DYNAMIC OPTIMIZATION OF CHEMICAL ENGINEERING PROCESSES BY EVOLUTIONARY METHOD

Q.T. Pham
School of Chemical Engineering and Industrial Chemistry
University of New South Wales, Sydney 2052, Australia.

Keywords:
Evolutionary method, Optimization, dynamic optimization

ABSTRACT

An evolutionary method is proposed for the constrained optimization of chemical engineering processes. Apart from the classical mutation, crossover and creep (small mutation), it makes use of several novel reproductive operators: shift, smoothing, extrapolation and swapping. An adaptive mutation rate is used to guard against stalling at local peaks. The method was able to solve dynamic optimization problems involving constrained time-dependent vectors, such as those arising in process control and inverse heat transfer. In addition, the method solves reputedly difficult test problems such as Shwefel's and Grierwangk's functions better than any known previous method.

INTRODUCTION

This paper presents an evolutionary method for optimizing chemical engineering processes, particularly (but not exclusively) those in which the control variables can be naturally arranged into a series. Typically, these processes may involve a series of reactors or other process equipment, a plug-flow reactor with position-dependent conditions, or a batch process with time-dependent operating conditions. For example, the time-temperature regime (rate of heating and cooling) plays a crucial importance in areas as varied as metallurgy and food processing. The operating conditions will almost always be subjected to an upper and a lower bound imposed by physical, practical or legal constraints.

Assuming that a mathematical representation of the objective function can be written, the process can be optimized using any of the well known multidimensional optimization methods. However, this paper explores the use of a class of stochastic methods known as evolutionary programs (Michalewicz, 1992), of which genetic algorithms (De Jong, 1975; Fogel, 1994) are the most well-known subclass.

Evolutionary methods such as genetic algorithms begin with a population of starting points. These points randomly generate offsprings using a number of predefined rules or operators; these may be unary operators (mutations), which create new individuals by a random change in a single individual, or binary operators (crossovers) which create new individuals by combining two (sometimes more) "parents". The parent population may be discarded, partially retained, or completely retained depending on the implementation. A selection is then carried out to weed out the least optimal members. In that way, the population progresses towards the optimum. Because a population of members is always maintained, evolutionary methods tend to be little affected by random errors and are good at getting out of local optima.

Originally only two reproduction operators were used, mutation and crossover (De Jong, 1975). Mutation is the creation of an offspring from a parent by some random change in the parent. Crossover is the "splicing" of the bits that represent two parents in the computer’s memory into one or two new offsprings. The justification for using these

operators were a desire to imitate the reproduction of biological cells. The binary computer representations of the variables were considered to be somehow equivalent to the genes that encapsulate the hereditary properties of a living thing.

From an engineering point of view there is no reason to retain this rather crude imitation of nature, and new data representation methods and operators were invented to apply the principles of evolutionary optimization to real-world problems (Michalewicz, 1992). For best performance, each class of problem tends to requires its own set of operators. It is the aim of this paper to propose and test a set of operators specially designed for problems that can be cast in the form of a time-series representation.

Evolutionary optimization is ideally suited to cases where there are known constraints (minimum and maximum) to the independent variables. This applies to almost every engineering problems, since there are always some kind of technological or physical constraints. Classical methods, on the other hand, need to be modified to handle constraints.

In chemical engineering, evolutionary optimization has been applied by the author and others to system identification (Pham and Coulter, 1995; Moros, 1996); a model of a process is built and its numerical parameters are found by error minimization against experimental data. Evolutionary optimization has been widely applied to the evolution of neural networks models for use in control applications (e.g. Li & Haussler, 1996). It has also been used to derive optimal PID control parameters for a heat exchanger system (Jones & De Moura Oliverira, 1995).

THEORY

Let the problem be to maximize an objective function f(**u**), where $\mathbf{u} = (u_1, u_2 .. u_N)^T$ is the vector of control variables. For problems involving a single time- or distance-dependent variable, **u** will be a series approximation of the control variable:

$$\mathbf{u} = (u(t_1), u(t_2), ...u(t_P))^T \quad (1)$$

For problems involving N time- or distance-dependent variables, **u** can be formed by stacking (concatenating) the series on top of each other into a single vector of size NP. For example, if the time-dependent control variables are denoted by v(t) and w(t), then

$$\mathbf{u} = (v(t_1), .., v(t_p), w(t_1), .., w(t_p))^T \quad (1a)$$

Values at other times or distances can be found by interpolation (the precise interpolation method does not matter). P will be called the number of stages. For problems that do not fall into these categories, the ordering of the elements of the control vector is more or less arbitrary.

The general pseudocode for an evolutionary program is

Generate a random population of **u** vectors
Repeat
    Reproduction:
    Choose one (or two) existing members $\mathbf{u}^A$, $\mathbf{u}^B$ at random
    Choose a reproduction operator
    Generate a new member

    Elimination:
    Choose two members at random
    Eliminate less optimal member
Until some convergence criterion is reached

The type of evolution program shown in the above pseudocode is known as a steady-state population replacement, zero generation gap, random tournament selection. Steady-state means that as soon as a new member is generated, one is eliminated. Zero generation gap means that new and old members are equally likely to be selected for reproduction and elimination (other strategies may involve automatically eliminating the least fit members of the old population). Random tournament selection refers to the way candidates are chosen during the elimination step. Much work has been done on comparing the relative merits of various strategies, but this is outside the scope of this paper. The above strategy

has been found to work well for a number of test problems (Pham, 1995) and will be sufficient for this work.

This paper concentrates on the reproduction step. A number of operators were intuitively derived to generate a new vector from one or more existing vectors. As is customary in investigations of artificial intelligence methods, they were based on what a person might do when using a trials and errors approach. The reproduction operators are termed mutation, creep, shift, smoothing, swap, crossover, interpolation and extrapolation.

The first five are unary operators: they generate a new member from one existing member. *Mutation* generates an offspring from a parent member by causing a random, possibly large change in one element u_i:

$$u_i^{NEW} := u_i + \delta u_i^{max} (1 - \exp(-abs|G|)) \qquad (2)$$

where δu_i^{max} is the maximum possible change in u_i, i.e. the value of $u_i^{max}-u_i$ or $u_i-u_i^{min}$, depending on the direction of the change, and G is a randomly generated standard Gaussian variable (in most industrial processes, there will be legal, technological or physical constraints on the operating parameters; if there is no practical constraint, a variable transformation can be used so that the range of variable becomes finite). The exponent varies from 0 to 1, ensuring that the boundary value is not exceeded. The expression also ensures that small changes are more likely than large ones.

Creep causes a small change in all of the u-values:

$$u_i^{NEW} := u_i + A(u_i^{max}-u_i^{min})G, \quad i=1 \text{ to } n \qquad (3)$$

where A is a coefficient (chosen to be 0.02 in this work) and the random Gaussian variable G is generated independently for each variable i. If u_i^{NEW} exceeds a boundary then it is brought back inside the boundary by repeatedly halving the magnitude of the mutation for as long as necessary.

The *shift* operator will first be described in terms of a single time-dependent control variable $\mathbf{u} = (u_1, .., u_P)$. First, an integer k is randomly chosen in the range 1 to P. A random choice is made to shift to the right or to the left. For a right shift

$$u_i^{NEW} = u_i \quad \text{for } i \leq k$$
$$u_i^{NEW} = u_{i-1} \quad \text{for } i > k \tag{4}$$

and for a left shift

$$u_i^{NEW} = u_{i+1} \quad \text{for } i < k$$
$$u_i^{NEW} = u_i \quad \text{for } i \geq k \tag{4a}$$

If **u** has been formed by concatenating N time-dependent control variables, the shift operator is applied to one of these only, i.e. to a random subvector $(u_{KP+1}, .., u_{KP+P})$, $0 \leq K < N$.

Smoothing consists of performing a rolling average over a random section $i=i_{min}$ to i_{max} of **u**:

$$u_i^{NEW} = 0.67u_i + 0.33u_{i+1}, \quad i=i_{min}$$
$$u_i^{NEW} = 0.25u_{i-1} + 0.5u_i + 0.25u_{i+1} \text{ for } i_{min}<i<i_{max} \tag{5}$$
$$u_i^{NEW} = 0.67u_i + 0.33u_{i-1} \quad \text{for } i=i_{max}$$

Again, if **u** has been formed by concatenating N time-dependent control variables, the operator is confined to one of these chosen at random.

Swapping consist of interchanging the values of u_i and u_{i+1} where i is randomly chosen. If **u** has been formed by concatenating N time-dependent control variables, u_i and u_j must denote values of the same control variable at adjacent times.

We now come to the binary operators, which generate an offspring $\mathbf{u}^{NEW}$ by combining features of two existing members $\mathbf{u}^A$, $\mathbf{u}^B$. The *crossover* operator combines one section of one parent with another section from a second parent:

$$u_i^{NEW} = u_i^A, \quad i \leq k$$
$$u_i^{NEW} = u_i^B, \quad i > k \tag{6}$$

where k is a random integer. The *two-point crossover* is similar but two cuts were made in each parent rather than one. The *interpolation* operator averages two parents, with a bias towards the better parent:

$$u_i^{NEW} = (1-\alpha)\, u_i^A + u_i^B, \quad i=1 \text{ to } NP \tag{7}$$

where α is a positive number (chosen to be 0.3 in this work) and the superscript A denotes the better (more optimal) parent. The *extrapolation* operator extrapolates past the better parent:

$$u_i^{NEW} = (1+ß)\, u_i^A - ßu_i^B \tag{8}$$

where ß is a positive number (chosen to be 0.4 in this work).

Obviously there are a large number of parameters we can vary to improve the performance of the optimization method: the size of the population, the relative frequency of each operator, the parameters of the mutation, creep, smooth, interpolation and extrapolation operators. The optimal values of these parameters can be found by trial and error or by a meta-optimization (Grefenstette, 1986) but the process is time consuming and the results are problem-dependent. Alternatively, a competitive evolution strategy can be used (Pham, 1995) in which several populations, each with different parameter tunings, fight to compete for computer time. A meta-optimization was carried out for some of the test functions and it was found that the efficiency of the algorithm was not very sensitive to its parameters or to the problem, except for the population size. The values used in this paper were chosen as a result of compromise. The probabilities of the various operators were: mutation, 10%; smoothing, 15%; swap, 10%, creep, 15%; shift, 15%; crossover, 10%; two-point crossover, 10%; extrapolation, 15%; and interpolation, 0%. The population size was chosen to be 4, although the effect of other population sizes (2 and 8) were also investigated (more on this later).

Guarding against stalling

A common problem with all search methods is stalling which occurs at a local peak. To guard against such occurrence, Starkweather et al (1990) suggested using an adaptive mutation rate. In the present method this idea was implemented as follows (Pham, 1995). At the start of a reproduction cycle, two members $\mathbf{u}^A$, $\mathbf{u}^B$ of the population are chosen as

potential parents (even though one may subsequently choose a unary reproductive operator). The relative distance δ between these parents is calculated according to

$$d^2 = \sum_{i=1}^{NP} \left(\frac{u_i^A - u_i^B}{u_i^{max} - u_i^{min}} \right)^2 \tag{9}$$

The probability of selecting the mutation operator is then increased by an amount $e^{-d/\delta}$ where d is a constant (chosen to be $10^{-5}/\ln(2)$ in this work). The probabilities of the other operators are scaled back accordingly.

The rationale for this is that when stalling occurs, the population will crowd together near a local peak. The more severe the stalling, the smaller the distance d becomes, and the exponential term $e^{-d/\delta}$ will increase the likelihood of a mutation that will enable the search to escape from the local peak. At the limit, d=0 and mutation is certain to occur.

TEST PROBLEMS

Problem 1: Control of a plug flow reactor (Edgar & Himmelblau, 1989, p.369) The following reactions take place in a tubular reactor:

$$A \rightarrow B \rightarrow C$$

Introducing the Arrhenius relationship, the concentrations x_1 and x_2 of A and B respectively are given by

$$\frac{dx_1}{dt} = -k_1 x_1 e^{-E_1/RT}$$
$$\frac{dx_2}{dt} = k_1 x_1 e^{-E_1/RT} - k_2 x_2 e^{-E_2/RT} \tag{10}$$
$$50 \le T \le 100$$

The problem is to maximize the concentration of component B in the product stream by manipulating the temperature profile T(x). The residence time is kept at eight minutes, and the other parameters are: E_1 = 75 kJ/mol, E_2 = 125 kJ/mol, R = 8.31 J/molK, x_1^0 =

0.7 mol/l, $x_2^0 = 0$. Unfortunately the authors neglected to give the values of the pre-exponential factors k_1 and k_2, consequently their results cannot be directly compared with those of this paper. The following values were arbitrarily chosen: $k_1 = 1.0 \times 10^8$ min^{-1}, $k_2 = 4.0 \times 10^{15}$ min^{-1}.

Edgar and Himmelblau applied a classical optimization method (quasi-Newton) to the temperature function for three cases: a stepwise function with five steps, one with ten steps, and a third order polynomial. All the functions start with a high temperature near the inlet to increase the rate of reaction 1, decreasing to a low outlet temperature to minimize the rate of reaction 2. In this paper a 10-step function will be used, i.e. **u** = $(T(t_1), ..T(t_{10}))$. The optimal temperature profile, shown in Figure 1, yields an objective function value of 0.17796. Figure 2 shows the performance of the method for three different runs.

Problem 2: Control of a CSTR (Lapidus & Luus, 1967; Bojkov & Luus, 1993). This system is described by a set of seven O.D.E.s and is controlled by a set of four parameters (N = 4), u_1 to u_4, each of which vary with time over the interval $u_1 = 0$ to 20, $u_2 = 0$ to 6, $u_3 = 0$ to 4, $u_4 = 0$ to 20.

The maximum of the objective function, given by Bojkov & Luus, is 339.10 for 20 time intervals (P = 20). Figure 3 shows the convergence pattern for two runs. It may be noticed that a temporary "stalling" occurred in one of the runs between about 400 and 600 function evaluations; this stalling is due to a local minimum or near-stationary region, where small changes in the control variables have only very small effects on the objective function. Stalling is not unusual in this method and may cause some runs to be markedly slower than others; however, the random nature of the method ensures that it always escapes from the stall.

Problem 3: Control of a series of three CSTRs (Aris, 1961; Fan and Wang, 1964) This system is described by difference equations expressing the concentrations of three components:

$$x_{1,k} = \frac{x_{1,k-1}}{1 + u_{2.k}(1+u_{1,k})}$$

$$x_{2,k} = \frac{x_{2,k-1} + x_{1,k}u_{2,k}}{1 + 0.01u_{2,k}(1+u_{1,k})/u_{1,k}} \qquad (11)$$

$$x_{3,k} = x_{3,k-1} + 0.01u_{2,k}x_{2,k}$$

The control variables u_1 and u_2 (N = 2) are bounded by

$$0 \le u_{1,k} \le 10^4 e^{-3000/Tmax}$$
$$0 \le u_{2,k} \le 2100 \qquad (12)$$

for k=1 to 3 (P = 3), and T_{max} = 394. The problem is to determine the values of $u_{1,k}$ and $u_{2,k}$ that would maximize the concentration of component 3 from the last stage, $x_{3,3}$. Direct search techniques gave a maximum of 0.54897 (Rangaiah, 1985).

Figure 4 shows the convergence curve for a few runs, together with one for the randomized iterative dynamic programming method of Bojkov and Luus (1992). The latter consists of a series of partial integration of the governing differential equations, each integration covering from one (time) step to the total number of (time) steps; therefore on average one integration in that method is equivalent to half an integration in the present method. It can be seen from the graph that the present method generally performs better than the iterative dynamic programming method. Rangaiah (1985) tested the method of multipliers (Powell, 1969; Hestenes, 1969) with the BFGS method, the complex method of Box (1965) and the adaptive random search method, and reported that none of these consistently reach the global minimum because of the existence of a local minimum.

Problem 4: Control of a series of ten CSTRs

This problem is identical to the previous one except for the number of tanks (P = 10). Again there is no problem with convergence. Figure 5 shows a few typical convergence curves. The method is much quicker than randomized iterative dynamic programming,

which requires about 8000 function evaluations to reach a fitness value of 0.6296 (Bojkov & Luus, 1992).

Problem 5: Nagurka's problem (1991)

A system is described by

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{u}(t)$$

$$\boldsymbol{x}(0) = [1\ 2\ 3\ ..\ N]^T$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & . & . & . & 0 \\ 0 & 0 & 1 & . & . & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & . & . & . & N & (-1)^{N+1} \end{bmatrix} \tag{13}$$

with N = 6, and the problem is to find **u**(t) in the time interval $0 \le t \le 1$ that minimizes

$$10\ \boldsymbol{x}^T(1)\boldsymbol{x}(1) + \int_0^1 (\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{u})\, dt \tag{14}$$

u is assumed to vary in 10 stages (P = 10) and subject to the constraints

$$-15 \le u_i \le 0 \tag{15}$$

The optimum for this problem is 153.98 (Bojkov & Luus, 1992). Figure 6 shows the convergence curves for a few runs, together with that for the above authors' iterative dynamic programming method.

Problem 6: Inverse heat transfer

A sphere with radius 0.5 (in consistent units), volumetric specific heat 4.0 x 10^{-6} and thermal conductivity 1.0 is cooled from a uniform initial temperature 0 in an environment at variable temperature $T_e(t)$ and a constant heat transfer coefficient of 50. The centre temperature is recorded and the problem is to find the environment temperature function $T_e(t)$. This is known as an inverse heat transfer problem (Beck et al, 1985) and there are

alternative methods for solving such problems, depending on the case. However, the method of this paper lends itself easily to such problems.

The centre temperature "data" was numerically generated by using a finite difference model where the cylinder is divided into eight shells of equal thickness and the temperature calculated at intervals of $\delta t = 20$, using the Crank-Nicolson central difference time stepping method. The environmental temperature used was 0 at the start, decreasing linearly with time to -100 at $t = 10^4$, then staying constant at that value.

To find the environmental temperature, the same finite difference model was used but with a much larger time step of 1000. Apart from reducing calculation times, such a large time step would be expected to introduce numerical linearization errors which will test the ability of the evolutionary method to deal with such errors. The objective is to find $T_e(t)$ that would minimize the sum of squared errors

$$SSE = \sum_{j=1}^{20} T_c^{calc}(t_j) - T_c^{actual}(t_j) \tag{16}$$

where T_c is the centre temperature and $t_j = 1000j$. The environmental temperature was subject to the constraint

$$-100 \le T_e \le 0 \tag{17}$$

The method of this paper easily solved the inverse heat transfer problem (Figure 7). The slight difference between the "calculated" and "actual" environment temperature is attributable to the large time interval used for the calculated results. Some typical convergence curves are shown in Figure 8; it can be seen that the calculated centre temperatures agree with "data" to within a few hundredths of a unit. That the method worked well is not surprising since the by using the finite difference method the problem has been reduced to a system of ordinary differential equations.

Problem 7: Schwefel's function

Although not belonging to the same category as the others, this and the next problems are included to illustrate how seemingly "difficult" problems can be easily solved by

evolutionary methods if the "right" reproduction operators are used. Schwefel's (1977) function, quoted by Mühlenbein et al (1991) is described by

$$f(\mathbf{u}) = \sum_{i=1}^{30} u_i - u_i \sin(\sqrt{|u_i|}), \quad -500 \le u_i \le 500 \tag{18}$$

Here **u** = (u_1, u_2 ... u_{30}). We take N = 1, P = 30. This is a severely multimodal function in 30 dimensions with the global maximum situated near one corner of the region (u_i = 420.9687 for i=1 to 30) and many secondary peaks near other corners that tend to trap the search procedure. Mühlenbein et al (1991), using an optimized parallel genetic algorithm on four parallel processors, found the global optimum in 37573 function evaluations on average.

The present evolutionary algorithm easily solved this problem (Figure 9) within about 1000 function evaluations, because its "shift" operator enables it to jump from one corner to another, quickly converging to the global maximum.

Problem 8: Grierwangk's problem

Grierwangk's (1981) function is considered to be one of the most difficult test functions to optimize in the literature (Mühlenbein et al, 1991):

$$f_8(\mathbf{u}) = 0.00025 \sum_{i=1}^{10} u_i^2 - \prod_{i=1}^{10} \cos \frac{u_i}{\sqrt{i}} + 1, \quad -600 \le u_i \le 600 \tag{19}$$

Mühlenbein et al's (1991) parallel genetic algorithm could not find the global optimum. Snyman and Fatti (1987) needed 23399 function evaluations. Rinnooy Kan and Timmer (1987) found the global minimum in only six of ten test runs, though they stopped at 17000 function evaluations. The present method did not work well with small population sizes, but with a population size of 12 or more was 90% successful and, for the runs that converged, found the global optimum after 5750 function evaluations on average. Thus, it performs far better than any other method that the author knows of. Figure 10 shows three runs. One of these is many times slower to converge than the others. This pattern did not occur in the other problems: it appears that the reputation for difficulty of

Grierwangk's problem is justified. The competitive evolution method suggested by Pham (1995) can be used to guard against occasionally poor performance.

Effect of population size

Table 1 shows the effect of population size on performance as measured by the average number of function evaluations to get to within 1% of the global optimum. It is unlikely that a closer approach would be required in a practical situation; should that be the case, the GA solution can be "polished up" with a conventional gradient search which can be expected to be much faster in the immediate neighbourhood of the global optimum.

For most functions the method performs best with a population size of 2; however, with problems 3 and 8 the method fails with this population size. Moreover it is difficult to predict the best population size from the nature of the problem. Problems 3 and 4 are formally similar yet problem 4 is best solved with a population size of 2 while problem 3 fails for this size. A population size of 4 works reasonably well for all the time-series problems (problems 1 to 6) and for Schwefel's problem. Grierwank's problem, however, requires a large population size.

Effect of adaptive mutation term

Removing the adaptive mutation term $e^{-d/\delta}$ did not significantly affect the performance of the method: the average number of fitness function evaluations per run were almost unchanged from those of Table 1 (maximum change about 15%). It is thought that this is due to the mutation frequency employed is already large (10%), so that the additional mutation contributed by the adaptive term has little effect.

Operators' contributions

The contribution of each reproductive operator to the progress towards the optimum can be measured by the number of times the application of that operator leads to an improvement, expressed as a percentage of the total number of improvements. However,

with evolutionary algorithms, because "bad" tries are not immediately eliminated, an operator may also cause a *delayed* improvement: although the offspring is not better than the present optimum, the offspring's offspring (and so on) may bring about an improvement. In fact this is one of the main strengths of evolutionary algorithms over other optimization methods.

This paper will only consider one generation's delay. Thus an operator causes a *direct* improvement when it produces an offspring that is better than the present best, and a *delayed* improvement when its offspring survives and produces an offspring that is better than the (new) present best.

Figure 11 shows the percentage of all direct (black bars) and delayed (white bars) improvements brought about by the various operators. It can be seen that all the operators play an important role in one or more problems, although the crossover, two-point crossover and swap operators are less prominent than the others. The creep and smoothing operators are expected to contribute to gradual improvements in the classical manner, and particularly the final "polishing up", while the other operators provide large jumps that enable the search to escape from local minima.

CONCLUSIONS

The evolutionary method of this paper has been tested over a range of problems, both academic and practical, and found to perform well in all of them. Its wide choice of operators is believed to give it good all-round performance for problems of this type, and it is able to efficiently solve test problems that are reputed to be difficult.

NOTATION

A	small constant
d	relative distance between two parents $\mathbf{x}^A$,$\mathbf{x}^B$
G	randomly generated standard Gaussian variable
i,j,k	indices

k_1,k_2 pre-exponential vectors

N number of control variables

P number of stages for each control variable.

x state vector

u control vector

u_i^{max} maximum possible value of u_i

u_i^{min} minimum possible value of u_i

α interpolation parameter

ß extrapolation parameter

δ parameter for calculating probability of mutation

Superscript

A more optimal parent

B less optimal parent

NEW newly generated value of **u**

REFERENCES

Aris, R., *The Optimal Design of Chemical Reactors*, pp.85-95. Academic Press, New York (1961).

Beck, J.V., Blackwell, B. and St Clair C.R., *Inverse Heat Conduction Problems*. Wiley, New York (1985).

Bojkov, B. and Luus, R., Evaluation of parameters used in iterative dynamic programming. *Can. J. Chem. Eng.* **71**, 451-9 (1993).

Box, M.J., A new method of constrained optimizationi and a comparison with other methods. *Comp. J.* **8**, 42 (1965).

De Jong, K.A., *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, Doctoral thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor (1975).

Edgar, T.F. and Himmelblau D.M., *Optimization of Chemical Processes*. McGraw-Hill, New York, p.369-371 (1989).

Fan, L.T. and Wang, C.S., *The Discrete Maximum Principle*, pp.104-108. Wiley, New York (1964).

Fogel, D.B., An introduction to simulated evolutionary optimization. *IEEE Trans. on Neural Networks* **5**, 3-14 (1994).

Grierwangk, A.O., Generalized descent for global optimization. *J. Opt. Theory Appl.* **34**, 11-39 (1981).

Grefenstette, J.J., Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics* **SMC-16**, 122-8 (1986).

Hestenes, M.R., Multiplier and gradient mehods. *J. Opt. Theory Appl.* **4**:303 (1969).

Jones, A.H. & P.B. De Moura Oliveria, Genetic auto-tuning PID controllers. *Proc. First IEE/IEEE Internat. Conference GA's in Engineering Systems: Innovations and Applications*, Sheffield, UK, pp.141-145 (1995).

Lapidus, L. and Luus, R., *Optimal Control of Engineering Processes*. Blaisdell Pub., Waltham, MA, pp.155-229 (1967).

Li, Y. and Haubler, A. Artificial evolution of neural networks and its application to feedback control. *Artificial Intelligence in Engineering* **10**, 143-152 (1996).

Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, N.Y. (1992).

Moros, R., Kalies, H., Rex, H.G. and Schaffarczyk, S., A genetic algorithm for generating initial parameter estimations for kinetic models of catalytic processes. *Computers & Chemical Engineering* **20**, 1257-1270 (1997).

Mühlenbein, H., Schomisch, M. and Born, J., The parallel genetic algorithm as function optimizer. *Parallel Computing* **17**, 619-632 (1991).

Nagurka, M., Wang, S. and Yen, V., Solving linear quadratic optimal control problems by Chebychev-based state parametrization. *Proc. 1991 American Control Conference*, Boston, MA. Am. Automatic Control Council, IEEE, Piscataway, NJ, pp.104-109.

Pham, Q.T. (1995) Competitive evolution: a natural approach to operator selection. In: *Progress in Evolutionary Computation*, Lecture Notes in Artificial Intelligence, Vol. 956, p.49-60. X. Yao (ed.), Springer-Verlag, Heidelberg (1995).

Pham, Q.T. and Coulter, S. Modelling the chilling of pig carcasses using an evolutionary method. *Proc. Int. Congress of Refrig.* Vol. **3a** pp.676-683 (1995).

Powell, M.J.D., A method for nonlinear constraints in minimization problems. In *Optimization*, ed. R. Fletcher, Academic Press, N.Y., p.283 (1969).

Rangaiah, G.P., Studies in Constrained Optimization of chemical process problems. *Comput. Chem. Eng.* **9**, 395-404 (1985).

Rinnooy Kan, A.H.G. and Timmer G.T. Stochastic global optimization methods, I and II. Mathematical Programming v39 p 27-56, 57-78 (1987).

Schwefel, H.P., Numerisch Optimierung von Computer-Modellen mittels der Evolutionsstrategie, *Interdisciplinary System Research*, Vol. 26. Birkhauser, Basel (1977).

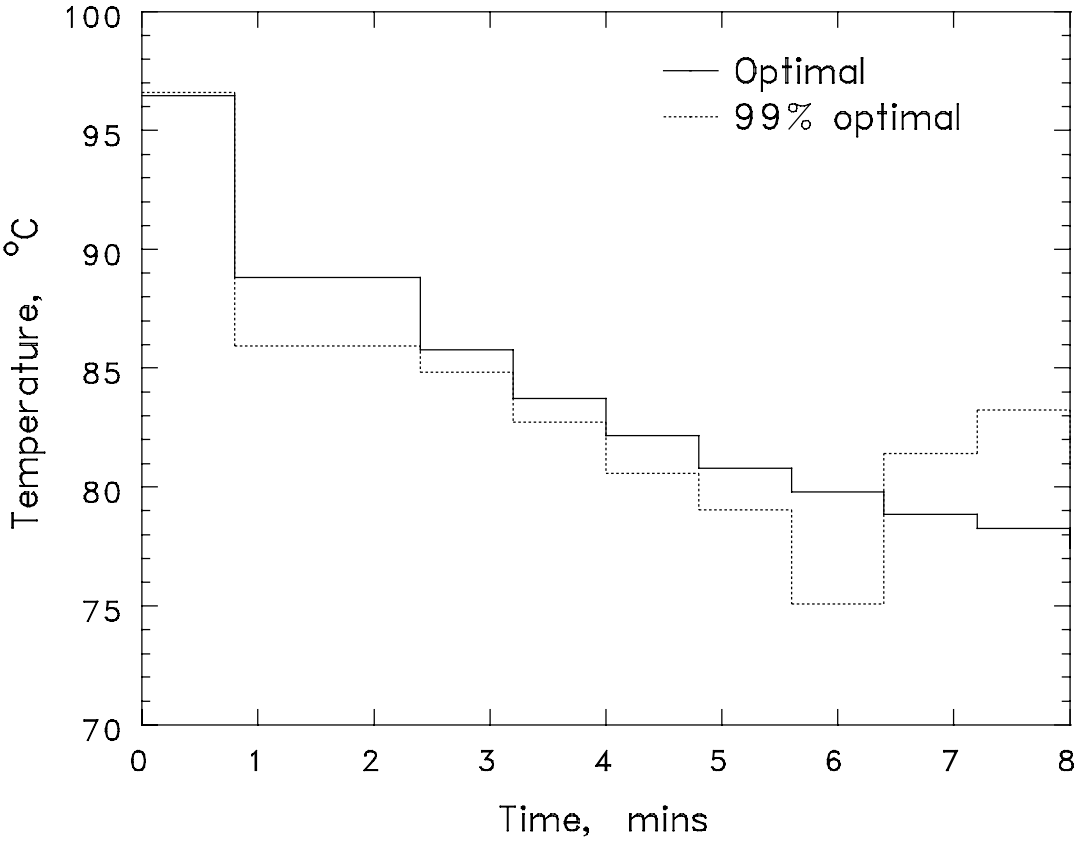
Snyman, J.A. and Fatti, L.P. A multistart global minimization algorithm with dynamic search trajectories. *JOTA* **54**, 121-141 (1987).

Starkweather,T., Whitley, L.D. and Mathias, K.E., Optimization Using Distributed Genetic Algorithms, PPSN 1, pp.176-185. H.P. Schwefel and R. Manner (ed.). Springer-Verlag, Heidelberg (1990).

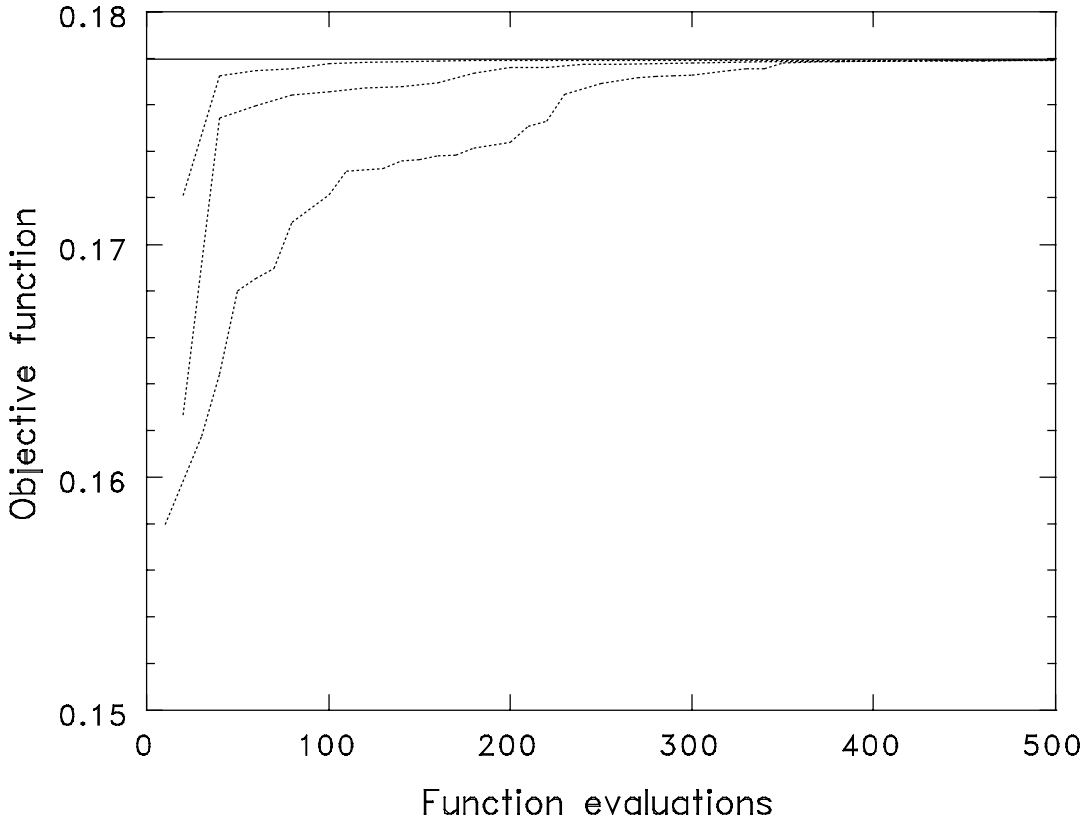
Table 1: Average number of function evaluations to get within 1% of optimum. For Grierwangk's function, percentage failure to converge within 30,000 function evaluations.

Problem	No. of runs	N = 2	N = 4	N = 8
1: Control of PFR	50	103	113	135
2: Control of a CSTR	20	755	845	968
3: Control of three CSTRs	50	>10000	504	643
4: Control of ten CSTRs	50	1503	1566	2129
5: Nagurka	20	2198	2696	3233
6: Inverse heat transfer	50	502	587	746
7: Schwefel	50	750	1145	1585
8: Grierwangk	50	76% failure	30% failure	10% failure

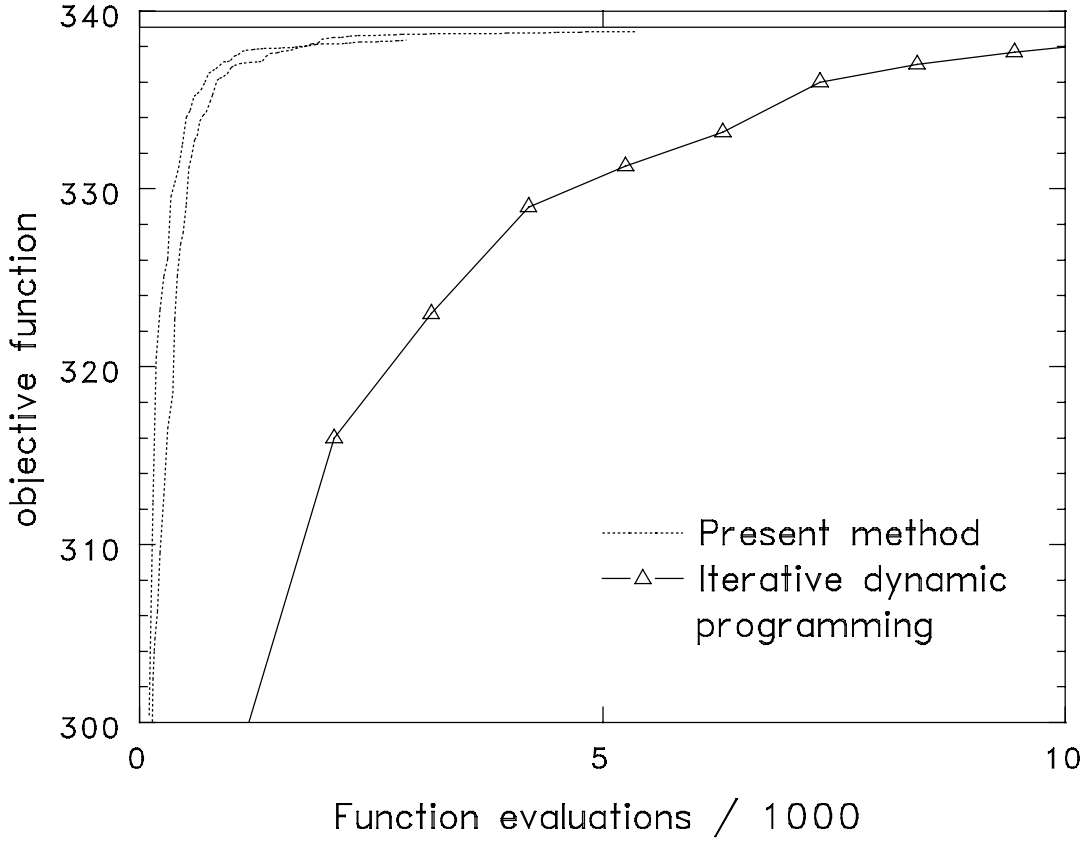
LIST OF FIGURES


Optimal
99% optimal
Temperature, °C
Time, mins
100
95
90
85
80
75
70
0
1
2
3
4
5
6
7
8

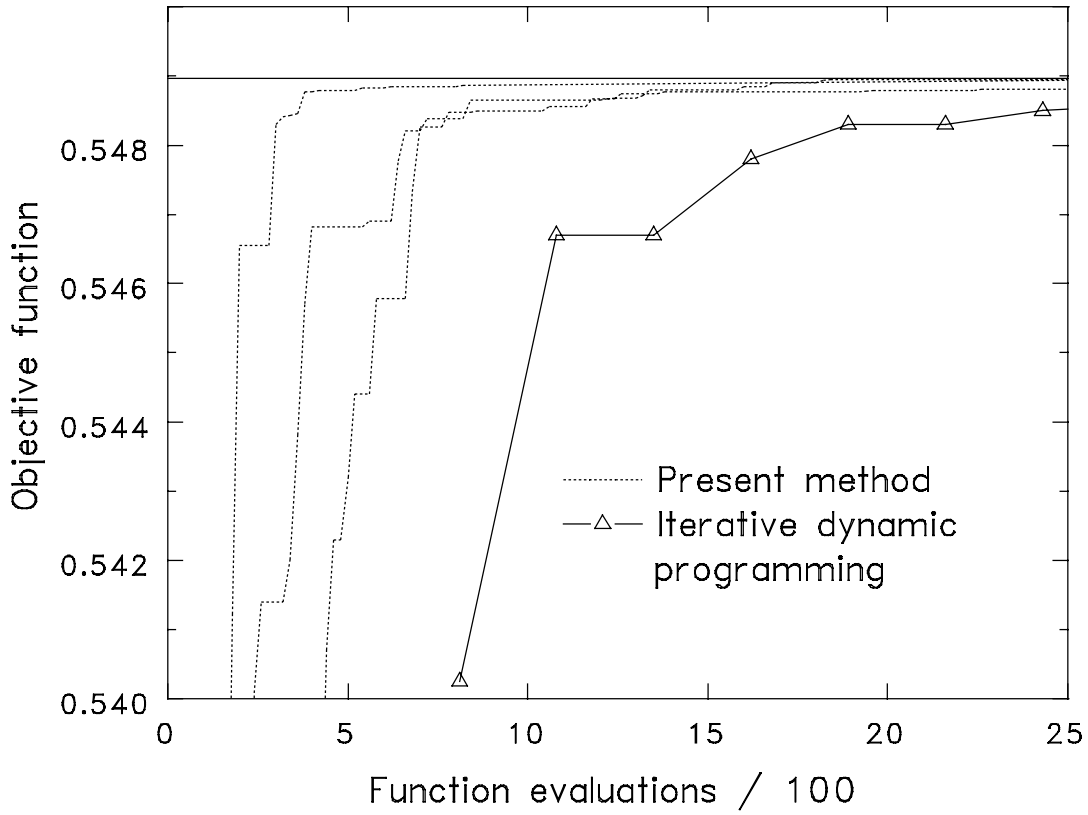
Pham, Fig.1


0.18
0.17
0.16
0.15
Objective function
0
100
200
300
400
500
Function evaluations

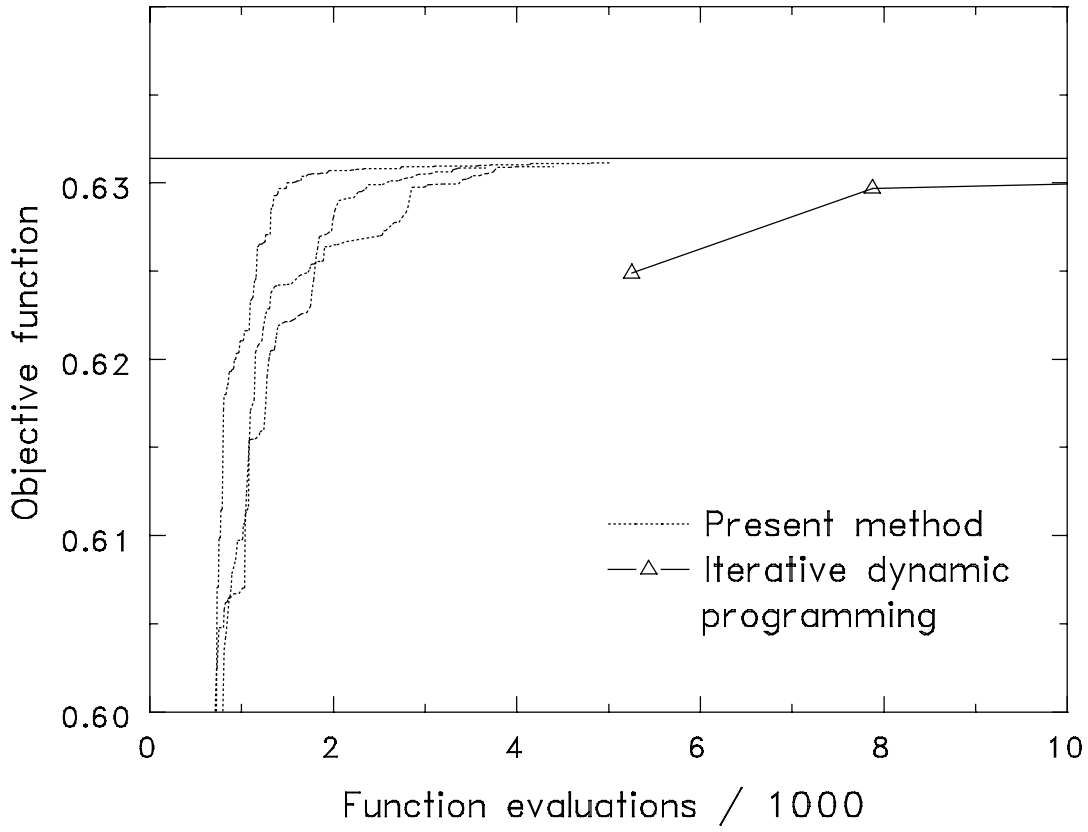
Pham, Fig.2


340
330
320
310
300
objective function
0
5
10
Function evaluations / 1000
Present method
Iterative dynamic
programming

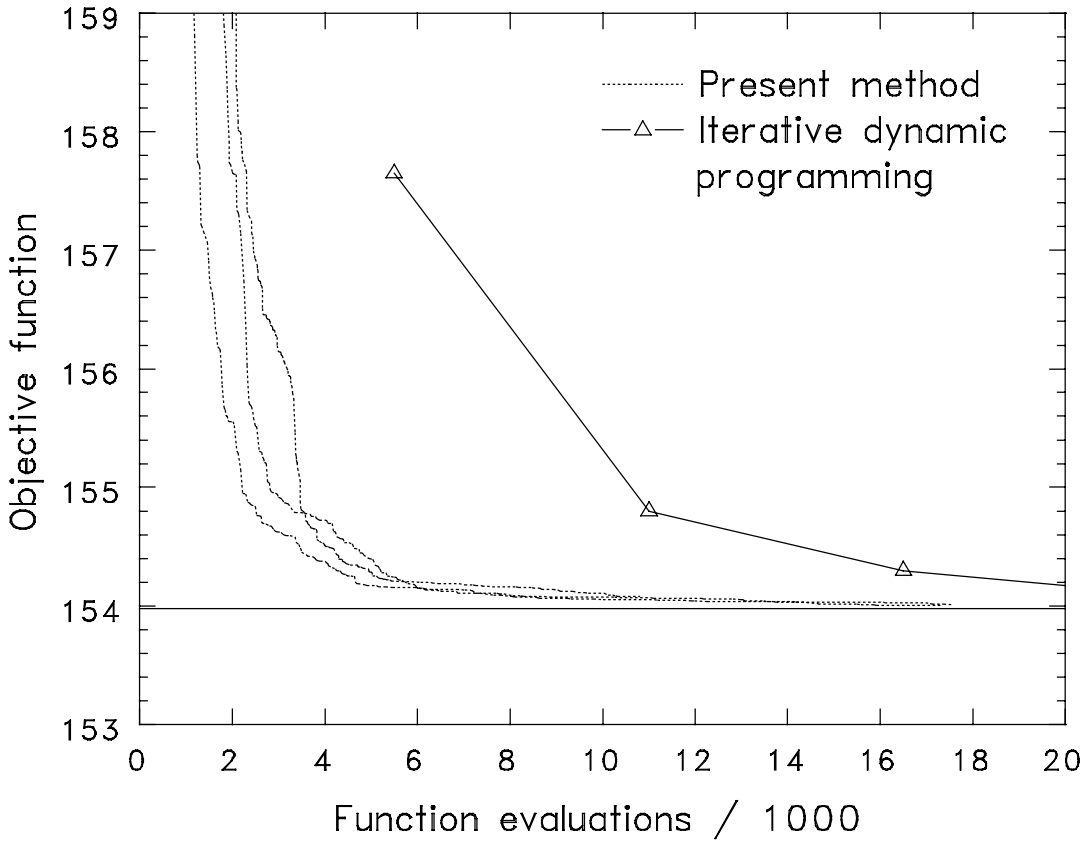
Pham, Fig.3


Objective function
0.548
0.546
0.544
0.542
0.540
0
5
10
15
20
25
Function evaluations / 100
Present method
Iterative dynamic programming

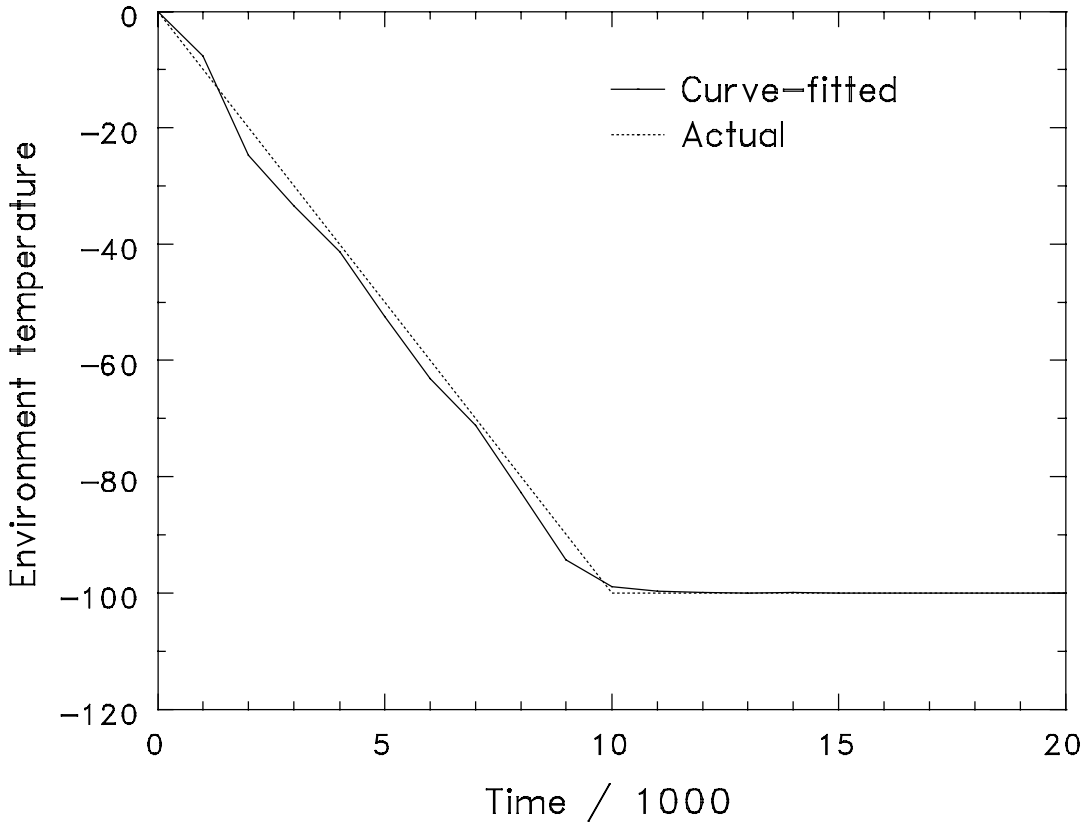
Pham, Fig.4


Objective function
0.63
0.62
0.61
0.60
0
2
4
6
8
10
Function evaluations / 1000
Present method
Iterative dynamic programming

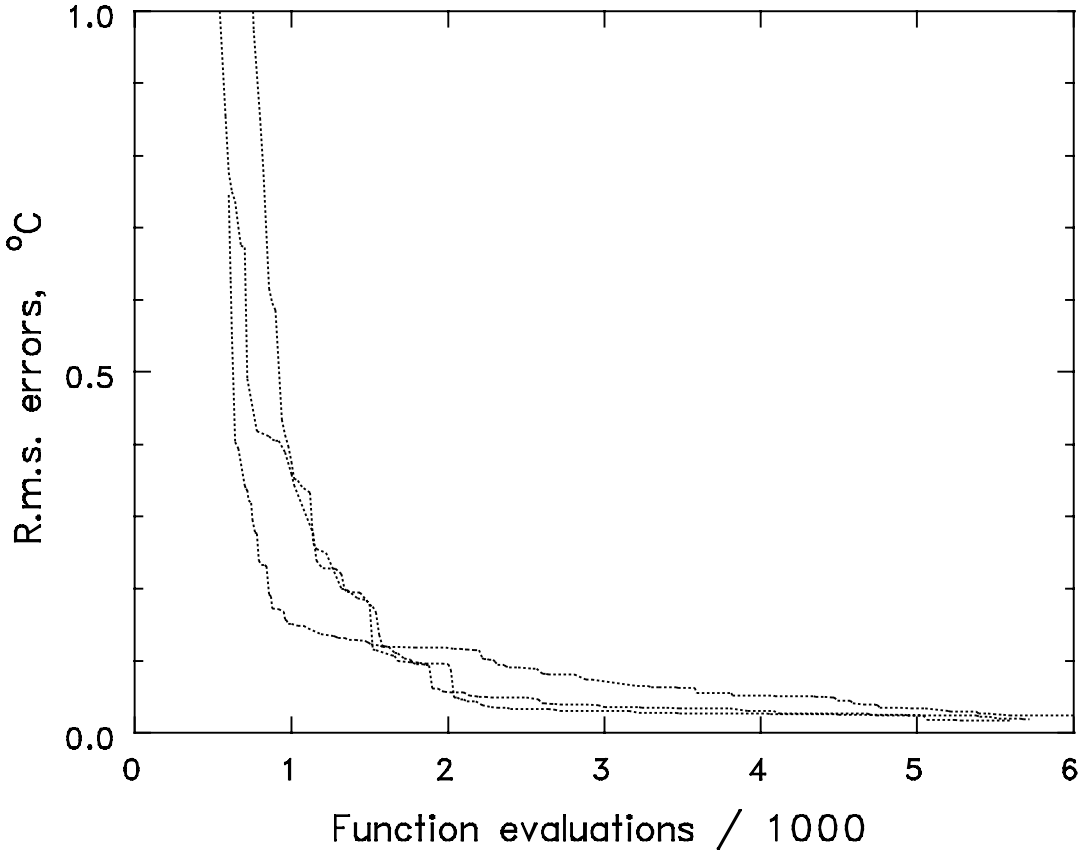
Pham, Fig.5


159
158
157
156
155
154
153
Objective function
0
2
4
6
8
10
12
14
16
18
20
Function evaluations / 1000
Present method
Iterative dynamic
programming

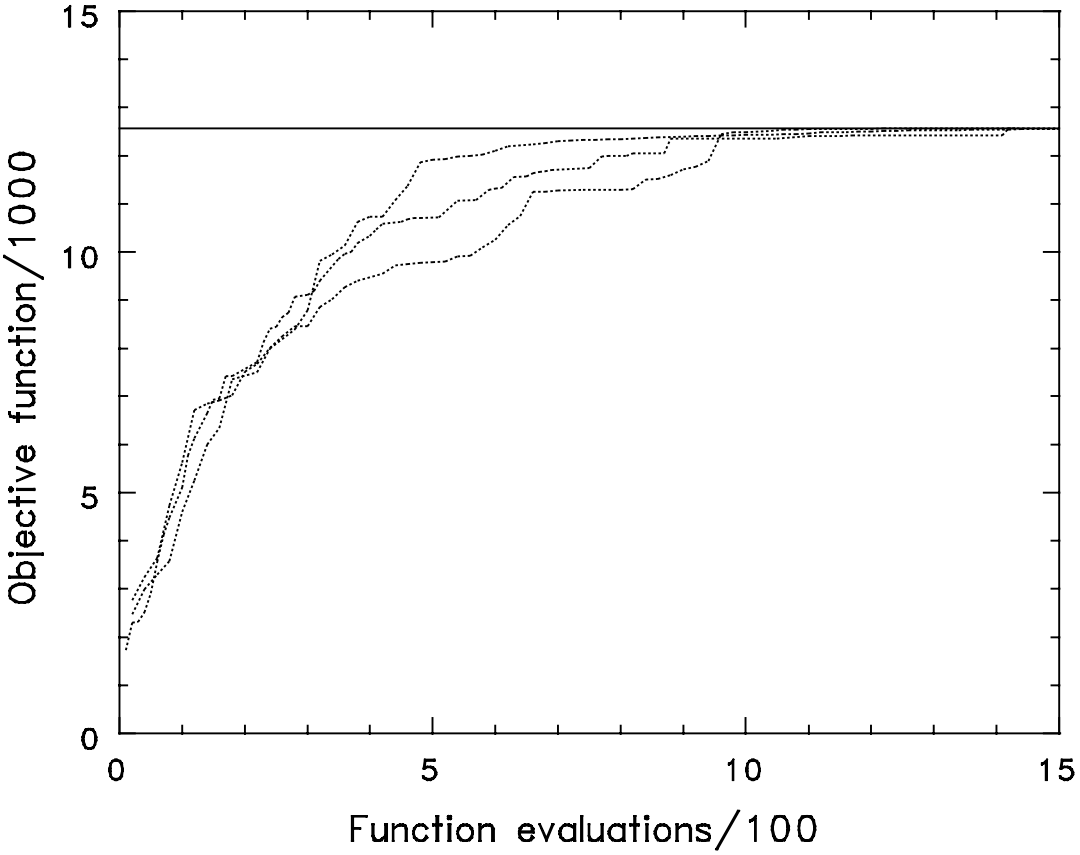
Pham, Fig.6


Curve-fitted
Actual
Environment temperature
0
-20
-40
-60
-80
-100
-120
0
5
10
15
20
Time / 1000

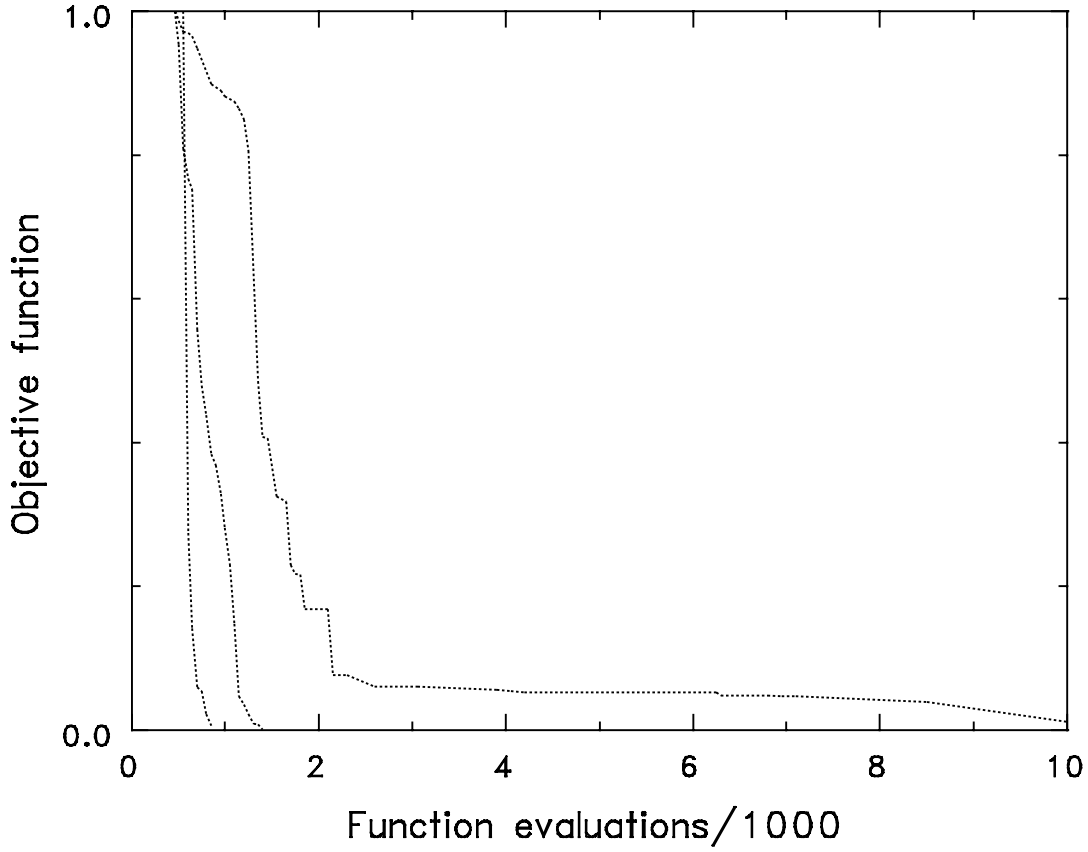
Pham, Fig.7



Pham, Fig.8


15
10
5
0
Objective function/1000
0
5
10
15
Function evaluations/100

Pham, Fig.9


1.0
0.0
Objective function
0
2
4
6
8
10
Function evaluations/1000

Pham, Fig.10