most important and widely studied, both theoretically and in practice (see, for example, (Dieudonné, 1985)). In fact, solving the case in which there are more equations than unknowns is rewarded with 1 million US dollars by the Clay Mathematics Institute (*P vs NP* Millennium Prize).

2 STATE OF THE ART

Traditionally there are, in addition to heuristics, two ways of addressing the root finding problem for polynomial systems of equations, the symbolic form (with

Borges C., Montaña J. and Pardo L..
A Sharp Fitness Function for the Problem of Finding Roots of Polynomial Equations Systems.
DOI: 10.5220/0005140002940301
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 294-301
ISBN: 978-989-758-052-9

exact calculations) and the numerical approximation (with approximate computations).

2.1 Symbolic and Numerical Methods

The standard way to solve the problem symbolically is via Gröbner bases (see (Cox et al., 1997; Becker and Weispfenning, 1993; Giusti et al., 2003) for an extensive review of this technique). Indeed, the algorithm described in the above mentioned papers, despite its doubly exponential complexity, is the method commonly implemented in symbolic solvers like Maple™, Mathematica™ or Matlab™ . In (Castro et al., 2003), a simply exponential lower bound for this problem is given (see also (Castro et al., 2001)). They also give a simple exponential algorithm for the problem. An implementation can be found in the symbolic solver Magma™ . See (Durvye and Lecerf, 2008) for the details.

On the other hand, in the framework of numerical computation, the problem of root finding is solved using the well known Newton's method that can be applied not only to polynomial systems of equations but also to non linear systems with good derivability properties. In (Blum et al., 1998) the authors initiated a series of papers with the aim of formalizing this technique and analyzing its inherent computational complexity when applied to polynomial systems. This analysis resulted in the so-called homotopy continuation methods. Recently, in another series of papers (see (Beltrán and Pardo, 2011) for a survey) the authors provided a probabilistic polynomial time algorithm for the case of systems having complex coefficients (i.e. looking for complex roots of systems of polynomials with complex coefficients). Unfortunately the same techniques presented several difficulties when applied to the case of systems having real coefficients (see (Borges and Pardo, 2008)). Other alternatives to Newton's method exist in the literature: Reduction Methods (see (Grapsa and Vrahatis, 2003)) and Interval Methods (see (Hentenryck et al., 1997) are well studied examples.

2.2 Evolutionary Methods

As it is well known, evolutionary algorithms provide a simple optimization method that mimics some process of biological evolution and, over some particular problems, can perform better than random or exhaustive search. There are several ways to translate the problem of root finding into an optimization problem suitable for applying evolutionary techniques. As an example, given the polynomial equation $f(x) = 0$, the usual way of translating the corresponding root finding problem into a minimization problem is just to minimize $g(x) = ||f(x)||$, where x can take values in $\mathbb{R}^n$. Here $||\cdot||$ denotes the Euclidean norm. This strategy can be easily extended to a system of polynomial equations. Given $f_1(x) = 0, \ldots, f_n(x) = 0$, the corresponding root finding problem is to minimize $g(x) = ||f_1(x)|| + \ldots, ||f_n(x)||$. Another natural way of making the translation of root finding into an optimization problem is to minimize $g(x) = \sum_{1 \leq i \leq n} [f_i(x)]^2$. This method is described for instance in (Mastorakis, 2005), see also (Kuri-Morales, 2003), and is used in many situations to solve many particular problems from a variety of fields. See the works by (Benedetti et al., 2006), (Tan et al., 2006), (K. Deb and Majumdar, 2004) as illustration of this technique. A remarkable result using a slightly different idea is found in (Grosan and Ajith Abraham, 2007) where a novel approach that transforms a system of nonlinear equations into a multiobjective optimization problem is presented. The obtained problem is then solved using the standard Pareto dominance relationship between solutions and an iterative strategy that evolves some random solutions in the search for optimal solutions. The technique uses principles from the evolutionary computation field and, apparently, is able to approximate the solutions even for large-scale systems of equations. The objection is that under these reductions and transformations the method can easily lead to *false roots* although it might work in some particular cases. See, for example, Figure 1. In this case, we can find a local minimum near 0.5 that is *fundamentally* far from a real root. We remark that any optimization method based on minimizing some suitable combination of the original equations cannot certify, in general, correctness of the provided solutions and has the same limitations as those pointed out by the example given in Figure 1 above.

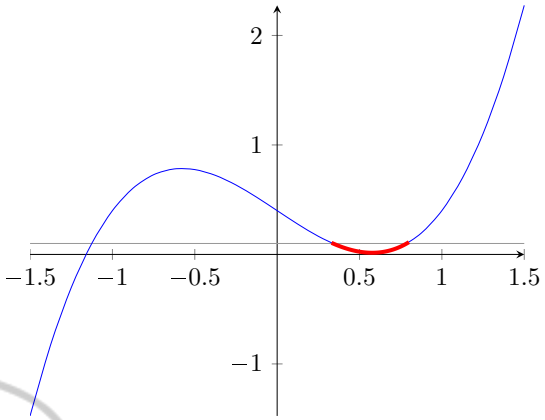

Figure 1: Example of a polynomial equation with probably bad properties.

3 BACKGROUND AND SUMMARY OF RESULTS

Because of the limitation for certifying correctness of the solutions present in the existing evolutionary methods, we propose –regardless of the evolutionary algorithm to be subsequently applied– a thorough study of the characteristics of the function to be optimized when trying to solve equations. The problem in its full generality is very difficult to address however; in the case of systems of polynomial equations, the theory of approximate zeroes may shed some light on it.

We recall some elementary notions of numerical analysis. Given a system of n polynomial equations with real coefficients and n variables as below,

$$\begin{array}{c} f_1(x_1,\ldots,x_n)=0 \\ f_2(x_1,\ldots,x_n)=0 \\ \ldots \\ f_n(x_1,\ldots,x_n)=0 \end{array}$$

let $f:\mathbb{R}^n \longrightarrow \mathbb{R}^n$, $f=(f_1,\ldots,f_n)$ be the polynomial function corresponding to such system. In addition let ζ be a root of f (i.e. $f_i(\zeta)=0$, $for\ 1\leq i\leq n$). The *Newton operator* is defined as follows.

$$N_f(x) := x - Df(x)^{-1}f(x),$$

where Df is the differential of the function f (also called the *Jacobian matrix* of f).

$$Df = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ & \cdots & \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

As usual Df^{-1} denotes the inverse of the matrix Df. Let x_k be the point obtained by applying k iterations of the Newton operator to some initial point $x_0 \in \mathbb{R}^n$. We say that x_0 is an *approximate zero* of ζ if the sequence x_k converges quadratically to ζ, i. e. the following holds for all values of k:

$$||x_{k+1}-\zeta|| \leq \frac{1}{2^{2^k-1}}||x_0-\zeta||.$$

In the next Section, Theorems 8 and 9, we show that there exists a function $\alpha^*(f,x)$ having the following properties:

- $\alpha^*(f,x)$ depends only on the system of equations f and requires essentially to compute the Jacobian matrix of f.
- There is a constant $\alpha_0 := \frac{13-3\sqrt{17}}{4}$ such that if $\alpha^*(f,x_0) < \alpha_0$ then x_0 is an approximate zero of f.
- There is a convex bounded region $\mathcal{B}_f \subset \mathbb{R}^n$ depending only of the function f such that approximate zeroes of f are guaranteed to belong to $\mathcal{B}_f \subset \mathbb{R}^n$.

As a main contribution, our approach avoids the problem of computing false solutions using $\alpha^*(f,x)$ as fitness function. The precise form in which α^* is defined is described in next Section. The way in which any evolutionary or local search algorithm can compute one or more solutions of a polynomial equations system is by driving the search trying to minimize the function $\alpha^*(f,x)$ and using as stopping criterion condition $\alpha^*(f,x) < \alpha_0$. The individuals of the evolutionary or local search algorithm must belong to region $\mathcal{B}_f$.

4 DEFINING THE FUNCTION α^*

Next theorem in (Blum et al., 1998) characterizes the points x_0 which are approximate zeroes of a polynomial system of equations:

Theorem 3 (α and γ-Theorem)**.** *Let $f:\mathbb{R}^n \longrightarrow \mathbb{R}^n$ be a system of polynomial equations, $x \in \mathbb{R}^n$ and*

$$\begin{aligned} \beta(f,x) &:= ||Df(x)^{-1}f(x)||, \\ \gamma(f,x) &:= \sup_{k>1}\left|\left|\frac{Df(x)^{-1}D^kf(x)}{k!}\right|\right|^{\frac{1}{k-1}}, \\ \alpha(f,x) &:= \beta(f,x)\gamma(f,x), \end{aligned}$$

$D^kf(x)$ being the k-th differential map of f in x, $||\cdot||$ the Euclidean norm and sup *the supreme value. Then, the following holds:*

- *There is a constant $\alpha_0 := \frac{13-3\sqrt{17}}{4}$ such that if $\alpha(f,x_0) < \alpha_0$ then x_0 is an approximate zero of f.*
- *Let ζ be a root of f, then there is a constant $\gamma_0 := \frac{3-\sqrt{7}}{2}$ such that for all $x_0 \in \mathbb{R}^n$ verifying $||x_0-\zeta|| < \frac{\gamma_0}{\gamma(f,\zeta)}$ is an approximate zero of f.*

The quantities previously defined do not exhibit good mathematical properties. In addition they are difficult to compute. In order to overcome these difficulties Smale introduced the notion of non-linear condition number μ_{norm} (in the same spirit of Turing's linear condition numbers of matrices) and did it in the following terms:

Definition 4 (Non linear condition number)**.** *Let f be a system of polynomial equations and $x \in \mathbb{R}^n$. The non-linear condition number of f at x, $\mu_{norm}(f,x)$, is defined as follows:*

$$\mu_{norm}(f,x) := ||(Df(x))^{-1}\Delta||,$$

where Δ is a matrix that depends only on the degrees of the system f and $||\cdot||$ is the usual matrix norm. The non-linear condition number of f is:

$$\mu(f) := \max_{\zeta \in V(f)} \mu_{norm}(f,\zeta),$$

where $V(f)$ denote the set of all roots of f.

The non linear condition number $\mu_{norm}(f,x)$ has better mathematical properties than α and γ, mainly orthogonal invariance, see (Blum et al., 1998) and references therein for more details, and in addition it is easy to compute. The following theorem gives an estimation of its expected value:

Theorem 5. *The expected value of the non linear condition number of a system of polynomial equations is bounded by:*

$$\mathrm{E}[\mu(f)] \leq 2\sqrt{2}C(n+1)^{3/2}\left(\frac{N\mathcal{D}}{\pi}\right)^{1/4},$$

where $N := \sum_{i=1}^{n}\binom{n+d_i}{n}$, d_i denotes the degree of the variable x_i in the system, $\mathcal{D} := \prod_{i=1}^{n} d_i$ is the Bézout number and C is a universal constant such that $5 < C < 7$.

The connection between the non linear condition number and the previous quantities is summarized in the following result:

Theorem 6 (μ-theorem)**.** *Using the previous notation:*

$$\gamma(f,x) < \frac{D^{3/2}}{2}\mu_{nomr}(f,x),$$

where D is the maximum degree of the variables of the system f.

Using this bound it is easy to prove the next corollary.

Corollary 7 (Root distances)**.** *Using the previous notation, the expected value of the distance d between any two roots of the system of polynomial equations f satisfies the following inequality*

$$\mathrm{E}(d) > \frac{2u_0\pi^{1/4}}{CD^{3/2}(N\mathcal{D})^{1/4}(n+1)^{3/2}},$$

where $u_0 \approx 0.05992$ and $5 < C < 7$ are universal constants.

Proofs of the previously stated results can be found in (Blum et al., 1998), (Beltrán and Pardo, 2009) and (Borges and Pardo, 2008).

As we have seen before it is easy to approximate a true zero provided that we have found an *approximate zero*. Hence, using Theorem 3 it is enough to find a point x in $\mathbb{R}^n$ such that $\alpha(f,x) < \alpha_0$. We can transform this criterion into an optimization problem in the evolutionary (or local search) sense: the function to be minimized would be $\alpha(f,x)$ and the stopping criterion would be "to find a point x such that the previous bound holds".

In this situation two problems arise:

1. To find a bounded set in which one can guarantee that there is an *approximate zero* and
2. to provide an efficient algorithm to calculate, or at least to give a bound on, $\alpha(f,x)$.

Solving the first item provides a finite region in which one can generate the initial population as well as a limit for all the other populations. Solving the second item will provide a suitable fitness function.

For solving the first problem we use the following result.

Theorem 8. *With probability $1-\varepsilon$ the norm of any root is bounded above by*

$$\frac{\Gamma\left(\frac{1}{4}\right)^2\mathcal{D}}{\varepsilon^2},$$

where $\mathcal{D}$ denotes the Bézout number and $\Gamma(\cdot)$ is the hyper-geometric Γ function (see (Gradshteĭn et al., 2007) for a comprehensive list of its properties).

Due of lack of space and to preserve readability we omit the proof of this theorem. The main ideas can be found in (Borges and Pardo, 2008). For the second problem we show the following result. We give a short proof in order to illustrate the technique.

Theorem 9 (α^*-theorem)**.** *Let f be a system of polynomial equations, $0 < \varepsilon \leq \frac{1}{2}$ and $x \in \mathbb{R}^n$ with norm bounded by $\frac{\Gamma(1/4)^2\mathcal{D}}{\varepsilon^2}$. We define $\alpha^*(f,x)$ as:*

$$\alpha^*(f,x) := \beta(f,x)\mu_{norm}(f,x) = \\ = \frac{D^{3/2}||x||}{2}||Df(x)^{-1}f(x)||\,||Df(x)^{-1}\Delta||,$$

Then, x_0 is an approximate zero of f with probability $1-\varepsilon$ if any of the following conditions holds:

$$\alpha^*(f,x_0) \leq \alpha_0 \quad or \quad \alpha_c^*(f,x_0) := \alpha^*(f,x_k) \leq \alpha_0,$$

x_k denotes the k-th iteration of the Newton operator. Here k has order

$$k \in O\left(\log\log\left(\frac{(Dn)^{3/2}N^{1/4}\mathcal{D}^{5/4}}{\varepsilon^{5/2}}\right)\right),$$

where D denotes the maximum degree of the system f, d_i denotes the degree of the variable x_i in the system, $N := \sum_{i=1}^{n}\binom{n+d_i}{n}$, $\mathcal{D}$ is the Bézout number and α_0 is the constant defined in Theorem 3.

Proof. The first part is trivial. Let us suppose that the first condition holds, i.e. $\alpha^*(f,x_0) \leq \alpha_0$. Then, using the bounds given in Theorem 6, we conclude that $\alpha(f,x_0) \leq \alpha^*(f,x_0) < \alpha_0$; as a consequence we realize that x_0 is an approximate zero.

Let us suppose now that $\alpha^*(f,x_0) > \alpha_0$ but $\alpha(f,x_0) \leq \alpha_0$. In this case, x_0 is a true approximate zero hence the iteration of the Newton's operator $N_f^k(x_0)$ converges quadratically to a root ζ. Hence we have

$$\alpha^*(f,x_k) \leq \frac{D^{3/2}}{2}\frac{1}{2^{2^k-1}}||x_0-\zeta||\mu_{norm}(f,x_0)$$

Let $0 < \varepsilon \leq \frac{1}{2}$, using Corollary 5 and Corollary 8 we have:

- $\mu_{norm}(f,x_0)$ is bounded, with probability $1-\varepsilon$, by:

$$\mu_{norm}(f,x_0) \leq C(n+1)^{3/2}\left(\frac{N\mathcal{D}}{4\pi}\right)^{1/4},$$

  where C is the constant defined in Corollary 5.
- $||x_0-\zeta||$ is bounded, with probability $1-\varepsilon$, by:

$$||x_0-\zeta|| \leq \frac{\Gamma\left(\frac{1}{4}\right)^2\mathcal{D}}{\varepsilon^2},$$

  ζ is the root for which the iteration converges and Γ is the usual Γ function.

After $k \geq \log\log\left(L(D(n+1))^{3/2}N^{1/4}\mathcal{D}^{5/4}\varepsilon^{-5/2}\right)$ we have that $\alpha^*(f,x_k) < \alpha_0$ and as a consequence, x_k is an approximate zero. Note that here L is the universal constant

$$L := \frac{\alpha_0\Gamma\left(\frac{1}{4}\right)^2 C}{(4\pi)^{1/4}}.$$

□

Observe that both functions α^* and α_c^* can be easily computed. Moreover, the previous result gives an alternative version of Theorem 3. Hence we can use α_c^* as fitness function.

5 EVOLUTIONARY ALGORITHMS

In this section we present several evolutionary and local search algorithms that use the results to approximate zeros of a system of polynomial equations with real coeffuicients. Listing 1 presents in a homogenized way some common evolutionary an local search techniques. We choose the operators of the evolutionary/local search algorithm following four of the traditional evolutionary/local search methods: Evolution Strategy (Michalewicz and Schoenauer, 1996), Simulated Annealing (Laarhoven and Aarts, 1987), Differential Evolution (Price et al., 2005) and Particle Swarm Optimization (Poli, 2008). See (Koza, 1992) and the following references for a more complete descriptions of these algorithms. Next follows a more detailed description.

Listing 1: Pseudocode of an Evolutionary Algorithm. Note that t represents the current generation, T represents the maximum number of generations, op_1 and op_2 represent the probability of using Operator_1 and Operator_2 respectively and finally m_1, m_2 and m_r represent the number of individuals needed to perform the Operator_1, Operator_2 and the Reproduction Operator respectively.

```
1  generate initial population p_0
   compute fitness of p_0
3  elite := best individual of p_0
   t := 1
5  while (t < T) or (found solution?)
   p_aux := empty population
7    for i in 1 to length p_{t-1}
       aux := empty set of individuals
9      r := random number in [0,1]
       switch r
11       case 0 ≤ r ≤ op_1
           select individuals a_1,...,a_{m_1}
13         aux := Operator_1 over
           a_1,...,a_{m_1}
15       end case
         case op_1 < r ≤ op_2
17         select individuals a_1,...,a_{m_2}
           aux := Operator_2 over
19         a_1,...,a_{m_2}
         end case
21       case op_2 < r ≤ 1
           aux := select a_1,...,a_{m_r}
23       end case
       end switch
25     compute fitness of aux
       reproduction :=
27     selection(aux, individuals)
       p_aux := p_aux ∪ reproduction
29   end for
     p_aux := p_aux ∪ elite
31   elite := best individual of p_aux
     p_t := p_{t-1}
33   t := t+1
   end while
```

ESα *Evolution Strategy*: Operator_1 and Operator_2 are respectively the arithmetic crossover and non uniform mutation operator (see (Michalewicz et al., 1994)). Given two parents $p_1, p_2 \in \mathbb{R}^n$ the arithmetic crossover produces the children c_1 and c_2 according to the following expressions:

$$c_1 = \lambda p_1 + (1-\lambda)p_2 \qquad c_2 = \lambda p_2 + (1-\lambda)p_1,$$

where λ is a random number in $[0,1]$. For the non uniform mutation operator, given $p \in \mathbb{R}^n$ and

$[a,b]^n$ the bounds given by Lemma 8 below, this operator produces a child $c = (c_1,\ldots,c_n)$ randomly chosen among the following two cases:

$$c_k = p_k + \Delta(t, b - p_k) \qquad c_k = p_k - \Delta(t, p_k - a),$$

for all $k = 1,\ldots,n$, where t is the actual generation and Δ is defined as:

$$\Delta(t,y) := yr\left(1 - \frac{t}{T}\right),$$

r is a random number in $[0,1]$ and T is the maximum number of generations.

The selection of the individuals is a 2-tournament selection and the reproduction step is based in a raking scheme where the best two (or one in the case of the mutation operator) *different* individuals between parents and children survive in the next generation. This procedure preserves diversity of the population. Finally the elitist operator takes the best individual of the previous generation and places it in the current one. See (Koza, 1992) for a more detailed description of all these operators.

SA *Simulated Annealing*: The principal difference between this algorithm and the previous one lies in the reproduction operator: when children are better than the parents, the best one passes to the next generation but when they are worse they can still pass with hight probability (0.9). The rest of the operators behave as in ESα.

DE *Differential Evolution*: The difference between this algorithm and the Evolution Strategy lies in its first operator. Instead of using the arithmetic crossover it uses the so called differential crossover: given three parents p_1, p_2 and $p_3 \in \mathbb{R}^n$, the differential crossover produces the following child:

$$c = p_1 + \lambda(p_2 - p_3),$$

where λ is a random number in $[0,1]$. The remaining operators behave in the same way a ESα.

PSO *Particle Swarm Optimization*: In this algorithm the crossover an the elitist operators are as follows. Instead of preserving the best element of the population (that we denote by p^e) we preserve, for every position i in the population, the best element that has held this position. We denote this element by p_i^e. Given a parent $p_i \in \mathbb{R}$ the child c_i produced by the swarm crossover is:

$$c_i = \lambda_1 p_i + \lambda_2(p_i^e - p_i) + \lambda_3(p^e - p_i),$$

where λ_1, λ_2 and λ_3 are random numbers in $[0,1]$.

6 EXPERIMENTAL RESULTS

In order to experiment with the algorithm we have carried an extensive test in a Core 2 Duo SU4100 with 4 GB of RAM using a Gentoo operating system up to date.

- We have tested the algorithms with systems of sparse and dense polynomials. We represent sparse polynomials by means of *Straight Line Programs* (see (Giusti et al., 1998) for an exhaustive reference of this data structure and (Alonso et al., 2009) for an introductory reference to their use as an artificial intelligence tool). Dense encoding is as usual, i.e. the vector of coefficients.
- We have randomly generated 100 systems in every case. For the dense encoding we have generated every coefficient as a normal random variable with mean 0 and variance 1. In the sparse case we have used the method described in (Alonso et al., 2009).
- We have experimented with systems having the same number $n \in \{3,5,7,10\}$ of variables as equations. The degree was at least 2 in the sparse case and at least 3 in the dense case. Note that in the last case, the number of roots of the polynomials could be $\{27,243,2187,59049\}$ and the number of coefficients would be $\{60,280,840,2860\}$.
- The executions finish after 100 generations or when an *approximate zero* was found.
- The probability of operator$_1$ has been set to 0.9 and the probability of operator$_2$ to 0.05 so that the reproduction probability is 0.05. In the case of the Simulated Annealing algorithm the probabilities have been reversed.
- The population has been set to 100 individuals.
- All of our methods are elitist, so we add the best individual of the previous generation to the next one.
- We have also implemented an evolution strategy using the Euclidean norm of the evaluated polynomial as the fitness function. In the following tables we have denoted this algorithm by **ES**$||\cdot||$.

We also wanted to compare our algorithms with the symbolic algorithm used in Matlab^{TM} but in the case $n = 3$, using dense encoding of polynomials it takes more than 10 hours to solve **only** one of the systems of equations we have generated, even using a very powerful machine.

The experimental results for both the sparse and the dense encoding are quite similar although they present some remarkable differences. In Table 1 we present the results for the sparse encoding case. As

Table 1: Results using sparse encoding.

(a) Successful runs (%).

n	PSO	DE	SA	ESα	ES$\|\|\cdot\|\|$
3	**83%**	83%	83%	79%	68%
5	**95%**	81%	78%	56%	31%
7	**92%**	68%	68%	21%	9%
10	**90%**	33%	41%	16%	3%

(b) Execution time (s).

n	PSO	DE	SA	ESα	ES$\|\|\cdot\|\|$
3	2.18	2.23	2.21	2.3	**0.2**
5	2.57	7.3	6.91	6.44	**0.48**
7	8.87	25.36	21.83	23.74	**0.79**
10	29.14	78.45	72.09	83.5	**1.3**

Table 2: Results using dense encoding.

(a) Successful runs (%).

n	PSO	DE	SA	ESα	ES$\|\|\cdot\|\|$
3	**100%**	100%	100%	100%	75%
5	**100%**	91%	88%	28%	26%
7	**100%**	0%	1%	0%	2%
10	**100%**	0%	0%	0%	0%

(b) Execution time (s).

n	PSO	DE	SA	ESα	ES$\|\|\cdot\|\|$
3	0.04	0.02	0.02	0.02	**0.02**
5	0.25	1.39	0.83	0.5	**0.06**
7	2.9	29.21	29.04	29.91	**0.21**
10	30.35	174.28	174.68	179.86	**0.99**

it can be seen all methods perform quite well in the first two cases ($n = 3$ and $n = 5$) but their success rate starts to decrease quickly with exception of the Particle Swarm Optimization method (PSO). Note that the Evolution Strategy (ES$||\cdot||$) is the fastest one but also it presents the worst success rate and very often reports false solutions. This good execution time is due to the simplicity of its fitness function.

In Table 2 we display the results for the dense encoding case. In this case the particle swarm optimization algorithm has exhibited a remarkable success rate while for the other methods the performance is quite week.

The differences in the execution time between each group of problems (sparse and dense encodings) are due to the different programming tools used in each case. While for the dense encoding very fast and specific classes were used, in the sparse case we have used a general purpose and slow library. In all cases, algorithms performed quite fast when compared with the symbolic solution provided by Matlab^{TM}. We also want to remark that the traditional Evolution Strategy had a better behavior than expected, however some false solutions were detected.

7 CONCLUSIONS AND FUTURE WORK

Our experimentation supports the idea that most common evolutionary algorithms and local search techniques can deal quite well with the problem of solving randomly generated polynomial equation systems if the α^* fitness functions is used. The two main features of the α^* fitness function are:

- it seems to work well in practice with any reasonable evolutionary or local search technique,
- it can certify the correctness of the provided solutions just by checking an inequality which is easier and more robust, when performing numerical computations, than checking zero equalities.

To our knowledge this last feature makes α^* different from any other fitness function previously used to solve polynomial equation systems. Experimentation with systems coming from specific fields has been deliberately excluded since our goal was to show that the α^* function "generically" works. Application to real world problems is part of a future development that requires an extensive comparison with a variety of heuristics designed *ad hoc* for the problems they try to solve and is out of the scope of this paper.

A main future development will be to extend the proposed method to analytic functions. There are several extensions of the α-theorem (see for example (Dedieu and Kim, 2002) or (Dedieu et al., 2003)) but the equivalent to our α^*-theorem and bounds ensuring the existence of at least one root of the system in certain bounded regions is not clear in this more general context.

ACKNOWLEDGEMENTS

This work is partially supported by spanish grant TIN2011-27479-C04-04.

REFERENCES

Alonso, C., Montaña, J., Puente, J., and Borges, C. (2009). A New Linear Genetic Programming Aproach Based on Straight Line Programs: Some Theoretical and Exeperimental Aspects. *International Journal on Artificial Intelligence Tools*, 18(5):757–781.

Becker, T. and Weispfenning, V. (1993). *Gröbner Bases, Graduate Texts in Mathematics*. Springer-Verlag, New York.

Beltrán, C. and Pardo, L. (2009). On Smale's 17 Problem: Average Polynomial Solver for Affine and Proyective Solutions. *J.Amer. Math. Soc.*, 22:363–385.

Beltrán, C. and Pardo, L. (2011). Fast linear homotopy to find approximate zeros of polynomial systems. *Found. Comput. Math.*, 11(1):95–129.

Benedetti, A., Farina, M., and Gobbi, M. (2006). Evolutionary multiobjective industrial design: the case of a racing car tire-suspension system. *IEEE Trans. Evolutionary Computation*, 10(3):230–244.

Blum, L., Cucker, F., Shub, M., and Smale, S. (1998). *Complexity and Real Computation*. Springer-Verlag, New York.

Borges, C. and Pardo, L. (2008). On the Probability Distribution of Data at Points in Real Complete Intersection Varieties. *Journal of Complexity*, 24(4):492–523.

Castro, D., Giusti, M., Heintz, J., Matera, G., and Pardo, L. M. (2003). The hardness of polynomial equation solving. *Foundations of Computational Mathematics*, 3(4):347–420.

Castro, D., Pardo, L., Hägele, K., and Morais, J. (2001). Kronecker's and newton's approaches to solving: A first comparison. *J. Complexity*, 17(1):212–303.

Cox, D., Little, J., and O'Shea, D. (1997). *Ideals, Varieties, and Algorithms. Undergraduate Texts in Mathematics*. Springer-Verlag, New York.

Dedieu, J., Priouret, P., and Malajovich, G. (2003). Newton's method on riemannian manifolds: covariant alpha theory. *IMA Journal of Numerical Analysis*, 23(3):395–419.

Dedieu, J.-P. and Kim, M.-H. (2002). Newton's method for analytic systems of equations with constant rank derivatives. *Journal of Complexity*, 18(1):187 – 209.

Dieudonné, J. (1985). *History of Algebraic Geometry: An Outline of the History and Development of Algebraic Geometry*. Chapman & Hall.

Durvye, C. and Lecerf, G. (2008). A concise proof of the Kronecker polynomial system solver from scratch. *Expositiones Mathematicae*, 26(2):101–139.

Giusti, M., Heintz, J., Morais, J., Morgenstern, J., and Pardo, L. (1998). Straight-line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124(1–3):101–146.

Giusti, M., Pardo, L., and Weispfenning, V. (2003). *Algorithms of Commutative Algebra and Algebraic Geometry: Algorithms for Polynomial Ideals and Their Varieties, Handbook of Computer Algebra*. Springer Verlag.

Gradshteĭn, I., Ryzhik, I., Jeffrey, A., and Zwillinger, D. (2007). *Table of integrals, series and products*. Academic Press. Academic.

Grapsa, T. N. and Vrahatis, M. N. (2003). Dimension reducing methods for systems of nonlinear equations and unconstrained optimization: A review. *A Review, Recent Adv. Mech. Related Fields*, pages 215–225.

Hentenryck, P. V., Mcallester, D., and Kapur, D. (1997). Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34:797–827.

K. Deb, K. Mitra, R. D. and Majumdar, S. (2004). Towards a better understanding of the epoxy-polymerization process using multi-objective evolutionary computation. *Chem. Eng. Sci.*, 59(20):4261–4277.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press.

Kuri-Morales, A. (2003). Solution of simultaneous nonlinear equations using genetic algorithms. In *WSEAS Transactions on SYSTEMS, Issue 1, vol. 2*, pages 44–51.

Laarhoven, P. and Aarts, E., editors (1987). *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA.

Mastorakis, N. E. (2005). Solving non linear equations via genetic algorithms. In *Proceedings of the 6th WSEAS Int. Conf. on EVOLUTIONARY COMPUTING*, pages 24–28.

Michalewicz, Z., Logan, T., and Swaminathan, S. (1994). Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 84–97. World Scientic.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4:1–32.

Poli, R. (2008). Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, 2008:1–10.

Price, K., Storn, R., and Lampinen, J. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 1 edition.

Tan, K. C., Yang, Y. J., and Goh, C. K. (2006). A distributed cooperative coevolutionary algorithm for multiobjective optimization. *IEEE Trans. Evolutionary Computation*, 10(5):527–549.