Original Article

Institution of MECHANICAL ENGINEERS

Journal of SYSTEMS AND CONTROL ENGINEERING

Proc IMechE Part I:
J Systems and Control Engineering
226(7) 956–971

DOI: 10.1177/0959651812443462
pii.sagepub.com
SAGE

Dynamic optimisation by a modified bees algorithm

Marco Castellani[1], Q Tuan Pham[2] and Duc T Pham[3,4]

Abstract

A modified bees algorithm was applied to dynamic optimisation problems in chemical engineering. A two-level factorial experiment was used to tune the settings of the population parameters, based on the premise that it is most important to avoid those configurations that cause the worst performances than to look for those that reach the best performance. Tested on eight well known benchmark problems, the tuned algorithm outperformed the standard bees algorithm and other two well known optimisation methods. The performance of the proposed algorithm was also competitive with that of the state-of-the-art in the literature, and the solutions produced were very close to the known optima of the benchmarks. The results demonstrate the efficacy of the modified bees algorithm as a tool for the solution of dynamic optimisation problems. The results also proved the effectiveness of the proposed statistical parameter tuning algorithm, and indicated its competitiveness as an alternative to the standard complex and subjective trial-and-error methods.

Keywords

Evolutionary optimisation, tuning, dynamic optimisation, factorial experiment, parameter setting, swarm intelligence, bees algorithm

Date received: 2 October 2011; accepted: 1 March 2012

Introduction

Dynamic optimisation, also known as optimal control, open loop optimal control or trajectory optimisation, is a class of problems where a control profile is adjusted to vary with time or some other independent variable so an optimal output is obtained, while possibly complying with certain constraints. A well known popular example is the moon landing game, where a lunar landing module has to be decelerated to a soft landing while minimising the fuel consumption. Other examples of dynamic optimisation problems range over a wide variety of applications: economics, aeronautics, medicine (drug delivery), reaction engineering, food processing, etc. In dynamic optimisation, the system to be optimised is described by a set of differential equations (such as those governing the speed of a chemical or biological reaction) whose coefficients depend on a number of control variables (such as temperature, pressure, catalyst concentration, heat or power input, degree of steering, etc.). The engineer has to manipulate these control variables over time in order to optimise given measures of product quality, yield, cost and other criteria.

Methods for solving dynamic optimisation problems can be divided into indirect and direct methods. Indirect methods are based on the calculus of variations. From Pontryagin et al.'s Minimum Principle,[1] a first-order optimality condition is derived, resulting in a two-point boundary value problem which is then solved to obtain the optimum.[2] Indirect methods are among the most popular approaches, although the resulting boundary value problems are usually difficult to solve. A review in the field of chemical reaction engineering is given by Srinivasan et al.[3]

Direct methods attempt to carry out the optimisation directly on the control variable profiles. Direct methods can be further classified according to the optimisation algorithm used to solve the problem: dynamic

[1]Department of Biology, University of Bergen, Norway
[2]School of Chemical Engineering, University of New South Wales, Australia
[3]School of Mechanical Engineering, University of Birmingham, UK
[4]Department of Information Systems, King Saud University, Saudi Arabia (visiting)

Corresponding author:
Marco Castellani, Theoretical Ecology Group, Department of Biology, University of Bergen, Postboks 7803, 5020 Bergen, Norway.
Email: Marco.Castellani@bio.uib.no

programming, deterministic nonlinear optimisation and stochastic optimisation.

In dynamic programming,[4] the process is modelled as a chain of transitions from one state to another. The feasible transitions from the initial state to the final state are systematically explored to determine the optimal chain of transitions. For continuous problems, the state space needs to be discretised into a grid. The quality of the optimum depends on how fine this grid is, but the size of the problem increases rapidly with the fineness of the grid. One way to reduce the number of computations is iterative dynamic programming, in which the discretisation of the state grid is coarse at first then gradually refined by concentrating on the region around the optimal found in the previous iteration.[5–10] For large numbers of differential equations, dynamic programming leads to very large dimensionality (number of paths to explore) and tends to be very slow.

Other direct methods start by a parameterisation of the control profiles (sequential or control vector parameterisation methods) or by discretising both the control and the state variables (simultaneous methods). In simultaneous methods, as reviewed by Biegler,[11] the control and state variables are discretised by collocation on finite elements in time. This results in large nonlinear programming problems that are usually solved by sequential quadratic programming. In sequential methods, the continuous control profiles are represented by approximate equations, such as polynomials, series of steps, piecewise linear profiles or spline curves, which can be described by a finite number of parameters. The value of the objective function depends on these parameters and the dynamic optimisation problem is reduced to an ordinary (generally nonlinear) optimisation problem, which can be solved by any numerical optimisation algorithm.[12] These optimisation algorithms can be classified into two types: deterministic and stochastic.

Deterministic methods such as sequential quadratic programming have been the most widely used in the classical optimisation literature. The search starts from a user-selected initial solution and progresses towards the optimum by making use of information on the gradient of the objective function. The gradient may be calculated directly or inferred from previous trials. Examples of dynamic optimisation being solved by direct deterministic methods can be found elsewhere.[13–18]

Stochastic methods, such as simulated annealing or genetic algorithms, are optimisation methods that include an element of randomness in the search. Published uses of stochastic methods for dynamic optimisation include controlled random search,[19,20] simulated annealing,[21] evolution strategies,[22–24] genetic algorithms,[25] the ant colony algorithm,[26,27] the iterative ant colony algorithm,[28] memetic algorithms[29] and the bees algorithm.[30] Q.T. Pham[31] introduced a technique called progressive step reduction (PSR) whereby a rough profile is first obtained, then progressively refined by repeatedly doubling the number of points.

Stochastic methods tend to be time consuming because they spend a large amount of time exploring the search space in a pseudo-random manner. On the other hand, they have the advantage over other optimisation methods in being able to escape from local optima and converge towards the global optimum, given enough time. Deterministic optimisation and other 'classical' methods, because they rely on gradient information, are not able to escape from a local optimum once they have converged to that point. This well known characteristic has been demonstrated by Rangaiah[13] in the context of dynamic optimisation. Dynamic programming is a special case: if the discretisation of state space is sufficiently fine, dynamic programming will be as good as an exhaustive search and the global optimum is guaranteed to be found, but the computing cost will be very great. Iterative dynamic programming reduces computing costs but the coarser grids used in the early iterations may cause the global optimum to be missed.

The present article tests the effectiveness of a recently introduced, modified version of the bees algorithm[32] as a tool for dynamic optimisation. In the context of the above classification it falls into the class of direct, stochastic dynamic optimisation methods. The outline of the article is as follows. First, the problem of dynamic system optimisation is formalised. Next, the standard bees algorithm is outlined and the modified version presented in detail, after which the PSR procedure is briefly motivated. A novel method to achieve a rough tuning of the parameters is then introduced. Thereafter the experimental setup is described, the results are reported and conclusions are drawn.

Dynamic optimisation

Dynamic control problems can be formulated as follows

$$\underset{\mathbf{u}(t)}{Maximize} \quad F(\mathbf{u}, \mathbf{y}) \tag{1}$$

subject to

$$\frac{dy}{dt} = G(t, \mathbf{y}(t), \mathbf{u}(t)), \qquad t \in [0, t_f] \tag{2}$$

$$\mathbf{y}(0) = \mathbf{y}_0 \tag{3}$$

$$\mathbf{u}^{min} \leqslant \mathbf{u}(t) \leqslant \mathbf{u}^{max} \tag{4}$$

where t is the independent variable (henceforth called 'time' although it may also be another coordinate), $\mathbf{y}$ is the state variable vector (size m), $\mathbf{u}$ is the control variable vector (size n) and t_f is the final time. The equations are almost always integrated numerically. The objective is to manipulate $\mathbf{u}(t)$ in order to optimise F. In practice, the time interval [0, t_f] is divided into a finite number of steps s (15–39 in this work) and each variable u_i is specified at each of these steps. The number of steps was chosen mainly for historical reasons, to enable present results to be compared with results in

previous work. The control variables may be assumed to be constant within each time interval, or to ramp between the end points. This article takes the second approach (ramping). Thus, each control variable u_i is represented by a vector of s elements, defining $s-1$ time intervals. The complete solution, or 'chromosome' in genetic algorithm terminology, is formed by juxtaposing the **u**-vectors to form a gene vector **x** of length $n \times s$, such that

$$\mathbf{x} = \{u_1(1), \ldots, u_1(s), u_2(1), \ldots, u_2(s), u_n(1), \ldots, u_n(s)\} \quad (5)$$

The eight objective functions[5–9,23,27,31–35] considered in the present work are listed in Table 1. Problems 2–8 are fully described by Rajesh et al.,[27] Table 1. Problem 1 was given in full by Q.T. Pham.[31] The eight benchmarks are described fully in Appendix 2.

Dynamic optimisation using the bees algorithm

Functions of the kind listed in Table 1 represent complex optimisation tasks. The search space is highly multimodal,[27] and analytical solution is generally not possible. On this kind of problem, global population-based metaheuristics such as evolutionary algorithms (EAs)[39] and swarm intelligence (SI) algorithms[40] are known to perform well. Indeed, as mentioned in the introduction, a number of successful optimisers were created using evolutionary strategies[22–24] and ant colony optimisation.[26,27]

The bees algorithm[41] is a population-based optimisation procedure that mimics the foraging behaviour of honey bees. Recently, D.T. Pham and Castellani[42] proved the ability of the bees algorithm to search effectively complex and multimodal continuous performance landscapes. Compared to different implementations of EAs[43] and particle swarm optimisation (PSO),[44] the bees algorithm yielded competitive results in terms of accuracy, learning speed and robustness. These results suggest that the bees algorithm may be used with success in the solution of dynamic optimisation problems.

This section outlines first the standard formulation of the bees algorithm and subsequently describes a recently proposed modification of the customary procedure.

The standard bees algorithm

The bees algorithm divides the search effort into the two tasks of exploration and exploitation. The former is concerned with the global sampling of the solution space in search of areas of high fitness (performance). The latter is concerned with the local search in the most promising regions. Figure 1 shows the flowchart of the bees algorithm.

At initialisation, N agents (*scout bees*) are randomly scattered across the environment (i.e. the search space). Each bee evaluates the fitness of the site (i.e. solution) where it landed. The algorithm enters then the main loop of the procedure. At each iteration, the *nb* scouts that located the solutions of highest fitness recruit a variable number of foragers to further explore their surroundings (local search phase). In bees algorithm terminology, the neighbourhood of a solution is called a 'flower patch', and corresponds to a hyperbox of sides $s^i = ngh \cdot (x^i_{min} - x^i_{max})$, $i = 1, \ldots, n$, centred on the scout location. x_{min} and x_{max} are respectively the lower and upper extremes of the interval of definition of variable x_i.

In a way that mirrors the waggle dance of biological bees,[45] each of the $ne < nb$ scouts that located the top (*elite*) solutions recruit *nre forager bees*, and each of the remaining $ne - nb$ scouts recruit $nrb < nre$ foragers. Each forager is randomly placed within the flower patch located by the scout. If one of the foragers lands

Table 1. Test objective functions

No.	References	Optimum from literature	m	n	s	Type of problem	Description
1	Q.T. Pham,[23,31] Lapidus and Luus,[8] Bojkov and Luus[9]	339.1	8	4	19	Maximisation	CSTR
2	Dadebo and McAuley,[5] Rajesh et al.,[27] Ray,[33] Renfro et al.[34]	0.61077	2	1	39	Maximisation	Batch reactor consecutive reactions, A→B→C
3	Rajesh et al.,[27] Yeo,[35] Luus[6]	0.12011	4	1	31	Minimisation	Nonlinear unconstrained mathematical system
4	Rajesh et al.,[27] Luus[7]	0.76159	2	1	15	Minimisation	Unconstrained mathematical system
5	Dadebo and McAuley,[5] Rajesh et al.,[27] Logsdon and Biegler[36]	0.57353	2	1	19	Maximisation	Tubular reactor parallel reaction
6	Dadebo and McAuley,[5] Rajesh et al.,[27] Jackson[37]	0.47695	2	1	39	Maximisation	Plug flow reactor catalyst blend, A↔ B→C
7	Rajesh et al.,[27] Tieu et al.[38]	0.48005	2	1	19	Maximisation	Consecutive reactions, A→B→C
8	Rajesh et al.,[27] Luus[7] constraint $y_1(t_f) = 1$	0.92518	2	1	15	Minimisation	Constrained mathematical system

```
Initialise N random solutions.
Iterate until a stopping criterion is met
        Evaluate the fitness of the population.
        Sort the solutions according to their fitness.
        // waggle dance
        Select the highest-ranking m solutions for neighbourhood search.
        Assign x=ne foragers to each of the e≤m top-ranking elite solutions.
        Assign x=nb≤ne foragers to each of the remaining m – e selected solutions.
        // local search
        For each of the m selected solutions
                Create x new solutions by perturbing the selected solution randomly or otherwise.
                Evaluate the new solutions.
                Retain the best solution amongst the selected and new solutions.
                Adjust the neighbourhood of the retained solution.
        Re-initialise sites where search has stagnated for a given number of iterations.
        // global search
        Repeat N – m times
                Generate randomly a new solution.
                Evaluate the new solution.
        Form new population (m solutions from local search, N – m from global search)
```

Figure 1. Pseudocode of the standard bees algorithm.

on a site of higher fitness than the scout, that forager becomes the new scout and participates in the waggle dance in the next generation. If none of the foragers manages to improve the fitness of the solution found by the scout, the size of the neighbourhood is shrunk (*neighbourhood shrinking procedure*). The purpose of the neighbourhood shrinking procedure is to make the search increasingly focused in the proximity of the local peak of fitness. If the local search fails to bring any improvement in fitness for a given number *stlim* of iterations, the local peak is abandoned and the scout bee is randomly re-initialised (*site abandonment procedure*). The purpose of the site abandonment procedure is to prevent the algorithm from being stuck into local peaks of performance.

Finally, the remaining $ns = N - nb$ scouts are randomly re-initialised (global search phase). At the end of one iteration, the new population is formed by the *nb* scouts resulting from the local exploitative search and *ns* scouts resulting from the global explorative search. For each learning cycle, the following number of fitness evaluations is performed

$$ev = ne \cdot nre + (nb - ne) \cdot nrb + ns \tag{6}$$

For a more detailed description of the procedure, the reader is referred to D.T. Pham and Castellani.[42]

The implementation of the bees algorithm involves a minimal amount of problem domain expertise. In most cases, this expertise is restricted to the criterion for the definition of the fitness evaluation function. Furthermore, the bees algorithm makes no assumption on the problem domain, such as the differentiability of the fitness landscape.

The modified bees algorithm

In the standard bees algorithm,[42] neighbourhood exploration is the only evolution operator. As mentioned above, this operator generates offspring (foragers) randomly within a small subspace around the parent (scout bee).

The modified bees algorithm uses a set of operators which may be expanded at will depending on the type of problem. For generic continuous variable problems where each solution is a vector of real numbers, the following set may be used:[32] mutation, creep, crossover, interpolation and extrapolation.[46] Mutation consists of assigning random values, uniformly distributed within the variables' range, to randomly chosen elements of the variables vector. Creep consists of applying small Gaussian changes (standard deviation equal to 0.001 of the variable range) to all elements of the solution vector. Crossover consists of combining the first one or more elements of a parent vector with the remaining elements of a second parent vector. Interpolation is defined by

$$\mathbf{x}^{Offspring} = 0.7\mathbf{x}^{A} + 0.3\mathbf{x}^{B} \tag{7}$$

where *A* is the fitter parent and *B* the less fit parent, giving a solution which is biased to the first. Extrapolation is defined by

$$\mathbf{x}^{Offspring} = 1.3\mathbf{x}^{A} - 0.3\mathbf{x}^{B} \tag{8}$$

For dynamic optimisation problems, three more operators are introduced: swap, shift and smooth. *Smooth* performs a rolling average of all the control variables u_i during a random time interval t_{j0} to t_{j1}

$$\mathbf{u}^{new}(t_{j0}) = \frac{2}{3}\mathbf{u}^{A}(t_{j0}) + \frac{1}{3}\mathbf{u}^{A}(t_{j0+1}) \tag{9}$$

$$\mathbf{u}^{new}(t_{j}) = \frac{1}{4}\mathbf{u}^{A}(t_{j-1}) + \frac{1}{2}\mathbf{u}^{A}(t_{j}) + \frac{1}{4}\mathbf{u}^{A}(t_{j+1}),$$
$$j = j0 + 1, \ldots, j1 - 1 \tag{10}$$

$$\mathbf{u}^{new}(t_{j0}) = \frac{1}{3}\mathbf{u}^{A}(t_{j1-1}) + \frac{2}{3}\mathbf{u}^{A}(t_{j1}) \tag{11}$$

Table 2. Operator probabilities.

Extrapolate	0.2
Crossover	0
Interpolate	0
Mutation	0.05
Creep	0.5
Swap	0
Shift	0.2
Smooth	0.05

Shift causes the value of all control variables u_i at time t_{j0} to spread to earlier or later times, so that the control vector becomes flat between two random times t_{j0} and t_{j1}. If $t_{j0} < t_{j1}$ then the u-values spread to later times, else they spread to earlier times

$$\mathbf{u}(t_j) = \mathbf{u}(t_{j0}), \qquad j = j0, \ldots, j1 \tag{12}$$

Swap interchanges the value of a randomly chosen gene x_i and its neighbour

$$\begin{aligned} x_{j0}^{new} &= x_{j0+1}^{A} \\ x_{j0+1}^{new} &= x_{j0}^{A} \end{aligned} \tag{13}$$

The probabilities of all the operators were kept unchanged for all problems in the present work at the values recommended by Q.T. Pham[31] (Table 2).

From one generation to the next, the best of the population is retained (truncation selection). Each surviving solution or 'bee' is then given an opportunity to improve by evolving a few times using the above operators. At each step, if a change results in an improvement, the new solution replaces its parent, else the original solution is retained. The number of search steps, m, is a decreasing function of rank

$$m = \text{int}\left[(n_0 - 1)\left(\frac{m - Rank}{m - 1}\right)^f\right] + 1 \tag{14}$$

where *Rank* is 1 for the fittest member. n_0 is the number of search steps applied to the fittest member of a generation. The least fit selected member always receives only one search step. The exponent f is an indication of the selective pressure: for $f = 1$ the number of search steps varies linearly with rank, while for $f > 1$ the number of search steps decreases more quickly with rank.

Another difference between the modified and standard versions of the bees algorithm is the introduction of the category of 'young bees'.[32] At each generation, after each member has received its allocated number of evolution steps, the least fit individuals are deleted and new random bees are created to replace them. However, forcing new solutions to compete immediately with previous solutions, which have evolved for a long time, is counterproductive, just as it would be in a society where newborns were forced to compete with adults without going through a period of nurturing. Particularly in the case of multimodal objective functions, a new solution may land at the foot a high peak and yet be still worse than old solutions that have stalled at a lesser peak. Therefore, 'young members' only compete among themselves, until they reach 'adulthood' after evolving more than M steps. In summary, the algorithm strongly favours the best members, but also continuously creates young members and nurtures them, so that the pool of 'best members' may be continuously rejuvenated if necessary.

```
Initiate N random bees
Iterate
    Sort all the bees in the present population
    Select the e fittest bees into the new population
    Sort the young bees (age ≤ M) in the present population
    Select the g fittest young bees into the new population
    Create N – e – g random new bees and put them in the new population
    Let each bee evolve m times, where m is given by equation(14) (*)
Until a stopping criterion is met

(*) In the evolution steps, an offspring replaces its parent only if it is better than the parent.
```

Figure 2. Pseudocode of the modified bees algorithm.

The pseudocode of the modified bees algorithm is given in Figure 2.

Progressive step reduction

Q.T. Pham[31] originally introduced the PSR technique to tune an EA. The procedure starts with a very small number (two or three) of sampling points s for each control variable u_i, then doubles the number of intervals every few generations until the desired number of control steps s (see 'Dynamic optimisation' section above) is reached. This is analogous to sketching the general features of a control profile, and then progressively refining it. The technique was found to considerably improve the quality of the solutions generated by an evolutionary algorithm for a set of dynamic optimisation problems.[31] In the present study, the use of the PSR method to enhance the dynamic optimisation performance of the modified bees algorithm is evaluated in the factorial experiment.

Modified bees algorithm parameter tuning

Compared to the standard version, the modified bees algorithm is characterised by a more sophisticated search procedure featuring a set of additional operators. The action of these operators is adjusted by tuning a number of system parameters. Unfortunately, understanding and estimating precisely the effect of these parameters on the performance of the algorithm is difficult, and the task is complicated by interactions among different operators.

As a consequence of the need of tuning a relatively large number of system parameters, the optimisation of the modified bees algorithm via standard trial-and-error tuning[42] is likely to be time consuming, and may yield sub-optimal results. Therefore, the statistical

approach proposed by Q.T. Pham[31] is used in the present work. This section discusses the algorithm optimisation problem and describes the proposed procedure.

Tuning procedures for population-based algorithms

One of the main factors that make it difficult to evaluate the effect of the tuning of a single parameter on the algorithm's behaviour is the significant interactions between the action of the different operators. In addition, global population-based metaheuristics of the kind analysed in this article are based on non-deterministic (stochastic) search processes. As a result, the outcome of a single run of a procedure is not repeatable, and the evaluation of the effect of a given parameter setting is affected by a noise problem.

Given an optimisation algorithm, the No Free Lunch Theorem[47] entails that there is no universal tuning that works best on all possible problems. Nevertheless, classes of similar mathematical and scientific problems share certain characteristic patterns, which correspond to regularities in the fitness landscape of the candidate solutions. On these classes of fitness landscapes, some tunings work better than others. That is, each parameter configuration allows the algorithm to exploit best the features of a given number of optimisation problems, and conversely is unsuitable to other problem domains. As a result, for certain classes of problem it may be possible to derive a common tuning based on a number of tests.

The importance of matching each optimisation problem with the right configuration for the optimiser has brought some researchers to consider parameter selection as one of the most important elements in the building of a successful application.[48]

The tuning of an algorithm is itself an optimisation task. It can be considered as a multi-objective optimisation problem, where speed and robustness are often contradictory requirements. Most practitioners carry out at least some manual tuning of the algorithm.[49] In many cases, due to the difficulty of tuning complex non-deterministic algorithms, the engineer needs to engage in time consuming trial-and-error procedures.

Grefenstette[50] used an automatic meta-optimisation method where an EA is used to optimise the performance of another EA. Unfortunately, due to the computational complexity of this kind of algorithm, the procedure is very time consuming for any non-trivial application. Moreover, because of the noisy nature of the results of stochastic search methods, there is the problem of evaluating precisely the fitness of the candidate solutions (i.e. parameter settings). More efficient meta-optimisation algorithms for parameter tuning have been developed such as REVAC (relevance estimation and value calibration of EA parameters)[51,52] and sequential parameter optimisation (SPO).[53,54] Grid searches over the parameter space are sometimes used, as in Francois and Lavergne,[55] where two parameters were optimised by discretising one parameter into 10 levels and the other into 20 levels. While this gives a complete picture of the EA's behaviour for the test problems considered, it would be difficult to use this approach if there are many more than two parameters to be tuned.

Adaptive tuning is a popular and effective approach. It was used to adapt the mutation step size in evolutionary strategies by Rechenberg[56] and first applied to genetic algorithms by Davis.[57] A recent implementation uses the statistical outlier detection method to estimate operator potentials and adjust operator frequencies.[58] Another type of adaptive tuning takes the form of competitive evolution between separate but interacting populations of tuning solutions.[46]

Adaptive tuning is conceptually attractive to users of population-based metaheuristics. However, it does not come for free since some computational resources have to be spent on the adaptation process. On a more fundamental level, adaptive tuning is a hill climbing procedure (with some randomness arising from noise), and there is no guarantee that it will converge to the global optimum. This problem is mitigated if the gradient-based search is started from a promising starting point (i.e. a solution of high performance).

Recently, Q.T. Pham[31] proposed a statistics-based tuning method where a two-level factorial experiment is carried out, with operator probabilities taking either a low (usually zero) or high value in each test. The effect of each operator is then quantified and the results are used to tune the algorithm.

Two-level factorial experiment design

To investigate the usefulness of some of the features and get a rough tuning for the others, a two-level full factorial experiment was carried out, taking into account all combinations of the following variables' values.

- Population size, N: 10 or 50.
- PSR: Not used or used.
- Fraction of population chosen to survive based on fitness, e: 0.5 or 0.8.
- Fraction of 'young' bees chosen to survive, g: 0 or 0.5.
- Maximum number of neighbourhood search steps per generation, m: 2 or 10.
- Linearity parameter, f: 2 or 5.
- Age at which young becomes adult, M: 5 or 10.

In statistics terminology, the first (lower) value is termed level -1 and the second (higher) value is termed level $+1$.

Tuning procedure

The tuning procedure first described by Q.T. Pham et al.[32] was used. The procedure is based on the following principles:

1. Calculate the weighted mean level of the parameters in the worst 10% of the runs (–1 is the low level of the factor, +1 is the high level). Weights are adjusted to vary linearly with rank, the worst run having a weight of 1, the 10th percentile a weight of 0. This gives the *mean worst settings* for the problem.
2. Calculate the weighted mean level of the parameters in the best 10% of the runs in same manner (best run has maximum weight). This gives the *mean best settings* for the problem.
3. Repeat steps 1 and 2 for several test problems and calculate the overall mean 'best' and 'worst' level of each parameter.
4. If the mean worst level of a parameter is low (from –1 to –0.33), set that parameter to the maximum level. If the mean worst level of a parameter is high (from +0.33 to +1), set that parameter to the minimum level. In other cases (from –0.33 to +0.33), set that parameter to the average of the best tunings.

Figure 3. Detailed bees algorithm tuning procedure.

- parameter settings that cause poor performance must be avoided;
- if both (low and high) parameter settings are equally likely to yield poor performance, the parameter setting that is more likely to give good performance is chosen.

These principles are based on the observation that, while the best runs usually give results that are very similar, the worst runs form a very long tail. Moreover, the results of the worst runs are much further below the median than the results of the best runs are above the median. Therefore, the main concern in tuning an algorithm is to prevent the occurrence of bad runs.

The best and worst settings are based on the weighted means of the top 10% and bottom 10% of the runs, respectively. For each setting, five independent runs were executed for benchmarks 2 to 8, and 10 for benchmark 1. Preliminary investigations have indicated that if fewer runs are used, statistical fluctuations may become significant, while if more are used the differences between good and bad tunings are weakened. The detailed tuning procedure is given in Figure 3.

Tables 3 and 4 give the best and worst levels of the parameters for the test problems, and their averages. The clearest effect is that of population size N. For all problems, a low population size gives better results. The best level of N for each problem varies from -0.98 to -0.95

Table 3. Best levels of parameters.

Problem	N	PSR	e_r	g_r	n_0	f	M
1	−0.97	−0.71	−0.18	0.35	0.63	0.15	0.21
2	−0.98	0.96	−0.12	0.25	0.98	0.22	−0.03
3	−0.97	0.17	−0.06	0.56	0.04	0.29	0.10
4	−0.97	−0.50	−0.27	0.39	0.27	0.18	−0.02
5	−0.96	0.90	0.28	0.30	0.74	−0.35	0.08
6	−0.97	0.28	−0.22	0.48	0.49	0.38	−0.09
7	−0.97	−0.91	−0.10	0.15	0.87	0.36	−0.04
8	−0.95	−0.33	−0.18	0.47	0.44	0.23	0.31
Mean best level	**−0.97**	**−0.02**	**−0.10**	**0.37**	**0.56**	**0.19**	**0.06**

Table 4. Worst levels of parameters.

Problem	N	PSR	e_r	g_r	n_0	f	M
1	0.97	−0.77	−0.03	−0.81	−0.72	−0.09	−0.15
2	0.98	−0.98	−0.07	−0.63	−0.78	−0.11	−0.04
3	0.97	−0.97	0.10	−0.64	−0.71	−0.07	−0.10
4	0.97	−0.75	−0.04	−0.32	−0.57	−0.07	0.17
5	0.97	−0.97	−0.01	−0.74	−0.79	−0.08	−0.04
6	0.97	−0.97	0.05	−0.54	−0.85	−0.03	0.04
7	0.97	−0.97	0.07	−0.48	−0.79	−0.17	0.01
8	0.96	−0.97	0.01	−0.50	−0.66	−0.18	−0.10
Mean worst level	**0.97**	**−0.92**	**0.01**	**−0.58**	**−0.74**	**−0.10**	**−0.03**

(Table 3), where −1 corresponds to a population size of 10 and + 1 to one of 50. On the other hand, the worst level of N varies from + 0.96 to + 0.98 (Table 4).

PSR does not always give the best results: half of the problems have a negative 'best level' for PSR, which means that most of the best runs for those are without PSR in those problems (Table 3). However, the worst level of PSR varies from −0.75 to −0.98 (Table 4). Thus, although PSR may not always give the very best results, it certainly provides a guarantee against poor results and is worth applying. Other parameters about which one can draw definite conclusions are g_r (avoid low level) and n_0 (avoid low level). The effects of the other parameters are ambiguous or neutral.

Table 5 shows how the recommended tuning is calculated using the proposed procedure. The first row (after the headings) gives the weighted mean worst levels (bottom 10% of runs) as calculated in the first step. The second row converts the values to qualitative judgements. The third row gives the mean best levels in the second step of the procedure. The fourth row calculates the recommended levels according to step 4 of the procedure. The actual parameter settings are calculated from the recommended level (row 4) and the definitions of high (row 5) and low (row 6) levels, and entered in the last row.

The recommended tuning according to the proposed factorial technique is given in the last row of Table 5. With the only exception of PSR (which is applicable only to dynamic control problems), the derived tuning is almost exactly the same as for the analytical test problems considered by Q.T. Pham et al.[32] This might imply that, within the limitations imposed by the No Free Lunch Theorem, this parameter configuration is fairly robust and relatively independent of the problems considered.

Experimental setup

In order to evaluate the effectiveness of the modified bees algorithm, its performance was compared with that of the standard bees algorithm, an EA and PSO. All four optimisation algorithms encode the candidate solutions as $n \times s$-dimensional vectors of decision variables as defined in equation (5). The duration of each algorithm is set to obtain approximately the same number of function evaluations of about 2000 (a lesser number of evaluations gave insufficient separation between the algorithms for most of the problems).

Evolutionary algorithm

The EA used in this work is described more in detail elsewhere.[42] The parameters and operators were chosen based on the results of a set of preliminary EA optimisation trials. The algorithm uses the fitness ranking procedure[43] to select the pool of reproducing individuals. Generational replacement with elitism[43] is employed to update the population at the end of each evolution cycle.

The mutation operator modifies the value of each of the variables of a solution by a small amount. This amount is randomly re-sampled for each variable with uniform probability as follows

$$x_{new} = x_{old} + a \cdot random \cdot \frac{x_{max} - x_{min}}{2} \tag{15}$$

where $x \in [x_{min}, x_{max}]$ is the variable being mutated, *random* is a number drawn with uniform probability in the interval $[-1, 1]$ and a is a system parameter defining the maximum width of the mutation. The parameter a is akin to *ngh* which determines the neighbourhood of a standard bees algorithm's flower patch. The mutation width a is encoded in as an extra variable in the chromosome of the individuals, and is subjected to the evolution process.

Following the results of D.T. Pham and Castellani,[42] genetic crossover is not employed in the evolutionary procedure. The preliminary optimisation trials confirmed the superior performance of this configuration.

Overall, due to the lack of genetic recombinations, the adaptation of the mutation width and the real encoding of the solutions, the EA procedure can be considered akin to the evolutionary programming[43] approach.

The values of the EA learning parameters are shown in Table 6. They are set to obtain 2040 fitness evaluations per optimisation trial.

Particle swarm optimisation

The algorithm used in the present study is the standard PSO procedure described by Kennedy and Eberhart.[44]

Table 5. Tuning of algorithm for dynamic control optimisation problems.

Parameter	N	PSR	e_r	g_r	n_0	f	M
Worst level (−1 to + 1)	0.97	−0.92	0.01	−0.58	−0.74	−0.10	−0.03
Worst (High/Low/Indifferent)	H	L	I	L	L	I	I
Best level (−1 to + 1)	−0.97	−0.02	−0.10	0.37	0.56	0.19	0.06
Recommended level (−1 to + 1)	−1	+ 1	−0.10	+ 1	+ 1	−0.10	−0.03
Level −1 parameter value	10	No	0.5	0.0	2	2	5
Level + 1 parameter value	50	Yes	0.8	0.5	10	5	10
Recommended actual value	10	Yes	0.7	0.7	10	2.5	7

Table 6. EA learning parameters.

Parameter	Value
Population size	20
Children per generation	19
Mutation rate (variables)	0.8
Mutation rate (mutation width)	0.8
Initial mutation interval width, a (variables)	0.1
Initial mutation interval width, ρ (mutation width)	0.1
No. of evaluations per iteration	20
No. of iterations	102
Total number of fitness evaluations	1040

The movement (velocity) of each particle is defined by three elements, namely the particle's momentum, personal experience (i.e. attraction force towards the best solution found so far) and social interaction with its neighbours (i.e. attraction force towards the best solution found by the neighbours). Each velocity vector component is clamped to the range $[-v_i^{max}, v_i^{max}]$, where

$$v_i^{max} = k \cdot \frac{max_i - min_i}{2} \tag{16}$$

The PSO parameters were set as recommended by Shi and Eberhart,[59] except for the connectivity (i.e. the number of neighbours per particle) and maximum speed which were set according to experimental trial and error. The maximum speed was adjusted via the multiplication factor k (see equation (16)). The values of the PSO learning parameters are detailed in Table 7. They are set to obtain 2040 fitness evaluations per optimisation trial.

Standard bees algorithm

The procedure outlined in 'The standard bees algorithm' subsection was used. The learning parameters were optimised experimentally. They are given in Table 8. They are set to obtain 2037 fitness evaluations per optimisation trial.

Modified bees algorithm

The learning parameters and operator probabilities were set according to Tables 2 and 5. The algorithm is run for 57 evolution cycles, which correspond to a total of 2031 fitness evaluations per optimisation trial.

Table 7. PSO learning parameters.

Parameter	Value
Population size	20
Connectivity (no. of neighbours)	4
Maximum velocity, k	0.1
c_1	2.0
c_2	2.0
Inertia weight at start, w_{max}	0.9
Inertia weight at end, w_{min}	0.4
No. of evaluations per iteration	20
No. of iterations	102
Total number of fitness evaluations	1040

Table 8. Standard bees algorithm learning parameters.

Parameter	Name	Value
No. of scout bees	*ns*	1
No. of elite sites	*ne*	1
No. of best sites	*nb*	3
Recruited bees for elite sites	*nre*	10
Recruited bees for remaining best sites	*nrb*	5
Initial size of neighbourhood	*ngh*	0.5
Limit of stagnation cycles for site abandonment	*stlim*	10
No. of evaluations per iteration	*ev*	21
No. of iterations		97
Total number of fitness evaluations		2037

Experimental results

The performance of each algorithm was evaluated on the average results of 30 independent optimisation trials. The statistical significance of the differences of the results obtained by the four algorithms was pairwise evaluated using Mann–Whitney U-tests. The U-tests were run for a 0.05 (5%) level of significance.

For each dynamic optimisation benchmark, Table 9 reports the main statistical estimates of the fitness of the solutions obtained by the four metaheuristics. For each benchmark, the last column of Table 9 indicates the algorithm(s) that obtained the best results (i.e. statistically significantly superior). Figures 4–11 visualise the median and 10th and 90th percentiles of the results obtained by the four algorithms, in which stdBA means the standard bees algorithm and modBA is the modified bees algorithm. Figure 12 shows the learning curves of the modified bees algorithm for the eight benchmark problems. The plots show the evolution of the average, minimum and maximum fitness of 20 independent runs.

The proposed optimisation method performed very effectively and consistently on most of the dynamic optimisation problems. In six out of eight cases, the modified bees algorithm ranked as the top performer. The standard bees algorithm excelled on benchmarks 4 and 8, while PSO obtained top results on function 3. Compared to PSO, the standard bees algorithm obtained better results in four cases, comparable results (i.e. no statistically significant difference) on three benchmarks, and worst results in one case.

In terms of robustness, the two bees-inspired methods excelled. This result is apparent from the comparison of the standard deviation and upper and lower percentiles of the distribution of the optimisation results. With the exception of the trials on functions 3 and 8, the modified bees algorithm showed the most consistent performances. The EA consistently ranked as the worst in terms of quality of the solutions and robustness.

Table 9. Experimental results.

	Mean	SD	Median	P10	P90	Best
Function 1 – maximisation task						
modBA	338.6855	0.3502	338.7670	338.2505	339.0647	X
stdBA	328.4946	1.7868	328.3990	326.1050	330.8190	
PSO	334.3692	0.5269	334.4450	333.6685	334.9475	
EA	312.1288	3.7074	312.6080	306.5485	316.9305	
Function 2 – maximisation task						
modBA	0.6107	0.0000	0.6107	0.6107	0.6108	X
stdBA	0.6103	0.0002	0.6104	0.6100	0.6105	
PSO	0.6102	0.0003	0.6103	0.6099	0.6104	
EA	0.6074	0.0010	0.6074	0.6064	0.6088	
Function 3 – minimisation task						
modBA	0.1238	0.0065	0.1206	0.1202	0.1363	X
stdBA	0.1325	0.0014	0.1323	0.1309	0.1346	
PSO	0.1313	0.0022	0.1308	0.1296	0.1331	X
EA	0.1398	0.0066	0.1376	0.1337	0.1504	
Function 4 – minimisation task						
modBA	0.7618	0.0001	0.7617	0.7617	0.7619	
stdBA	0.7595	0.0000	0.7595	0.7595	0.7595	X
PSO	0.7624	0.0018	0.7621	0.7604	0.7652	
EA	0.7865	0.0238	0.7779	0.7682	0.8172	
Function 5 – maximisation task						
modBA	0.5733	0.0001	0.5734	0.5732	0.5735	X
stdBA	0.5718	0.0011	0.5722	0.5699	0.5729	
PSO	0.5709	0.0016	0.5714	0.5696	0.5724	
EA	0.5634	0.0057	0.5644	0.5575	0.5681	
Function 6 – maximisation task						
modBA	0.4772	0.0002	0.4773	0.4769	0.4774	X
stdBA	0.4758	0.0004	0.4759	0.4752	0.4762	
PSO	0.4736	0.0011	0.4738	0.4725	0.4748	
EA	0.4697	0.0016	0.4696	0.4675	0.4719	
Function 7 – maximisation task						
modBA	0.4800	0.0001	0.4800	0.4799	0.4801	X
stdBA	0.4795	0.0003	0.4796	0.4795	0.4796	
PSO	0.4745	0.0036	0.4754	0.4687	0.4789	
EA	0.4716	0.0048	0.4720	0.4683	0.4763	
Function 8 – minimisation task						
modBA	0.9378	0.0095	0.9355	0.9275	0.9534	
stdBA	0.9211	0.0006	0.9209	0.9206	0.9219	X
PSO	0.9301	0.0114	0.9247	0.9226	0.9476	
EA	0.9644	0.0326	0.9544	0.9288	0.9938	

SD: standard deviation; P10: 10th percentile; P90: 90th percentile.

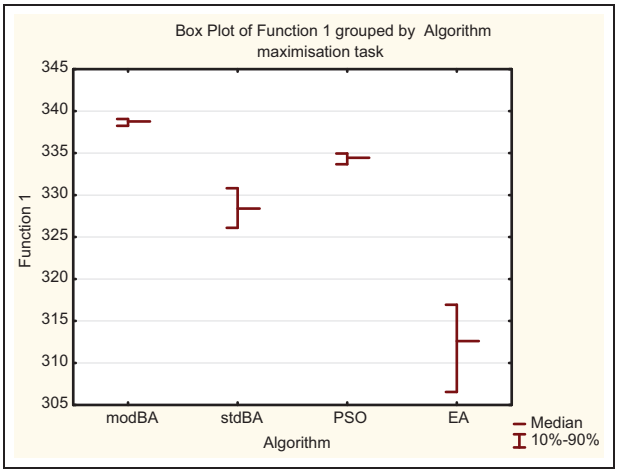

Figure 4. Function 1 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

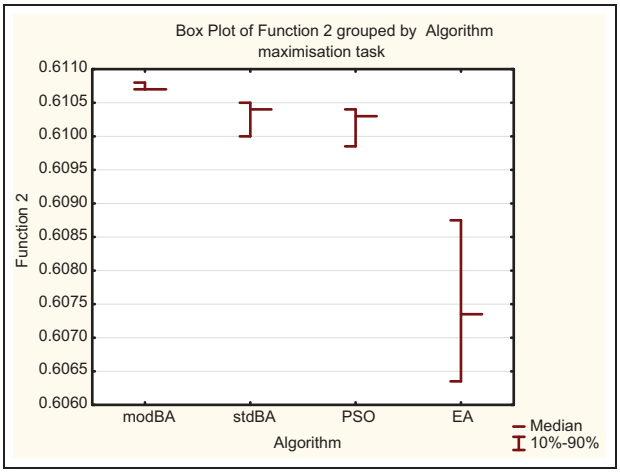

Figure 5. Function 2 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

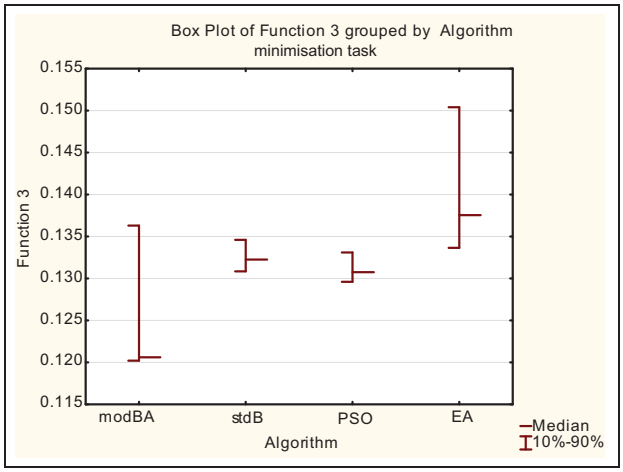

Figure 6. Function 3 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

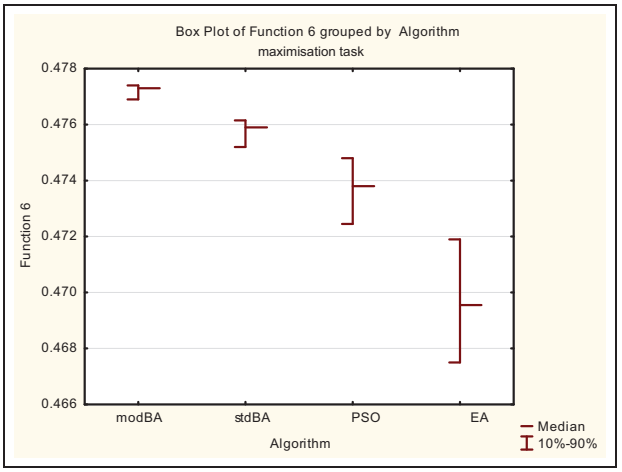

Figure 9. Function 6 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

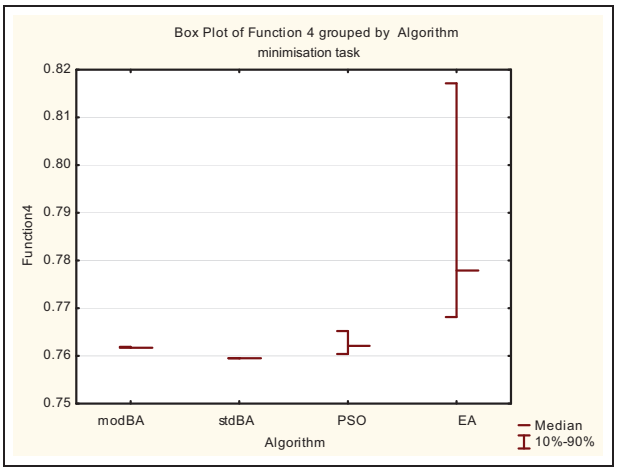

Figure 7. Function 4 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

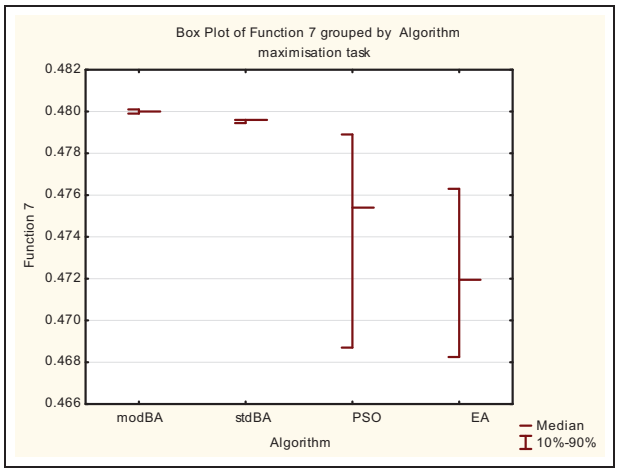

Figure 10. Function 7 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

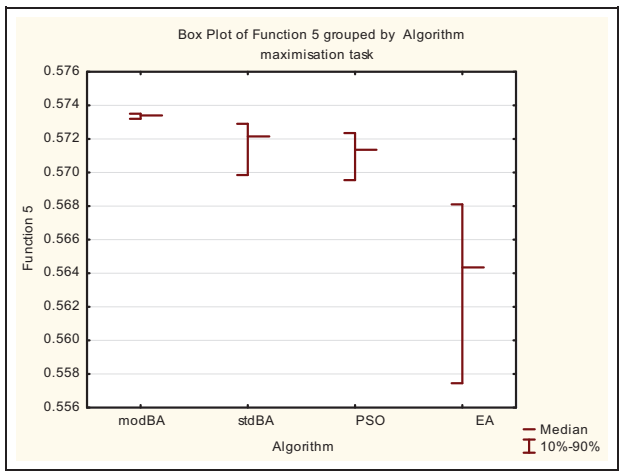

Figure 8. Function 5 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

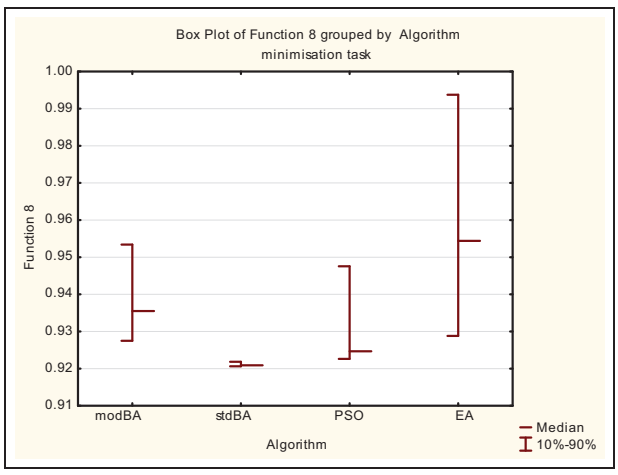

Figure 11. Function 8 – optimisation results (median, 10%–90% percentile) for the algorithms tested.

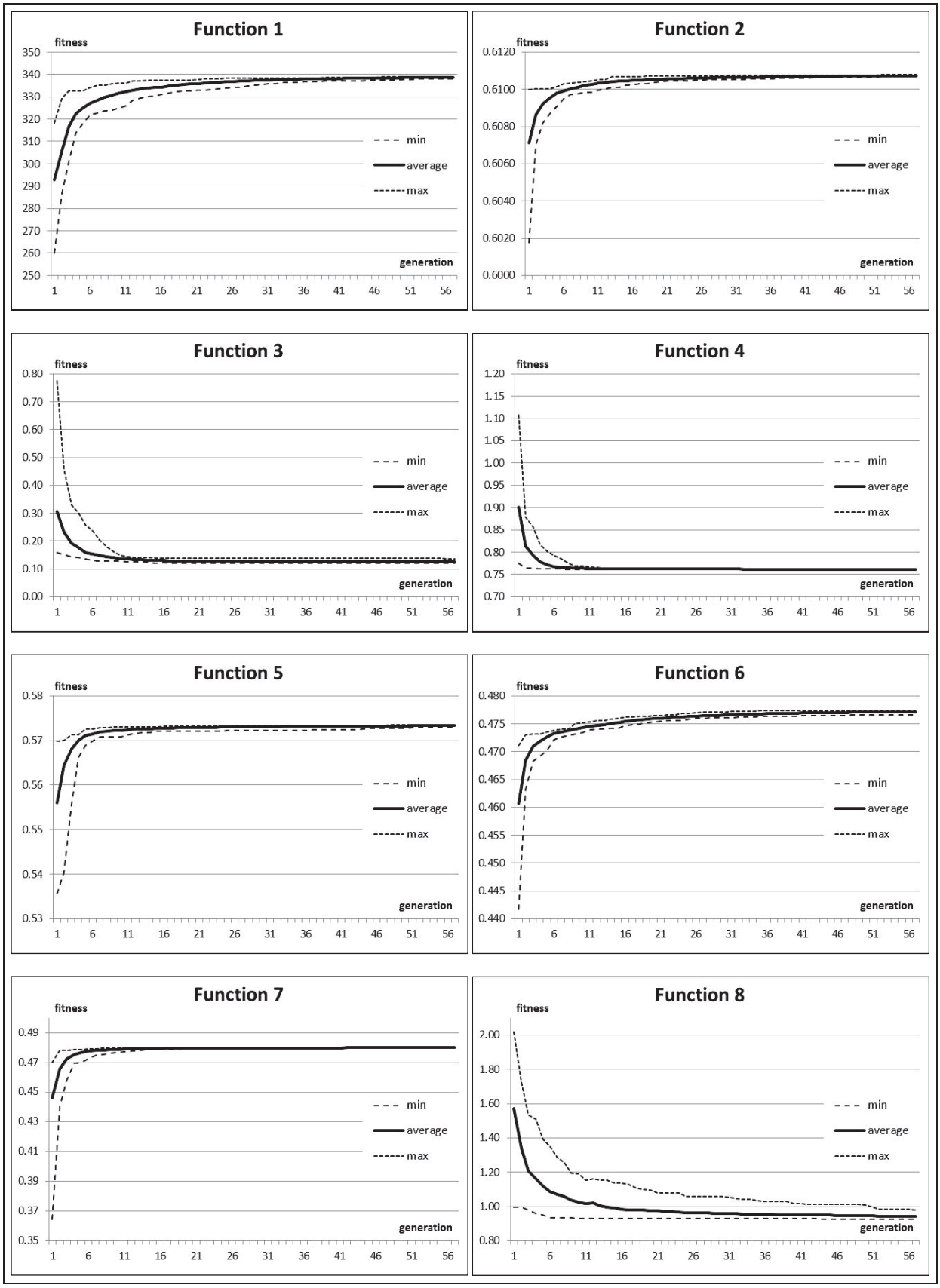

Figure 12. Learning curves of the modified bees algorithm.

In order to have a more complete evaluation of the competitiveness of the proposed method, the results obtained by the modified bees algorithm were compared with the state-of-the-art in the literature. Table 10 shows for each benchmark the known optimum published in the literature, the optimum obtained

Table 10. Comparison with literature.

Function	Task	Literature	Ant algorithm[27]	EADO[31]	modBA	%
1	Maximisation	339.1000	–	**338.9200**	338.6855	0.1224
2	Maximisation	0.6108	0.6105	**0.6107**	**0.6107**	0.0115
3	Minimisation	0.1201	0.1290	0.1241	**0.1238**	2.9806
4	Minimisation	0.7616	0.7624	**0.7616**	0.7618	0.0276
5	Maximisation	0.5735	0.5728	**0.5733**	**0.5733**	0.0401
6	Maximisation	0.4770	0.4762	0.4770	**0.4772**	0.0524
7	Maximisation	0.4801	0.4793	**0.4800**	**0.4800**	0.0104
8	Minimisation	0.9252	**0.9255**	0.9584	0.9378	1.3457

by Rajesh et al.[27] using ant colony optimisation, the optimum achieved by Q.T. Pham[31] using an evolutionary algorithm specially adapted for dynamic optimisation (EADO), and the arithmetic mean of the fitness of the solutions obtained in the 30 optimisation trials using the modified bees algorithm. The result of the best performing procedure is highlighted in bold. The last column reports the difference between the result achieved by the modified bees algorithm and the known optimum, expressed as a percentage of the optimum value.

The figures referring to the known optima refer to either analytical results or the best-so-far results obtained using other optimisation techniques. The results taken from Rajesh et al.[27] are the average of 25 independent optimisation trials. Unfortunately, Rajesh and colleagues did not specify which kind of average measure they used, whether it was the mean, the median or other. The results taken from Q.T. Pham[31] are the arithmetic mean of 10 independent runs.

Notwithstanding the caveats due to the differences in the estimation of the average solutions, Table 10 allows to one gauge the competitiveness of the proposed procedure. In five out of eight cases, the modified bees algorithm gave the best optimisation results. The solutions generated by the modified bees algorithm were also very close in fitness to the known optimum. Namely, they differed from the published optima by less than a percentage point in six cases out of six, and never by more than 3%.

The experimental results highlighted also the effectiveness of the proposed parameter tuning method. Despite the fact that it was created only for a first rough adjustment of the system parameters, the procedure allowed to find a highly effective and consistent algorithm configuration. Compared to the customary tedious manual tuning process, the proposed tuning method proved to be a valid and effective alternative.

Conclusions

The current article presented an application of a modified version of the bees algorithm to the optimisation of dynamic control problems in chemical engineering. Compared to the standard bees algorithm, the modified procedure uses an additional set of search operators, and allows newly generated bees to develop for a few optimisation cycles without competing with mature individuals. As a result of the enlarged set of operators, a relatively large set of system parameters needs to be adjusted in order to optimise the performance of the modified bees algorithm. These parameters are tuned using a statistical tuning procedure.

The performance of the proposed procedure was evaluated on a set of eight popular dynamic optimisation problems. The results obtained by the modified bees algorithm were compared with those obtained using the standard procedure and two other, well known, nature-inspired optimisation methods. The experimental results proved the effectiveness and consistency of the proposed algorithm. The performance of the modified bees algorithm compared also well with that of the state-of-the-art methods in the literature. The solutions obtained using the modified bees algorithm were always close to the known optima of the eight benchmarks.

The results of the proposed work should not lead the reader to assume the superiority of the modified bees algorithm over the other algorithms compared. Indeed, any such interpretation is openly in contradiction with the No Free Lunch Theorem. However the present study has demonstrated that the modified bees algorithm is capable of performing very competitively on the kind of dynamic optimisation problems used.

Experimental evidence has also proved the effectiveness of the statistical parameter tuning method. Despite being designed only for a first rough tuning of the parameters, the procedure obtained a high performing and robust algorithm configuration. The usefulness of the proposed tuning approach should also be evaluated in consideration of the complexity and subjectivity of the standard trial-and-error procedures. To be applied, the statistical parameter tuning technique requires only some knowledge of a reasonable setting range (low and high levels) for each parameter.

At present, the proposed tuning procedure has successfully been applied to the optimisation of EAs[31] and the modified bees algorithm (as reported by Q.T. Pham et al.[32] and in the present work). Further work should assess its effectiveness on other complex optimisation algorithms. The efficacy of the proposed statistical parameter tuning procedure should also be compared to that of other procedures such as the Taguchi method.[60]

Funding

This work was partially supported by the EC FP6 Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

References

1. Pontryagin LS, Boltyanski VG, Gamkrelidze RV, et al. *The mathematical theory of optimal processes*. New York: Wiley, 1962.
2. Bryson AE. *Dynamic optimization*. Menlo Park, CA: Editorial Addison Wesley Longman, Inc., 1999.
3. Srinivasan B, Palanki S and Bonvin D. Dynamic optimization of batch processes I. Characterization of the nominal solution. *Comput Chem Eng* 2003; 27(1): 1–26.
4. Bellman RE. *Dynamic programming*. Princeton, NJ: Princeton University Press, 1957.
5. Dadebo SA and McAuley KB. Dynamic optimization of constrained chemical engineering problems using dynamic programming. *Comput Chem Eng* 1995: 19(5): 513–525.
6. Luus R. Optimal control by dynamic programming using systematic reduction in grid size. *Int J Contr* 1990; 51(5): 995–1013.
7. Luus R. Application of dynamic programming to final state constrained optimal control problems. *Hung J Ind Chem* 1991; 19: 245–254.
8. Lapidus L and Luus R. *Optimal control of engineering processes*. Waltham, MA: Blaisdell Publishing, 1967.
9. Bojkov B and Luus R. Evaluation of parameters used in iterative dynamic programming. *Can J Chem Eng* 1993; 71(3): 451–459.
10. Luus R. In search for global optima in nonlinear optimal control problems. In: *Proceedings of 6th IASTED international conference*, Honolulu, HI, 23–25 August 2004, pp.394–399, ACTA Press.
11. Biegler LT. An overview of simultaneous strategies for dynamic optimization. *Chem Eng Process, Process Intensification* 2007; 46(11): 1043–1053.
12. Goh CJ and Teo LK. Control parameterization: a unified app roach to optimal control problems with general constraints. *Automatica* 1988; 24(1): 3–18.
13. Rangaiah GP. Studies in constrained optimization of chemical process problems. *Comput Chem Eng* 1985; 9(4): 395–404.
14. Hargraves CR and Paris SW. Direct trajectory optimization using nonlinear programming and collocation. *J Guid Contr Dyn* 1987; 10(4): 338–342.
15. Edgar TF and Himmelblau DM. *Optimization of chemical processes*. New York: McGraw-Hill, 1989, pp.369–371.
16. Binder T, Cruse A, Villar CAC, et al. Dynamic optimization using a wavelet based adaptive control vector parameterization strategy. *Comput Chem Eng* 2000; 24(2–7): 1201–1207.
17. Garcia MSG, Balsa-Canto E, Alonso AA, et al. Computing optimal operating policies for the food industry. *J Food Eng* 2006; 74(1): 13–23.
18. Cizniar M, Salhi D, Fikar M, et al. Dynopt – dynamic optimisation code for Matlab (posted 2005, accessed 2011). http://www.kirp.chtf.stuba.sk/~fikar/research/dynopt/dynopt.htm
19. Goulcher R and Casares Long JJ. The solution of steady-state chemical engineering optimisation problems using a random search algorithm. *Comput Chem Eng* 1978; 2: 33–36.
20. Banga JR, Irizarry-Rivera R and Seider WD. Stochastic optimization for optimal and model-predictive control. *Comput Chem Eng* 1998; 22(4–5): 603–612.
21. Li P, Lowe K, Arellano-Garcia H, et al. Integration of simulated annealing to a simulation tool for dynamic optimization of chemical processes. *Chem Eng Process* 2000; 39(4): 357–363.
22. Beyer HG and Schwefel HP. Evolution strategies – a comprehensive introduction. *Nat Comput* 2002; 1(1): 3–52.
23. Pham QT. Dynamic optimization of chemical engineering processes by an evolutionary method. *Comput Chem Eng* 1998; 22(7–8): 1089–1097.
24. Roubos JA, van Straten G and van Boxtel AJB. An evolutionary strategy for fed-batch bioreactor optimization: concepts and performance. *J Biotechnol* 1999; 67(2–3): 173–187.
25. Michalewicz Z, Janikow CZ and Krawczyk JB. A modified genetic algorithm for optimal control problems. *Comput Math Applic* 1992; 23(12): 83–94.
26. Dorigo M, Maniezzo V and Colorni A. Ant system: optimization by a colony of cooperating agents. *IEEE Trans Syst Man Cybern B Cybern* 1996; 26(1): 29–41.
27. Rajesh J, Gupta K, Kusumakar HS, et al. Dynamic optimization of chemical processes using ant colony framework. *Comput Chem Eng* 2001; 25(6): 583–595.
28. Zhang B, Chen D and Zhao W. Iterative ant-colony algorithm and its application to dynamic optimization of chemical process. *Comput Chem Eng* 2005; 29(10): 2078–2086.
29. Neri F, Toivanen J and Mäkinen RAE. An adaptive evolutionary algorithm with intelligent mutation local searchers for designing multidrug therapies for HIV. *Appl Intell* 2007; 27(3): 219–235.
30. Pham DT, Pham QT, Castellani M, et al. Dynamic optimisation of chemical engineering process using the bees algorithm. In: *Proceedings of 17th IFAC (international federation of automatic control) world congress*, Seoul, South Korea, 6–11 July 2008, paper 4047.
31. Pham QT. Using statistical analysis to tune an evolutionary algorithm for dynamic optimization with progressive step reduction. *Comput Chem Eng* 2007; 31(11): 1475–1483.
32. Pham QT, Pham DT and Castellani M. A modified bees algorithm and a statistics-based method for tuning its parameters. *Proc IMechE, Part I: J Systems and Control Engineering* 2012; 226(3): 287–301.
33. Ray WH. *Advanced process control*. New York: McGraw-Hill, 1981.
34. Renfro JG, Morshedi AM and Osbjornsen OA. Simultaneous optimisation and solution of systems described by differential algebraic equations. *Comput Chem Eng* 1987; 11(5): 503–517.
35. Yeo BP. A modified quasilinearization algorithm for the computation of optimal singular control. *Int J Contr* 1980; 32(4): 723–730.
36. Logsdon JS and Biegler LT. Accurate solution of differential-algebraic optimization problems. *Ind Eng Chem Res* 1989; 28(11): 1628–1639.
37. Jackson R. Optimization of chemical reactors with respect to flow configuration. *J Optim Theor Applic* 1968; 2(4): 240–259.

38. Tieu D, Cluett WR and Penlidis A. A comparison of collocation methods for solving dynamic optimization problems. *Comp Chem Eng* 1995; 19(4): 375–381.
39. Sarker RA, Mohammadian M and Yao X. *Evolutionary optimization*. Dordrecht: Kluwer Academic Publishers, 2002.
40. Bonabeau E, Dorigo M and Theraulaz G. *Swarm intelligence: from natural to artificial systems*. New York: Oxford University Press, 1999.
41. Pham DT, Ghanbarzadeh A, Koc E, et al. The bees algorithm, a novel tool for complex optimisation problems. In: *Proceedings of 2nd international virtual conference on intelligent production machines and systems (IPROMS 2006)*, 2006. Oxford: Elsevier, pp.454–459.
42. Pham DT and Castellani M. The bees algorithm – modelling foraging behaviour to solve continuous optimisation problems. *Proc IMechE, Part C: J Mechanical Engineering Science* 2009; 223(12): 2919–2938.
43. Fogel DB. *Evolutionary computation: Toward a new philosophy of machine intelligence*. 2nd ed. New York: IEEE Press, 2000.
44. Kennedy J and Eberhart R. Particle swarm optimization. In: *Proceedings of 1995 IEEE international conference on neural networks* (ed IEEE Neural Networks Council), Perth, Australia, 1995. Piscataway, NJ: IEEE Press, pp.1942–1948.
45. Seeley TD. *The wisdom of the hive: The social physiology of honey bee colonies*. Cambridge, MA: Harvard University Press, 1996.
46. Pham QT. Competitive evolution: a natural approach to operator selection. In: Yao X (ed) *Progress in evolutionary computation*. Lecture Notes in Artificial Intelligence no. 956. Berlin/Heidelberg: Springer, 1995, pp.49–60.
47. Wolpert DH and Macready WG. No free lunch theorem for optimization. *IEEE Trans Evol Comput* 1997: 1(1): 67–82.
48. Michalewicz M, Eben AE and Hinterding R. Parameter selection. In: Sarker R, Mohammadian M and Yao X (eds) *Evolutionary optimization*. Dordrecht: Kluwer Academic Press, 2002, pp.279–306.
49. De Jong KA. *An analysis of the behavior of a class of genetic adaptive systems*. PhD Thesis, University of Michigan, USA, 1975.
50. Grefenstette J. Optimization of control parameters for genetic algorithms. *IEEE Trans Syst Man Cybern* 1986; 16(1): 122–128.
51. Nannen V and Eiben AE. Relevance estimation and value calibration of evolutionary algorithm parameters. In: *Proceedings of 20th international joint conference on artificial intelligence (IJCAI)*, Hyderabad, India, 2007. AAAI Press, pp.1034–1039.
52. Smit S and Eiben A. Parameter tuning of evolutionary algorithms: generalist vs. specialist. In: Di Chio C, et al. (eds) *Applications of evolutionary computation*. Berlin/Heidelberg: Springer, 2010, pp.542–551.
53. Bartz-Beielstein T, Lasarczyk CWG and Preuss M. Sequential parameter optimization. In: *Proceedings of 2005 IEEE congress on evolutionary computation (CEC'05)*, Edinburgh, Scotland, 2005, pp.773–780.
54. Preuss M and Bartz-Beielstein T. Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms. In: Lobo FG, Lima CF and Michalewicz Z (eds) *Parameter setting in evolutionary algorithms*. Heidelberg: Springer-Verlag, 2007, pp.91–119.
55. Francois O and Lavergne C. Design of evolutionary algorithms – a statistical perspective. *IEEE Trans Evol Comput* 2001; 5(3): 129–148.
56. Rechenberg I. *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.
57. Davis L. Adapting operator probabilities in genetic algorithms. In: *Proceedings of the third international conference on genetic algorithms and their applications*, 1989. San Mateo, CA: Morgan Kaufman.
58. Whitacre JM, Pham QT and Sarker RA. Use of statistical outlier detection method in adaptive evolutionary algorithms. In: *Proceedings of the 8th annual conference on genetic and evolutionary computation (GECCO)*, Seattle, WA, 2006. New York: ACM Press, pp.1345–1352.
59. Shi Y and Eberhart R. Parameter selection in particle swarm optimization. In: *Proceedings of the 1998 annual conference on evolutionary programming, San Diego, CA, 1998*. Lecture Notes in Computer Science no. 1447. Berlin/Heidelberg: Springer, 1998, pp.591–600.
60. Roy RK. *Design of experiments using the Taguchi approach: 16 steps to product and process improvement*. New York: John Wiley and Sons Ltd, 2001.

Appendix 1

Notation

e	number of solutions chosen to survive into the next generation based on fitness alone
e_r	e/N
ev	number of evaluations per iteration
f	linearity exponent
F	objective function
g	fraction of young (less than M evolution steps) members chosen to survive into the next generation
g_r	$g/(1-e)$, surviving young members as a fraction of all young members
m	number of search steps per generation for a chosen member
M	number of search steps that define adulthood
n	number of control variables
nb	number of best sites
ne	number of elite sites
ngh	initial size of neighbourhood
nrb	recruited bees for remaining best sites
nre	recruited bees for elite sites
ns	number of scout bees
n_0	number of search steps per generation applied to the fittest member
N	population size
s	number of sampling points of each control variable, i.e. number of time steps + 1
$stlim$	limit of stagnation cycles for site abandonment
t	time
$\mathbf{u}(t)$	control vector
$\mathbf{x}$	solution vector ('bee' or 'chromosome')
$\mathbf{y}(t)$	state variable vector

Appendix 2

Dynamic optimisation benchmarks

Function 1: Continuous stirred-tank reactor (CSTR).

$$q = u_1 + u_2 + u_4$$
$$\frac{dy_1}{dt} = u_4 - q \cdot y_1 - 17.6 \cdot y_1 \cdot y_2 - 23 \cdot y_1 \cdot y_6 \cdot u_3$$
$$\frac{dy_2}{dt} = u_1 - q \cdot y_2 - 17.6 \cdot y_1 \cdot y_2 - 146 \cdot y_2 \cdot y_3$$
$$\frac{dy_3}{dt} = u_2 - q \cdot y_3 - 73 \cdot y_2 \cdot y_3$$
$$\frac{dy_4}{dt} = -q \cdot y_4 + 35.2 \cdot y_1 \cdot y_2 - 51.3 \cdot y_4 \cdot y_5$$
$$\frac{dy_5}{dt} = -q \cdot y_5 + 219 \cdot y_2 \cdot y_3 - 51.3 \cdot y_4 \cdot y_5$$
$$\frac{dy_6}{dt} = -q \cdot y_6 + 102.6 \cdot y_4 \cdot y_5 - 23 \cdot y_1 \cdot y_6 \cdot u_3$$
$$\frac{dy_7}{dt} = -q \cdot y_7 - 46 \cdot y_1 \cdot y_6 \cdot u_3$$
$$\frac{dy_8}{dt} = 5.8 \cdot (q \cdot y_1 - u_4) - 3.7 \cdot u_1 - 4.1 \cdot u_2 + q \cdot (23 \cdot y_4 + 11 \cdot y_5 + 28 \cdot y_6 + 35 \cdot y_7) - 5 \cdot u_3^2 - 0.099$$

Steps = 19
Initialisation:
$y = [0.1883, 0.2507, 0.0476, 0.0899, 0.1804, 0.1394, 0.1046, 0.0]$
Constraints:
$0 \leqslant u_1 \leqslant 20, 0 \leqslant u_1 \leqslant 6, 0 \leqslant u_1 \leqslant 4, 0 \leqslant u_1 \leqslant 20$
$0 < t < 4$
Maximise: y_8

Function 2: Batch reactor consecutive reactions.

$$\frac{dy_1}{dt} = -4000 \cdot y_1^2 \cdot e\left(-\frac{2500}{T}\right)$$
$$\frac{dy_2}{dt} = -\frac{dy_1}{dt} - 620,000 \cdot y_2 \cdot e\left(-\frac{5000}{T}\right)$$

Steps = 39
Initialisation: $y = [1.0, 0.0]$
Constraints: $298 \leqslant u \leqslant 398$
$0 < t < 1$
Maximise: y_2

Function 3: Nonlinear unconstrained mathematical system.

$$\frac{dy_1}{dt} = y_2$$
$$\frac{dy_2}{dt} = -y_3 \cdot u_1 + 16 \cdot T - 8$$
$$\frac{dy_3}{dt} = u_1$$
$$\frac{dy_4}{dt} = y_1^2 + y_2^2 + 0.0005\left(y_2 + 16 \cdot T - 8 - 0.1 \cdot y_3 \cdot u_1^2\right)^2$$

Steps = 31
Initialisation: $y = [0.0, -1.0, -\sqrt{5}, 0.0]$
Constraints: $-4 \leqslant u \leqslant 10$
$0 < t < 1$
Minimise: y_4

Function 4: Unconstrained mathematical system.

$$\frac{dy_1}{dt} = u_1$$
$$\frac{dy_2}{dt} = y_1^2 + u_1^2$$

Steps = 15
Initialisation: $y = [1.0, 0.0]$
Constraints: $-2 \leqslant u \leqslant 2$
$0 < t < 1$
Minimise: y_2

Function 5: Tubular reactor parallel reaction.

$$\frac{dy_1}{dt} = -\left(u_1 + 0.5 \cdot u_1^2\right) \cdot y_1$$
$$\frac{dy_2}{dt} = u_1 \cdot y_1$$

Steps = 19
Initialisation: $y = [1.0, 0.0]$
Constraints: $0 \leqslant u \leqslant 5$
$0 < t < 1$
Maximise: y_2

Function 6: Plug flow reactor catalyst blend.

$$\frac{dy_1}{dt} = u_1 \cdot (10 \cdot y_2 - y_1)$$
$$\frac{dy_2}{dt} = -u_1 \cdot (10 \cdot y_2 - y_1) - (1 - u_1) \cdot y_2$$

Steps = 39
Initialisation: $y = [1.0, 0.0]$
Constraints: $0 \leqslant u \leqslant 1$
$0 < t < 12$
Maximise: $1 - y_1 - y_2$

Function 7: Consecutive reaction.

$$\frac{dy_1}{dt} = -65.6 \cdot y_1 \cdot e\left(-\frac{10,000}{1.9872 \cdot (u_1 + 273)}\right)$$
$$\frac{dy_2}{dt} = 65.6 \cdot y_1 \cdot e\left(-\frac{-10,000}{1.9872 \cdot (u_1 + 273)}\right) + 1970 \cdot y_2 \cdot e\left(-\frac{-16,000}{1.9872 \cdot (u_1 + 273)}\right)$$

Steps = 19
Initialisation: $y = [1.0, 0.0]$
Constraints: $300 \leqslant u \leqslant 1000$
$0 < t < 12.5$
Maximise: y_1

Function 8: Constrained mathematical system.
Same as Function 4
Constraint: $y_1 = 1.0$