2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A Comparison of Algorithms for the Optimization of Fermentation Processes

Rui Mendes Isabel Rocha Eugénio C. Ferreira Miguel Rocha

Abstract— **The optimization of biotechnological processes is a complex problem that has been intensively studied in the past few years due to the economic impact of the products obtained from fermentations.**

In fed-batch processes, the goal is to find the optimal feeding trajectory that maximizes the final productivity. Several methods, including Evolutionary Algorithms (EAs) have been applied to this task in a number of different fermentation processes.

This paper performs an experimental comparison between Particle Swarm Optimization, Differential Evolution and a real-valued EA in three distinct case studies, taken from previous work by the authors and literature, all considering the optimization of fed-batch fermentation processes.

I. Introduction

A number of valuable products such as recombinant proteins, antibiotics and amino-acids are produced using fermentation techniques. Additionally, biotechnology has been replacing traditional manufacturing processes in many areas like the production of bulk chemicals, due to its relatively low requirements regarding energy and environmental costs. Consequently, there is an enormous economic incentive to develop engineering techniques that can increase the productivity of such processes.

However, these are typically very complex, involving different transport phenomena, microbial components and biochemical reactions. Furthermore, the nonlinear behavior and time-varying properties, together with the lack of reliable sensors capable of providing direct and on-line measurements of the biological state variables limits the application of traditional control and optimization techniques to bioreactors.

Under this context, there is the need to consider quantitative mathematical models, capable of describing the process dynamics and the interrelation among relevant variables. Additionally, robust global optimization techniques must deal with the model's complexity, the environment constraints and the inherent noise of the experimental process [3].

In fed-batch fermentations, process optimization usually encompasses finding a given nutrient feeding trajectory that maximizes productivity. Several optimization methods have been applied in this task. It has been shown that, for simple bioreactor systems, the problem can be solved analytically [24].

Rui Mendes and Miguel Rocha are with Department of Informatics and the Centro de Ciências e Tecnologias da Computação, Universidade do Minho, Braga, Portugal (email: azuki@di.uminho.pt, mrocha@di.uminho.pt). Isabel Rocha and Eugénio Ferreira with the Centro de Engenharia Biológica da Universidade do Minho (email: irocha@deb.uminho.pt, ecferreira@deb.uminho.pt).

Numerical methods make a distinct approach to this dynamic optimization problem. Gradient algorithms are used to adjust the control trajectories in order to iteratively improve the objective function [4].

In contrast, dynamic programming methods discretize both time and control variables to a predefined number of values. A systematic backward search method in combination with the simulation of the system model equations is used to find the optimal path through the defined grid. However, in order to achieve a global optimum the computational burden is very high [23].

An alternative approach comes from the use of algorithms from the *Evolutionary Computation (EC)* field, which have been used in the past to optimize nonlinear problems with a large number of variables. These techniques have been applied with success to the optimization of feeding or temperature trajectories [14][1], and, when compared with traditional methods, usually perform better [20][6].

In this work, the performance of different algorithms belonging to three main groups - *Evolutionary Algorithms* (EA), *Particle Swarm* (PSO) and *Differential Evolution* (DE) - was compared, when applied to the task of optimizing the feeding trajectory of fed-batch fermentation processes. Three test cases taken from literature and previous work by the authors were used. The algorithms were allowed to run for a given number of function evaluations that was deemed to be enough to achieve acceptable results. The comparison among the algorithms was based on their final result and on the convergence speed.

The paper is organized as follows: firstly, the fed-batch fermentation case studies are presented; next, PSO, DE and a real-valued EA are described; the results of the application of the different algorithms to the case studies are presented; finally, the paper presents a discussion of the results, conclusions and further work.

II. Case studies: fed-batch fermentation processes

In fed-batch fermentations there is an addition of certain nutrients along the process, in order to prevent the accumulation of toxic products, allowing the achievement of higher product concentrations.

During this process the system states change considerably, from a low initial to a very high biomass and product concentrations. This dynamic behavior motivates the development of optimization methods to find the optimal input feeding trajectories in order to improve the process. The typical input in this process is the substrate inflow rate time profile.

0-7803-9487-9/06/$20.00/©2006 IEEE

For the proper optimization of the process, a white box mathematical model is typically developed, based on differential equations that represent the mass balances of the relevant state variables.

A. Case study I

In previous work by the authors, a fed-batch recombinant *Escherichia coli* fermentation process was optimized by *EAs* [17][18]. This was considered as the first case study in this work and will be briefly described next.

During the aerobic growth of the bacterium, with glucose as the only added substrate, the microorganism can follow three main different metabolic pathways:

- Oxidative growth on glucose:

$$k_1 S + k_5 O \xrightarrow{\mu_1} X + k_8 C \tag{1}$$

- Fermentative growth on glucose:

$$k_2 S + k_6 O \xrightarrow{\mu_2} X + k_9 C + k_3 A \tag{2}$$

- Oxidative growth on acetic acid:

$$k_4 A + k_7 O \xrightarrow{\mu_3} X + k_{10} C \tag{3}$$

where S, O, X, C, A represent glucose, dissolved oxygen, biomass, dissolved carbon dioxide and acetate components, respectively. In the sequel, the same symbols are used to represent the state variables' concentrations (in g/kg); μ_1 to μ_3 are time variant specific growth rates that nonlinearly depend on the state variables, and k_i are constant yield coefficients.

The associated dynamical model can be described by the following equations:

$$\frac{dX}{dt} = (\mu_1 + \mu_2 + \mu_3)X - DX \tag{4}$$

$$\frac{dS}{dt} = (-k_1\mu_1 - k_2\mu_2)X + \frac{F_{in,S}S_{in}}{W} - DS \tag{5}$$

$$\frac{dA}{dt} = (k_3\mu_2 - k_4\mu_3)X - DA \tag{6}$$

$$\frac{dO}{dt} = (-k_5\mu_1 - k_6\mu_2 - k_7\mu_3)X + OTR - DO \tag{7}$$

$$\frac{dC}{dt} = (k_8\mu_1 + k_9\mu_2 + k_{10}\mu_3)X - CTR - DC \tag{8}$$

$$\frac{dW}{dt} \simeq F_{in,S} \tag{9}$$

being D the dilution rate, $F_{in,S}$ the substrate feeding rate (in kg/h), W the fermentation weight (in kg), OTR the oxygen transfer rate and CTR the carbon dioxide transfer rate.

The kinetic behavior, expressed in the rates μ_1 to μ_3, was given by specific functions of the state variables, that is out of the scope of the present work but can be found in [16].

The purpose of the optimization is to determine the feeding rate profile ($F_{in,S}(t)$) that maximizes the productivity of the process, defined as the units of product (recombinant protein) formed per unit of time. In this case, it is usually related with the final biomass obtained, when the duration of the process is pre-defined. Thus, a *performance index (PI)* is defined by the following expression:

$$PI = \frac{X(T_f)W(T_f) - X(0)W(0)}{T_f} \tag{10}$$

The relevant state variables are initialized with the following values: $X(0) = 5$, $S(0) = 0$, $A(0) = 0$, $W(0) = 3$. Due to limitations in the feeding pump capacity, the value of $F_{in,S}(t)$ must be in the range $[0.0; 0.4]$. Furthermore, the following constraint is defined over the value of W: $W(t) \leq 5$. The final time (T_f) is set to 25 hours.

B. Case study II

This system is a fed-batch bioreactor for the production of ethanol by *Saccharomyces cerevisiae*, firstly studied by Chen and Huang [5]. The aim is to find the substrate feed rate profile that maximizes the yield of ethanol.

The model equations are the following:

$$\frac{dx_1}{dt} = g_1 x_1 - u\frac{x_1}{x_4} \tag{11}$$

$$\frac{dx_2}{dt} = -10 g_1 x_1 + u\frac{150 - x_2}{x_4} \tag{12}$$

$$\frac{dx_3}{dt} = g_2 x_1 - u\frac{x_3}{x_4} \tag{13}$$

$$\frac{dx_4}{dt} = u \tag{14}$$

where x_1 is the cell mass, x_2 the substrate concentration, x_3 the ethanol concentration, x_4 the volume of the reactor, u the feeding rate.

On the other hand, the kinetic variables g_1 and g_2 are given by the following algebraic equations:

$$g_1 = \frac{0.408}{1 + \frac{x_3}{16}} \frac{x_2}{0.22 + x_2} \tag{15}$$

$$g_2 = \frac{1}{1 + \frac{x_3}{71.5}} \frac{x_2}{0.44 + x_2} \tag{16}$$

The aim of the optimization is to find the feeding profile (u) that maximizes the following *performance index*:

$$PI = x_3(T_f)x_4(T_f) \tag{17}$$

The final time is set to $T_f = 54$ (hours), and the initial values for the state variables are the following: $x_1(0) = 1$, $x_2(0) = 150$, $x_3(0) = 0$ and $x_4(0) = 10$. Additionally, there are physical constraints over the variables, namely: $0 \leq x_4(t) \leq 200$ for all time points and the feeding rate $0 \leq u(t) \leq 12$.

C. Case study III

This system is a model for the production of a secreted foreign protein using baker's yeast as the host organism in a fed-batch bioreactor, developed by Park and Ramirez [15]. The substrate feed flow rate is the only control variable and the system is governed by the following differential equations:

$$\frac{dx_1}{dt} = \frac{4.75A(x_2 - x_1)}{0.12 + A} - \frac{ux_1}{x_5} \quad (18)$$

$$\frac{dx_2}{dt} = \frac{x_3 x_4 e^{-5x_4}}{0.1 + x_4} - \frac{ux_2}{x_5} \quad (19)$$

$$\frac{dx_3}{dt} = (A - \frac{u}{x_5})x_3 \quad (20)$$

$$\frac{dx_4}{dt} = -7.3Ax_3 - \frac{u(x_4 - 20)}{x_5} \quad (21)$$

$$\frac{dx_5}{dt} = u \quad (22)$$

where x_1, x_2, x_3 and x_4 are the concentrations of secreted protein (units/L), total protein (units/l), cells (g/l) and substrate (g/l) respectively; x_5 is the fermenter's volume (l) and u the feed rate (l/h).

The specific growth A (h^{-1}) follows substrate inhibition kinetics and is given by:

$$A = \frac{21.87x_4}{(x_4 + 0.4)(x_4 + 62.5)} \quad (23)$$

The aim of the optimization is to find the feeding profile (u) that maximizes the following *PI*:

$$PI = x_1(T_f)x_5(T_f) \quad (24)$$

The final time is set to $T_f = 15$ (hours) and the initial values for relevant state variables are the following: $x_1(0) = 0$, $x_2(0) = 0$, $x_3(0) = 1$, $x_4(0) = 5$ and $x_5(0) = 1$. The feed rate is constrained to the range $u(t) \in [0.0; 3.0]$.

III. The Algorithms

The optimization task is to find the feeding trajectory, represented as an array of real-valued variables, that yields the best performance index. Each variable will encode the amount of substrate to be introduced into the bioreactor, in a given time unit, and the solution will be given by the temporal sequence of such values. In this case, the size of the genome would be determined based on the final time of the process (T_f) and the discretization step (d) considered in the numerical simulation of the model, given by the expression: $\frac{T_f}{d}$.

However, as the resulting genome would be very large (typically with 5000 genes, for case study I), feeding values were defined only at certain equally spaced points, and the remaining values are linearly interpolated. The size of the genome (G) becomes:

$$G = \frac{T_f}{dI} + 1 \quad (25)$$

where I stands for the number of points within each interpolation interval. The value of d used in the experiments was $d = 0.005$, for case studies I, II and III.

The evaluation process, for each individual in the population, is achieved by running a numerical simulation of the defined model, given as input the feeding values in the genome. The numerical simulation is performed using *ODEToJava*, a package of ordinary differential equation solvers, using a linearly implicit implicit/explicit (IMEX) Runge-Kutta scheme used for stiff problems [2]. The fitness value is then calculated from the final values of the state variables according to the *PI* defined for each case.

A. Particle Swarm Optimization

A particle swarm optimizer uses a population of particles that evolve over time by flying through space. The particles imitate their most successful neighbors by modifying their velocity component to follow the direction of the most successful position of their neighbors.

Each particle is defined by:

$$P_t^{(i)} = \langle x_t, v_t, p_t, e_t \rangle$$

$x_t \in \mathbb{R}^d$ is the current position in the search space; $p_t \in \mathbb{R}^d$ is the position visited by the particle in the past that had the best function evaluation; $v_t \in \mathbb{R}^d$ is a vector that represents the direction in which the particle is moving, it is called the 'velocity'; e_t is the evaluation of p_t under the function being optimized, i.e. $e_t = f(p_t)$.

Particles are connected to others in the population via a predefined topology. This can be represented by the adjacency matrix of a directed graph $M = (m_{ij})$, where m_{ij} = 1 if there is an edge from particle i to particle j and $m_{ij} = 0$ otherwise.

At each iteration, a new population is produced by allowing each particle to move stochastically toward its previous best position and at the same time toward the best of the previous best positions of all other particles to which it is connected.

The following is an outline of a generic PSO.

1) Set the iteration counter, $t = 0$.
2) Initialize each $x_0^{(i)}$ and $v_0^{(i)}$ randomly.
   Set $p_0^{(i)} = x_0^{(i)}$.
3) Evaluate each particle and set $e_0^{(i)} = f(p_0^{(i)})$.
4) Let $t = t + 1$ and generate a new population, where each particle i is moved to a new position in the search space according to:
   (i) $v_t^{(i)} = velocity\_update(v_{t-1}^{(i)})$.
   (ii) $x_t^{(i)} = x_{t-1}^{(i)} + v_t^{(i)}$.
   (iii) Constrain solutions to the bounds and update $x_t^{(i)}$ and $v_t^{(i)}$ accordingly.
   (iv) Evaluate the new position, $e = f(x_t^{(i)})$.
   (v) If the new position is better than the previous best, update the particle's previous best position. i.e if $e < e_{t-1}^{(i)}$ then let $p_t^{(i)} = x_t^{(i)}$ and $e_t^{(i)} = e$ else let $p_t^{(i)} = p_{t-1}^{(i)}$ and $e_t^{(i)} = e_{t-1}^{(i)}$.

If it is necessary to constrain solutions to the bounds, the offending coordinate is reset to the nearest limit and the velocity is changed accordingly. Clerc and Kennedy [7] introduced the use of a factor called the 'constriction factor', symbolized by χ, into the velocity update equation. The velocity update equation for this scheme is given by:

$$velocity\_update(v_{t-1}^{(i)}) = \chi[v_{t-1}^{(i)} + r_1 c_1 (p_{t-1}^{(i)} - x_{t-1}^{(i)}) + r_2 c_2 (p_{t-1}^{(g)} - x_{t-1}^{(i)})]$$

where c_1 and c_2 are constants known as the 'individual' and 'social' constants respectively, that represent the weight accorded to the influence of the particle's personal memory, and the memory of its neighborhood respectively; r_1 and r_2 are random variables selected from $U(0,1)$. New values for r_1 and r_2 are selected for each dimension of the updated velocity vector as it is being computed; $p_{t-1}^{(g)}$ is the previous best position of the particle in i's neighborhood at time $t-1$ that has the best previous best evaluation of all particles in that neighborhood. In other words, $g = arg\{min\{e_j(t-1)|j \in N(i)\}\}$, where $N(i)$ is the neighborhood of particle i. A common use of the parameters involves choosing $c_1 = c_2 = 2.05$ and $\chi = 0.729$.

B. Fully Informed Particle Swarm

The only difference between this method and the canonical particle swarm resides in the velocity update. An approach to the velocity update equation that involves utilizing information from all members of a particle's neighborhood was proposed by Mendes et al [12]. In this case, each member of the particle's neighborhood contributes to the new direction that the particle will travel in. Mendes' formulation allows for weighted contributions from each neighbor, with the possibility of equal weightings. The velocity update equation can be computed by

$$velocity\_update(v_{t-1}^{(i)}) = \sum_{j \in N(i)} r \cdot \frac{(c_1 + c_2)}{|N(i)|} \cdot (p_{t-1}^{(j)} - x_{t-1}^{(i)})$$

where $r \sim U(0,1)$ and $N(i)$ is the neighborhood (the set of the particles) of particle i. In this paper, the population topology used is *von Neumann* [11]. In this topology each particle is connected to four others, in a lattice that wraps around on itself.

C. Differential Evolution

DE is a population-based approach to function optimization that generates trial individuals by calculating vector differences between other randomly selected members of the population.

The following is an outline of a variant of the DE algorithm called *DE/rand/1* that uses a binomial crossover [22]. For clarity, the computation of the new trial vector has been shown separately from the crossover operation that selects only some of the dimensions of the trial vector.

1) Initialize the population;
2) Evaluate the population;
3) Generate a new population where each candidate individual is generated in parallel according to:
   (i) Randomly select 3 distinct individuals r_1, r_2, r_3 from the population that are different from i;
   (ii) Generate a trial vector based on the formula $\vec{t} = \vec{x}_{r_1} + F \cdot (\vec{x}_{r_2} - \vec{x}_{r_3})$
   (iii) Incorporate coordinates of this vector with probability CR, using at least one coordinate;
   (iv) If the candidate is not valid, change its invalid coordinates by resetting them to the closest bound;
   (v) Evaluate the candidate;
   (vi) Use the candidate in the new generation if it is at least as good as the current individual;
4) Loop to 3 unless the termination criterion is met.

Various schemes are currently in use for DEs [21]. Each scheme varies with respect to the number of other random individuals that are used to construct a new trial vector, as well as with respect to whether or not the current individual or the global best individual will be used as part of that computation. Three schemes are considered in this paper. These are shown below along with the corresponding trial vector generation formula. x_{r_j}, $2 \leq j \leq 3$ represent distinct randomly selected individuals that are different from the current individual x_i; x_{best} is the best individual and x_t is selected as the best of two randomly selected individuals from the population that are different from the current individual x_i.

DE/rand/1 $\vec{t} = \vec{x}_{r_1} + F(\vec{x}_{r_2} - \vec{x}_{r_3})$
DE/best/1 $\vec{t} = \vec{x}_{best} + F(\vec{x}_{r_2} - \vec{x}_{r_3})$
DE/tourn/1 $\vec{t} = \vec{x}_t + F(\vec{x}_{r_1} - \vec{x}_{r_2})$

D. Real-Valued EA

In this work, we adopted a real-valued EA that provided good results in previous work [18][19]. The overall structure of the *EA* is,

BEGIN
  Initialize time ($t = 0$).
  Generate and evaluate the initial population (P_0).
  WHILE NOT (termination criteria) DO
    Select from P_t individuals for reproduction.
    Apply the genetic operators to breed the offspring.
    Evaluate the offspring.
    Insert the offspring into the next population (P_{t+1}).
    Select the survivors from P_t to be kept in P_{t+1}.
    Increase current time ($t = t + 1$).
END

Regarding the reproduction step, both mutation and crossover operators were taken into account.

Two mutation operators were used, namely:

- *Random Mutation*, which replaces one gene by a new randomly generated value, within the range $[min_i, max_i]$ [13]; and
- *Gaussian Mutation*, which adds to a given gene a value taken from a Gaussian distribution, with a zero mean and a standard deviation given by $\frac{max_i - min_i}{4}$ (i.e., small perturbations will be preferred over larger ones).

where $[min_i; max_i]$ is the range of values allowed for gene i.

In both cases, an innovation is introduced: the mutation operators are applied to a variable number of genes (a value that is randomly set between 1 and 10 in each application).

p-value code	Condition
3	$p < 0.001$
2	$0.001 \leq p < 0.01$
1	$0.01 \leq p < 0.05$
N	$p \geq 0.05$

TABLE I
ENCODING USED IN THE PRESENTATION OF P-VALUES OF THE PAIRWISE T-TESTS.

On the other hand, the following crossover operators were chosen:

- *Two-Point crossover*, a standard *Genetic Algorithm* operator [13], applied in the traditional way;
- *Arithmetical*, where each gene in the offspring will be a linear combination of the values in the ancestors' chromosomes [13];
- *Sum* where the offspring genes denote the sum or the subtraction of the genes in the parents.

A set of experiments was conducted in order to find the best set of genetic operators for this problem [18]. The best result was obtained using an alternative that contemplates the use of all genetic operators described above. In this case the crossover operators are responsible for breeding 50% of the offspring and the mutation operators the remaining 50%.

All operators were constrained to respect the limits of the gene's values, i.e., when an operator creates a gene value outside of the allowed range, the value in the offspring is equal to the one in the parents. Different ranges can be defined to different genes at distinct locations.

The selection procedure is done by converting the fitness value into a linear ranking in the population, and then applying a roulette wheel scheme. In each generation, 50% of the individuals are kept from the previous generation, and 50% are bred by the application of the genetic operators.

IV. EXPERIMENTS

A. Methodology

When comparing algorithms, it is necessary to have a way to conclude that a certain approach is better than the other ones. As the procedure is empirical, conclusions may only be drawn after some statistical validation. The results presented in this paper use 95% confidence intervals. Additionally, we adopted the use of t-tests [9] for two-sample comparisons. We decided to present our results using a symbolic encoding of the p-values of t-tests. The reason for this is to improve the readability that will allow us to visually interpret the result of the test. The encoding used is presented in Table I.

When performing multiple pairwise comparisons, it is important to use a correction of the p-value of each test. There are several corrections that may be used. One of the simplest and most widely used is the *Bonferroni correction*. We decided to use the *Bonferroni step-down* (also called *Holm*) correction, which is very similar to the *Bonferroni*, but a little less stringent [10].

What makes a good algorithm? Even when a statistical test cannot find a significant difference between two algorithms (e.g., because the confidence interval of one of them is too wide), we are interested in a *reliable* method: one that always gives us good results. Thus, we prefer an algorithm with a good average and a narrow confidence interval.

B. Parameter Settings and Test Conditions

When solving a real world problem, the main concern is to have a tool that may be applied to the problem with as few fine-tuning as possible. We are mainly interested in the results and not in a thorough study about the algorithms involved. We do not wish to go through the cumbersome task of testing the valuation of all the parameters of these algorithms until we find a setting that is perfect for the problem at hand. Furthermore, these experiments take a long time[1] and we usually have time constraints to get good results. Thus, we decided to choose the standard configurations for each algorithm that were either validated by experimental results or suggested by previous studies.

Due to the previous experience of the authors with the real-valued EA, each run was stopped after 200,000 function evaluations. In the case of canonical particle swarm (henceforth denoted *CanPso*) or fully informed particle swarm (denoted *Fips*) the population size was 20 and the other parameters have the usual values given in the literature. The neighborhood topologies selected were *gbest* for *CanPso* and *von Neumann* for *Fips* [11].

For DE, the population size was set to 20, F was set to 0.5, CR to 0.6 and the schemes used were *DE/rand/1* (*DE*), *DE/best/1* (*DEBest*) and *DE/tourn/1* (*DETourn*). In terms of the real-valued EA, the population size was set to 200. Due to the amount of time it took to run the experiments, only 20 runs were performed with each algorithm.

Three values were tested for the parameter I in each test case, thus varying the number of variables to optimize.

C. Results for case I

Table II presents the results of the algorithms on case I for the different values of I. In all cases, the 95% confidence intervals for the PI are presented for 40000, 100000, 200000 function evaluations. We decided to probe PI at these time-steps to estimate the possibility of terminating the runs earlier whilst still maintaining good quality solutions.

The best solution found was by *DE* when $I = 100$. Using a higher number of points provides a better trajectory. It could be argued that it does not make sense to maintain the same population sizes when increasing the number of variables. However, it is interesting to note that certain algorithms maintain a very similar performance on all three instances while the performance of others degrades a lot. *DE* and *Fips* maintain the same performance on all three instances, closely followed by *DETourn*. Additionally, we remark the narrowness of the confidence intervals of *DE* on all instances of the problem.

It is also interesting to realize that *DE*, *DETourn* and *Fips* do not improve much after 40,000 NFEs. This is especially

[1] As much as 24 hours per run in case II with $I = 100$

Algorithm	PI 40k NFEs	PI 100k NFEs	PI 200k NFEs
CanPso	2.5154 ± 0.7123	2.5563 ± 0.7091	2.5641 ± 0.7168
DE	9.3693 ± 0.0570	9.4738 ± 0.0052	9.4770 ± 0.0028
DEBest	2.7077 ± 0.1921	2.7419 ± 0.2115	2.7936 ± 0.2176
DETourn	9.1044 ± 0.1983	9.2913 ± 0.1240	9.3596 ± 0.1114
EA	7.9371 ± 0.1355	8.5161 ± 0.0883	8.8121 ± 0.0673
Fips	9.1804 ± 0.1642	9.4280 ± 0.0551	9.4528 ± 0.0538
CanPso	3.9338 ± 0.8631	3.9909 ± 0.8551	3.9910 ± 0.8551
DE	9.4728 ± 0.0003	9.4728 ± 0.0001	9.4728 ± 0.0001
DEBest	4.2618 ± 1.1503	4.4745 ± 1.1054	4.4919 ± 1.1041
DETourn	9.4374 ± 0.0265	9.4500 ± 0.0214	9.4534 ± 0.0187
EA	8.3242 ± 0.1053	8.8004 ± 0.0993	8.9937 ± 0.0849
Fips	9.4691 ± 0.0056	9.4729 ± 0.0000	9.4729 ± 0.0000
CanPso	7.1461 ± 1.1152	7.1461 ± 1.1152	7.1461 ± 1.1152
DE	9.4351 ± 0.0000	9.4351 ± 0.0000	9.4351 ± 0.0000
DEBest	7.6932 ± 0.8321	7.6937 ± 0.8321	7.6937 ± 0.8321
DETourn	9.4099 ± 0.0551	9.4099 ± 0.0551	9.4099 ± 0.0551
EA	8.7647 ± 0.1441	9.0137 ± 0.1421	9.1324 ± 0.1320
Fips	9.4351 ± 0.0000	9.4351 ± 0.0000	9.4351 ± 0.0000

TABLE II

RESULTS FOR CASE I FOR $I = 100$ (51 VARIABLES), $I = 200$ (26 VARIABLES) AND $I = 500$ (11 VARIABLES) RESPECTIVELY.

true for *DE*. What can we conclude about convergence speed? We plotted the average of the best fitness in the population for each algorithm. Figure 1 presents the convergence curve of the algorithms for $I = 200$. The convergence curves for other values of I are quite similar to this one. *DE*, *DETourn* and *Fips* do not improve much after a certain mark. The convergence speed of *EA* is slower when compared with the other algorithms.

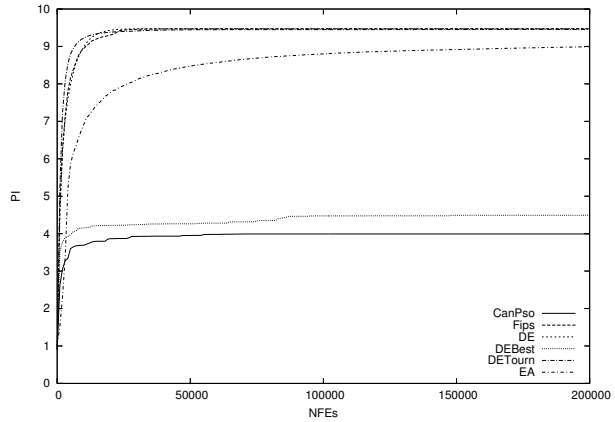

Fig. 1. Convergence of the algorithms for case I when $I = 200$.

Both *CanPso* and *DEBest* are clearly inferior and seem to be stuck in very poor local optima. This is to be expected given that both algorithms are quite greedy. We are sure that the result of *CanPso* could be improved with a more sparsely connected neighborhood topology but previous studies suggest it would not beat *Fips* [12].

We decided to test for statistical differences among the algorithms. Table III shows the pairwise t-test results for all values of I. The codes of the p-values are separated by dashes and correspond to the ordering of I.

As can be seen from consulting the table, there is no

	CanPso	DE	DEBest	DETourn	EA
DE	3-3-1				
DEBest	N-N-N	3-3-1			
DETourn	3-3-1	N-N-N	3-3-1		
EA	3-3-1	3-3-2	3-3-1	3-3-1	
Fips	3-3-1	N-N-N	3-3-1	N-N-N	3-3-2

TABLE III

PAIRWISE T-TEST WITH THE HOLM P-VALUE ADJUSTMENT FOR THE ALGORITHMS OF CASE I. THE P-VALUE CODES CORRESPOND TO $I = 100$, $I = 200$ AND $I = 500$ RESPECTIVELY.

statistically significant difference between *DE*, *DETourn* and *Fips* in this problem. We can conclude that *DEBest* and *CanPso* are inferior to the other ones, while the *EA* stands in an intermediate level between the two groups.

D. Results for case II

Table IV presents the results of the algorithms on case II. In this case, the best result was found by *DETourn* or *DE* when $I = 540$. Both of these algorithms also present narrow confidence intervals, which means that they are reliable. *EA* is a strong contender in this problem but needs more time to achieve good results. *Fips* somehow lost its edge, especially when the number of variables increases.

Algorithm	PI 40k NFEs	PI 100k NFEs	PI 200k NFEs
CanPso	19142.9 ± 131.3	19146.5 ± 133.7	19157.7 ± 140.4
DE	20383.6 ± 21.0	20396.5 ± 20.5	20402.3 ± 21.2
DEBest	19144.3 ± 162.5	19149.4 ± 161.7	19151.3 ± 161.4
DETourn	20365.2 ± 26.1	20396.7 ± 15.2	20402.7 ± 14.7
EA	19876.0 ± 94.4	20127.4 ± 74.1	20235.2 ± 73.7
Fips	19413.6 ± 163.2	19413.7 ± 163.2	19413.7 ± 163.2
CanPso	19385.2 ± 284.3	19386.4 ± 284.3	19406.8 ± 272.5
DE	20379.4 ± 11.6	20397.2 ± 13.9	20406.9 ± 14.5
DEBest	19418.1 ± 290.0	19421.0 ± 290.4	19430.5 ± 293.5
DETourn	20362.7 ± 52.4	20380.4 ± 42.7	20394.3 ± 32.8
EA	20151.8 ± 69.7	20335.1 ± 54.1	20394.7 ± 23.1
Fips	19818.0 ± 160.7	19818.9 ± 161.1	19818.9 ± 161.1
CanPso	19914.7 ± 243.3	19917.7 ± 244.2	19934.3 ± 259.6
DE	20388.0 ± 9.4	20406.7 ± 5.6	20416.8 ± 3.7
DEBest	20190.0 ± 224.3	20213.5 ± 194.6	20218.9 ± 189.6
DETourn	20400.3 ± 5.2	20414.7 ± 4.2	20419.9 ± 3.8
EA	20237.7 ± 52.9	20340.1 ± 39.4	20371.7 ± 25.6
Fips	20233.9 ± 120.8	20234.0 ± 120.8	20234.0 ± 120.8

TABLE IV

RESULTS FOR CASE II FOR $I = 100$ (109 VARIABLES), $I = 200$ (55 VARIABLES) AND $I = 540$ (21 VARIABLES) RESPECTIVELY.

Table V shows the comparison of the algorithms. As can be seen, *CanPso* continues to be the worst contender but *DEBest* is not a very bad choice when the number of variables is small. *EA* is still beaten by *DE* and *DETourn* in most cases. Figure 2 presents the convergence curve of the algorithms. *DE* and *DETourn* converge fast (around 40,000 NFEs); *Fips* gets stuck in a plateau that is higher than the one of *DEBest* and *CanPso*; *EA* converges slowly but is steadily improving. It seems that, given enough time, *EA* finds similar solutions to either *DE* and *DETourn*.

	CanPso	DE	DEBest	DETourn	EA
DE	3-3-1				
DEBest	N-N-N	3-3-N			
DETourn	3-3-1	N-N-N	3-3-N		
EA	3-3-1	2-N-1	3-3-N	2-N-1	
Fips	N-N-N	3-3-N	N-N-N	3-3-N	3-3-N

TABLE V

PAIRWISE T-TEST WITH THE HOLM P-VALUE ADJUSTMENT FOR THE ALGORITHMS OF CASE II. THE P-VALUE CODES CORRESPOND TO $I = 100$, $I = 200$ AND $I = 540$ RESPECTIVELY.

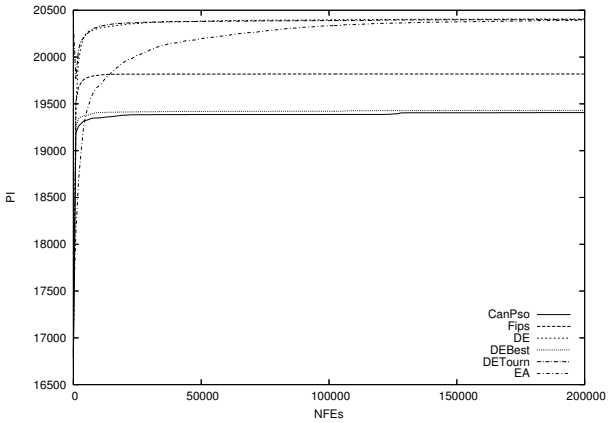

Fig. 2. Convergence of the algorithms for case II for $I = 200$.

E. Results for case III

Table VI presents the results of the algorithms on case III. This case seems to be quite simple and most algorithms find similar results. *DE*, *Fips* and *EA* are the best algorithms in this problem because of their reliability: they have narrow confidence intervals. *DETourn* seems to be a little less reliable, but its confidence intervals are still small enough.

Table VII shows the comparison of the algorithms for this problem. In this case, most algorithms are not statistically different. This is the case when we turn to the reliability of the algorithms to draw conclusions. As we stated before, most algorithms find similar solutions, which indicates that this case is probably not a good benchmark to compare algorithms.

Figure 3 presents the convergence curve of the algorithms for $I = 100$. In this case *DE*, *DETourn* and *Fips* converge very fast; *EA* has a slower convergence rate; *CanPso* and *DEBest* get stuck in local optima.

V. CONCLUSIONS AND FURTHER WORK

This paper compares canonical particle swarm (*CanPso*), fully informed particle swarm (*Fips*), a real-valued EA (*EA*) and three schemes of differential evolution (*DE*, *DEBest* and *DETourn*) in three test cases of optimizing the feeding trajectory in fed-batch fermentation. Each of these problems was tackled with different numbers of points (i.e., different values for I) to interpolate the feeding trajectory. This is a trade off: the more variables we have, the more precise the curve is but the harder it is to optimize.

Algorithm	PI 40k NFEs	PI 100k NFEs	PI 200k NFEs
CanPso	27.069 ± 1.751	27.370 ± 1.836	27.579 ± 1.681
DE	32.641 ± 0.029	32.674 ± 0.002	32.680 ± 0.001
DEBest	30.774 ± 1.004	30.775 ± 1.004	30.775 ± 1.004
DETourn	32.624 ± 0.057	32.629 ± 0.056	32.631 ± 0.056
EA	32.526 ± 0.025	32.633 ± 0.013	32.670 ± 0.008
Fips	32.625 ± 0.100	32.629 ± 0.099	32.630 ± 0.099
CanPso	30.668 ± 1.281	30.668 ± 1.281	30.668 ± 1.281
DE	32.646 ± 0.002	32.650 ± 0.000	32.651 ± 0.000
DEBest	31.766 ± 0.684	31.768 ± 0.685	31.768 ± 0.685
DETourn	32.648 ± 0.002	32.650 ± 0.002	32.650 ± 0.002
EA	32.553 ± 0.015	32.627 ± 0.005	32.643 ± 0.002
Fips	32.650 ± 0.001	32.651 ± 0.000	32.651 ± 0.000
CanPso	31.914 ± 0.662	31.914 ± 0.662	31.914 ± 0.662
DE	32.444 ± 0.000	32.444 ± 0.000	32.444 ± 0.000
DEBest	31.913 ± 0.700	31.914 ± 0.700	31.914 ± 0.700
DETourn	32.441 ± 0.005	32.441 ± 0.005	32.441 ± 0.005
EA	32.413 ± 0.012	32.439 ± 0.003	32.443 ± 0.001
Fips	32.444 ± 0.000	32.444 ± 0.000	32.444 ± 0.000

TABLE VI

RESULTS FOR CASE III FOR $I = 50$ (61 VARIABLES), $I = 100$ (31 VARIABLES) AND $I = 200$ (16 VARIABLES) RESPECTIVELY.

	CanPso	DE	DEBest	DETourn	EA
DE	2-N-N				
DEBest	1-N-N	1-N-N			
DETourn	2-N-N	N-N-N	1-N-N		
EA	2-N-N	N-3-N	1-N-N	N-3-N	
Fips	2-N-N	N-N-N	1-N-N	N-N-N	N-3-N

TABLE VII

PAIRWISE T-TEST WITH THE HOLM P-VALUE ADJUSTMENT FOR THE ALGORITHMS OF CASE III. THE P-VALUE CODES CORRESPOND TO $I = 50$, $I = 100$ AND $I = 200$ RESPECTIVELY.

A new *DE* scheme was presented: *DE/tourn/1*. The rationale behind this scheme was that it is a compromise between *DE/rand/1* and *DE/best/1*. The more individuals participate in the tournament and the more it should behave like *DE/best/1*. In these problems, the results of *DE* were very similar to the ones of *DETourn*. In some cases *DE* had narrower confidence intervals and therefore it seems that a more complicated scheme is not helpful in these problems. However, more research needs to be performed over a wider range of problems to ensure this conclusion. *DEBest* did not work well on these problems. In fact, it was one of the lowest performers: it suffered from premature convergence most of the time.

When compared to the other algorithms, *DE* and *DETourn* seemed to be able to achieve good results consistently and fast. From all the algorithms tested, they are the ones we recommend the most. *Fips* was a good contender and in the majority of the tests found good results and was as fast as *DE* and *DETourn*. However, it got stuck on local optima in case II. *CanPso* was one of the worst performers.

EA was slow to converge but reliable. If you can afford the computational time needed, it always finds good solutions. However, in some problems it requires a large number of function evaluations. Given that the computational time needed for these problems is quite large, it is a good reason

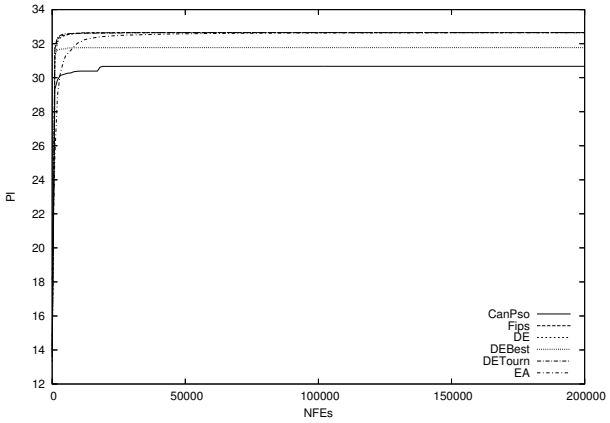

Fig. 3. Convergence of the algorithms for case III when $I = 100$.

to choose *DE* instead.

Previous work by the authors [19] developed a new representation in EAs in order to allow the optimization of a time trajectory with automatic interpolation. It would be interesting to develop a similar approach within *DE* or *Fips*. Another area of future research is the consideration of on-line adaptation, where the model of the process is updated during the fermentation process. In this case, the good computational performance of *DE* is a benefit, if there is the need to re-optimize the feeding given a new model and values for the state variables are measured on-line.

ACKNOWLEDGMENTS

This work was supported in part by the Portuguese Foundation for Science and Technology under project POSC/EIA/59899/2004. The authors wish to thank Project SeARCH (Services and Advanced Research Computing with HTC/HPC clusters), funded by FCT under contract CONC-REEQ/443/2001, for the computational resources made available.

REFERENCES

[1] P. Angelov and R. Guthke. A Genetic-Algorithm-based Approach to Optimization of Bioprocesses Described by Fuzzy Rules. *Bioprocess Engin.*, 16:299–303, 1997.

[2] Spiteri Ascher, Ruuth. Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25:151–167, 1997.

[3] J.R. Banga, C. Moles, and A. Alonso. Global Optimization of Bioprocesses using Stochastic and Hybrid Methods. In C.A. Floudas and P.M. Pardalos, editors, *Frontiers in Global Optimization - Nonconvex Optimization and its Applications*, volume 74, pages 45–70. Kluwer Academic Publishers, 2003.

[4] A.E. Bryson and Y.C. Ho. *Applied Optimal Control - Optimization, Estimation and Control*. Hemisphere Publication Company, New York, 1975.

[5] C.T. Chen and C. Hwang. Optimal Control Computation for Differential-algebraic Process Systems with General Constraints. *Chemical Engineering Communications*, 97:9–26, 1990.

[6] J.P. Chiou and F.S. Wang. Hybrid Method of Evolutionary Algorithms for Static and Dynamic Optimization Problems with Application to a Fed-batch Fermentation Process. *Computers & Chemical Engineering*, 23:1277–1291, 1999.

[7] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[8] J. Stuart Hunter George E. P. Box, William G. Hunter. *Statistics for experimenters: An introduction to design and data analysis*. NY: John Wiley, 1978.

[9] Cyril Harold Goulden. *Methods of Statistical Analysis, 2nd ed*. John Wiley & Sons Ltd., 1956.

[10] S Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[11] J. Kennedy and R. Mendes. Topological structure and particle swarm performance. In David B. Fogel, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, Honolulu, Hawaii, May 2002. IEEE Computer Society.

[12] Rui Mendes, James Kennedy, and José Neves. The fully informed particle swarm: Simple, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[13] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, USA, third edition, 1996.

[14] H. Moriyama and K. Shimizu. On-line Optimization of Culture Temperature for Ethanol Fermentation Using a Genetic Algorithm. *Journal Chemical Technology Biotechnology*, 66:217–222, 1996.

[15] S. Park and W.F. Ramirez. Optimal Production of Secreted Protein in Fed-batch Reactors. *AIChE J*, 34(9):1550–1558, 1988.

[16] I. Rocha. *Model-based strategies for computer-aided operation of recombinant E. coli fermentation*. PhD thesis, Universidade do Minho, 2003.

[17] I. Rocha and E.C. Ferreira. On-line Simultaneous Monitoring of Glucose and Acetate with FIA During High Cell Density Fermentation of Recombinant E. coli. *Analytica Chimica Acta*, 462(2):293–304, 2002.

[18] M. Rocha, J. Neves, I. Rocha, and E. Ferreira. Evolutionary algorithms for optimal control in fed-batch fermentation processes. In G.Raidl et al., editor, *Proceedings of the Workshop on Evolutionary Bioinformatics - EvoWorkshops 2004, LNCS 3005*, pages pp.84–93. Springer, 2004.

[19] Miguel Rocha, Isabel Rocha, and Eugénio Ferreira. A new representation in evolutionary algorithms for the optimization of bioprocesses. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 484–490. IEEE Press, 2005.

[20] J.A. Roubos, G. van Straten, and A.J. van Boxtel. An Evolutionary Strategy for Fed-batch Bioreactor Optimization: Concepts and Performance. *Journal of Biotechnology*, 67:173–187, 1999.

[21] Rainer Storn. On the usage of differential evolution for function optimization. In *1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS 1996)*, pages 519–523. IEEE, 1996.

[22] Rainer Storn and Kenneth Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *IEEE International Conference on Evolutionary Computation*, pages 842–844. IEEE, May 1996.

[23] A. Tholudur and W.F. Ramirez. Optimization of Fed-batch Bioreactors Using Neural Network Parameters. *Biotechnology Progress*, 12:302–309, 1996.

[24] V. van Breusegem and G. Bastin. Optimal Control of Biomass Growth in a Mixed Culture. *Biotechnology and Bioengineering*, 35:349–355, 1990.