Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®
Computational Statistics & Data Analysis 44 (2003) 377–390
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

A parallel algorithm for semi-infinite programming

S. Žaković[a,*], B. Rustem[a], S.P. Asprey[b,1]

[a] *Department of Computing, Imperial College, London, SW7 2BZ, UK*
[b] *Centre for Process Systems Engineering, Imperial College, London, SW7 2BY, UK*

Received 1 July 2002

Abstract

The implementation results of an algorithm designed to solve semi-infinite programming problems are reported. Due to its computational intensity, parallelisation is used in two stages of the algorithm—for evaluating the global optimum and for checking the feasibility of constraints. The algorithms are parallelised using MPI—the message passing interface. The algorithms are then applied to engineering and macroeconomic modelling problems, including designing robust optimally informative experiments for dynamic model identification and verification, and robust macroeconomic policies for inflation targeting.

1. Introduction

A parallel implementation of the solution of the following semi-infinite problem is considered:

$$\begin{aligned} \min_{x\in X} \quad & f(x) \\ s.t. \quad & G(x,y) \leqslant 0, \quad \forall y \in Y \\ & X = \{x \in \mathbf{R}^n | g_i(x) \leqslant 0,\ i = 1,\ldots,k\} \\ & Y = \{y \in \mathbf{R}^m | q_i(y) \leqslant 0,\ i = 1,\ldots,l\} \\ & G : \mathbf{R}^{n+m} \to \mathbf{R}^{n_G}, \end{aligned} \tag{1}$$

* Corresponding author. Tel.: +44-20-7594-8272; fax: +44-20-7581-8024.
E-mail addresses: s.zakovic@ic.ac.uk (S. Žaković), b.rustem@ic.ac.uk (B. Rustem), s.asprey@ic.ac.uk (S.P. Asprey).
[1] Present address: Orbis Investment Advisory Ltd, 5 Mansfield St, London W1G 9NG.

0167-9473/$ - see front matter
doi:10.1016/S0167-9473(03)00152-X

where X and Y are nonempty compact subsets of $\mathbf{R}^n$ and $\mathbf{R}^m$, n_G is the number of coupled constraints $G_j(x, y)$ $(j = 1, \ldots, n_G)$ and f and G are twice continuously differentiable on X and $X \times Y$, respectively. The term semi-infinite programming arises from the fact that $G(x, y) \leqslant 0,\ \forall y \in Y$ represents an infinite set of constraints on x.

Problem (1) was first considered by John (1948) who gave necessary and sufficient conditions for its solution. Since then, it has been extensively studied in the literature. Other efforts include Blankenship and Falk (1976), Coope and Watson (1985), Polak and Mayne (1976), Mayne and Polak (1982), Gustafson (1981) and, more recently, Lawrence and Tits (1998). The reader is referred to Hettich and Kortanek (1993) for further references.

Problem (1) can be used to formulate certain minimax and saddle point problems arising in optimal engineering design under parametric uncertainty. Applications to computer-aided design and parametric optimal decisions are presented in Kwak and Haug (1976) and Mayne et al. (1979). In such problems, $f(x)$ may represent the cost function where x is a vector of control/design variables and y is a vector of uncertain parameters. The formulation ensures that the resulting design will fulfil the constraints $G(x, y) \leqslant 0$ for all $y \in Y$, usually related to the inherent safety of the system. Worst-case design is essentially a minimax formulation when the best decision is taken in view of the worst-case scenario. This can be formulated as a semi-infinite program. It has the advantage that the decision is robust to uncertainty. Robustness is assured as system performance is optimal for the worst-case scenario and will be improved if any other scenario is realised (Rustem and Howe, 2001; Zakovic and Rustem, to appear).

In this paper, (1) is solved using the algorithm based on the work of Blankenship and Falk (1976) and Darlington et al. (1999), where a two-phase procedure to guarantee feasibility for the general nonlinear constraints was employed. The algorithm described in Darlington et al. (1999) is specialised towards constraint satisfaction in view of the worst-case realization of the uncertainties. This is further discussed in (Zakovic and Rustem, to appear) where the problem is solved using a two stage procedure that searches for global maximum violation of the constraints, together with a version of the algorithm that searches for any violation of constraints.

The main difficulty of the algorithm is the computational effort required. This is mostly in the computation of the global optimisation and the checking of constraint violation. In the present study, a parallel scheme is provided to address these difficulties and the advantages are illustrated using numerical examples.

The algorithm begins at $k = 0$ with a finite subset $Y_k \subset Y$, solving the approximate problem:

$$v_k = \min_{x \in F_k} f(x), \quad F_k = \{x \in X | G(x, y) \leqslant 0,\ \forall y \in Y_k\}. \tag{2}$$

F and v^* are defined as

$$F = \{x \in X | G(x, y) \leqslant 0,\ \forall y \in Y\},$$

$$v^* = \min_{x \in F} f(x).$$

The algorithm solves the kth auxiliary global optimisation problem—compute $y_{\max} \in Y$ such that

$$G(x_k, y_{\max}) \geqslant G(x_k, y), \quad \forall y \in Y, \tag{3}$$

where $x_k = \arg\min_{x \in F_k} f(x)$. Termination is declared when the inequality

$$G(x_k, y_{\max}) \leqslant 0$$

is satisfied. Otherwise, the algorithm sets

$$Y_{k+1} = Y_k + \{y_{\max}\}, \quad k = k + 1,$$

and solves (2) again.

Note that

$$F \subset \cdots \subset F_{k+1} \subset F_k \subset \cdots \subset F_0,$$

implies

$$v^* \geqslant \cdots \geqslant v_{k+1} \geqslant v_k \geqslant \cdots \geqslant v_0.$$

To justify the stopping rule, observe that the solution of the kth auxiliary problem satisfies

$$G(x_k, y_{\max}) \leqslant 0,$$

then

$$G(x_k, y) \leqslant 0, \quad \forall y \in Y.$$

Therefore

$$v_k \leqslant v^* \leqslant f(x_k) = v_k,$$

so that x_k is a solution of (2), with $F_k = F$.

1.1. Global optimisation

Algorithms for global optimisation problems are very complex in nature as they usually involve a large number of iterations, functions evaluations and local searches. Dixon and Szego (1975, 1978) distinguish two basic approaches—deterministic and stochastic. Most of the methods to solve the global optimisation problem can be placed in these two categories.

Deterministic methods find the global minimum by locating all the minima of the objective function. The global minimum is then the observed local minimum with the lowest function value. Methods like grid search (space covering), trajectory methods or interval arithmetic methods (i.e Hansen, 1992; Wolfe, 1996) are classified as deterministic methods, because, for a given initial point, they always find the same solution. Such methods usually involve additional assumptions on f. Interval arithmetic methods can in principle solve the global optimisation problem within some desired error bound (Hansen, 1992; Moore, 1979). Methods like random search (Deep and Evans, 1994)

and clustering (Rinnooy Kan and Timmer, 1987) are classified as stochastic, as they are based on random sampling in the region of interest and given an initial point may find a different solution if the run is repeated.

Recent years, especially as the required computer power has become available, have seen rapid development developments in global optimisation algorithms. Most notably among these are, to name only a few are those of Floudas et al. Adjiman et al. (1996, 1998), Smith and Pantelides (1997a, b), or a paper on parallel search global optimisation algorithms by Pardalos (1989). A good survey on recent development and trends in the area can be found in Pardalos et al. (2000).

For global optimisation in (3), a stochastic algorithm, developed by Rinnooy et al. (1987) is used. It is a well known algorithm and has been used, among other things, for solving complex real life problems arising in laser diffractometry (Zakovic, 1997; Zakovic et al., 1998)

The algorithm starts independent local optimisations from numerous different points, and keeps a list ($X_{\max}$) of maxima encountered. As a stopping rule, it uses a Bayesian estimate of the total number of maxima, based on the numbers of local optimisations performed, and local solutions found. Also $x_{\max}(x_{\text{st}}) = \arg\max_x\{f(x)\}$ in step 4 is defined as a local maximum of $f(x)$, obtained when the local optimisation was started from x_{st}. Algorithm 1 summarises Rinnooy Kan and Timmer's method.

Algorithm 1 Global Optimisation

1: $X_{\max} = \emptyset,\ n_{\text{loc}} = 0,\ n_{\max} = 0$
2: **repeat**
3: Choose a random starting point x_{st}
4: $x_{\max}(x_{\text{st}}) = \arg\max_x\{f(x)\}$
5: $n_{\text{loc}} = n_{\text{loc}} + 1$
6: **if not**$(x_{\max}(x_{\text{st}}) \in X_{\max}))$ **then**
7: $X_{\max} = X_{\max} \cup x_{\max}(x_{\text{st}})$
8: $n_{\max} = n_{\max} + 1$
9: **end if**
10: $n_{\text{tot}} = \dfrac{n_{\max}(n_{\text{loc}} - 1)}{n_{\text{loc}} - n_{\max} - 2}$
11: **until** $\{(n_{\text{tot}} \leqslant n_{\max} + 0.5).or.(f(x_{\max}(x_{\text{st}})) \geqslant 0)\}$
12: $x_{\text{glob}} = \{x_{\text{glob}} \in X_{\max} | f(x_{\text{glob}}) \geqslant f(x),\ \forall x \in X_{\max}\}$

In step 10 there is a possibility that $n_{\text{loc}} = n_{\max} + 2$. In such the case n_{tot} is not computed, but another local optimisation is performed. If new maximum is not encountered then $n_{\text{loc}} \neq n_{\max} + 2$, and n_{tot} is well defined. However if there is a new maximum, meaning that still $n_{\text{loc}} = n_{\max} + 2$, the whole procedure is repeated until there has been performed a local optimisation which did not encounter new maximum.

The condition $f(x_{\max}(x_{\text{st}})) \geqslant 0$ in step 11 is not part of the original Rinnooy Kan and Timmer algorithm. The price of computing the global optimum can be, and usually is, very high. As the search for points which violate constraints is performed, it may well be computationally cheaper to stop the global optimisation search at any point where

violation is encountered. This approach can bring benefits regarding the computational time, especially when functions that are being globally optimised are very complex and nonconvex Zakovic and Rustem (to appear). There may be the setback, however, that by including local violations of constraints, the nonlinear programming problem in (2) is solved more times than necessary. A switch is built into the proposed algorithm, which allows the user to decide, depending on the complexity of the functions $f(x)$ and $G(x, y)$, which version of the algorithm to use.

The constrained nonlinear programming problems that appear in (2), and in Algorithm 1, are solved using standard nonlinear programming packages; in this case the NAG subroutine e04ucf.[2] The subroutine computes the minimum of a general smooth function that may include simple bounds as well as linear and nonlinear constraints.

Pseudo code for solving (1) is summarised in Algorithm 2.

Algorithm 2 Semi-infinite Programming

1: $k = 0,\ Y_0 = \{y_0\},\ F_k = \{x \in X | G(x, y) \leqslant 0,\ \forall y \in Y_k\}$
2: OK = true
3: **while** (NOT OK) **do**
4: $x_k = \arg\min_x \{f(x) | x \in F_k\}$
5: **if** infeasible **then**
6: **STOP**: original problem infeasible
7: **end if**
8: OK = **true**, $\hat{Y} = \emptyset$
9: **for** $j = 1, n_G$ **do**
10: $y_k^j = \arg\max_{y \in Y} \{G_j(x_k, y)\}$ (global)
11: **if** $\{G_j(x_k, y_k^j) \geqslant 0\}$ **then**
12: OK = **false**
13: $\hat{Y} = \hat{Y} \cup \{y_k^j\}$
14: **end if**
15: **end for**
16: $Y_{k+1} = Y_k \cup \hat{Y}$
17: $k = k + 1$
18 **end while**
19: **optimal solution found**
20: **stop**

The feasibility check in step 11 depends on the nature of the constraints, whether they are hard or soft, and on the application being considered. Hard constraints indicate situations where no constraint violation can be tolerated by the physical system. This requires feasibility for all realizations of y, which may further increase the price of computation. If strict feasibility cannot be obtained or guaranteed, a penalty approach may be used, allowing optimal solutions with the smallest possible violation of the

[2] N.A.G., Library, "Subroutine E04UCF.", http://www.nag.co.uk/numeric/FLOLCH/mk18/e/E04XAF.html.

constraints $G(x, y)$ within Y. This approach is further discussed in Darlington et al. (1999).

1.2. The message passing interface—MPI

As can be seen, the algorithms used to solve the semi-infinite programming problem are computationally intensive. In order to accelerate their solution, parallelisation techniques are used; in particular, the proposed algorithms are parallelised using the Message Passing Interface (MPI). MPI is a specification for a library of routines to be called from C or Fortran programs, and was first developed in 1993 when a group of computer vendors, software writers, computational scientists, and others collaborated on setting a standard portable message passing library.

Parallel computers have recently become an everyday tool of computational scientists, and have widespread applications in optimisation and mathematical programming (Pardalos et al., 1992; Pardalos, 1999; Migdalas et al., 1997). However, there are still problems that prevent the widespread use of parallelism, both in terms of hardware and software. For instance, for hardware, it is still difficult to build intercommunication networks that can keep up with the speed of most advanced single processors. For software, compilers that automatically parallelise sequential algorithms are very limited in their applicability.

Parallelisation brings up some issues that are, obviously, not present in a serial context. Most important among those issues are task allocation, where the breakdown of the total workload into smaller tasks is performed so that such tasks are allocated to different processors. Another issue is communication, of intermediate results of different processors. Also among these important issues is synchronisation of the different computations processes.

MPI's design and development has been guided not only by the above issues but by the need for portability and to impose no semantic restrictions on efficiency. It has also been designed to be a convenient and complete definition of the message passing model (Gropp et al., 1994). The message passing model consists of processes that have only local memory, but are able to communicate with other processes by sending and receiving messages. In the message passing model, data transfer from the local memory of one process to the local memory of another process requires operations to be performed by both processes.

Although it cannot be said that the message passing model is superior to any other parallel methods, it has become widely used for a number of reasons, such as universality, expressivity, ease of debugging, performance, to name but few (Gropp et al., 1994). Efficiency is one reason too, for instance, some Crays have hardware barriers. Although there are about 125 functions in the MPI standard, there are only 6 functions that are indispensable, upon which a large number of efficient programs can be written. All the other functions add flexibility, robustness or convenience.

Fig. 1 illustrates task farming which is a communication pattern that can be implemented using message passing. The master process, enumerated Process 0, sends data to the slave processes, enumerated 1 to $n-1$. After the data is processed it is sent back to the master process.

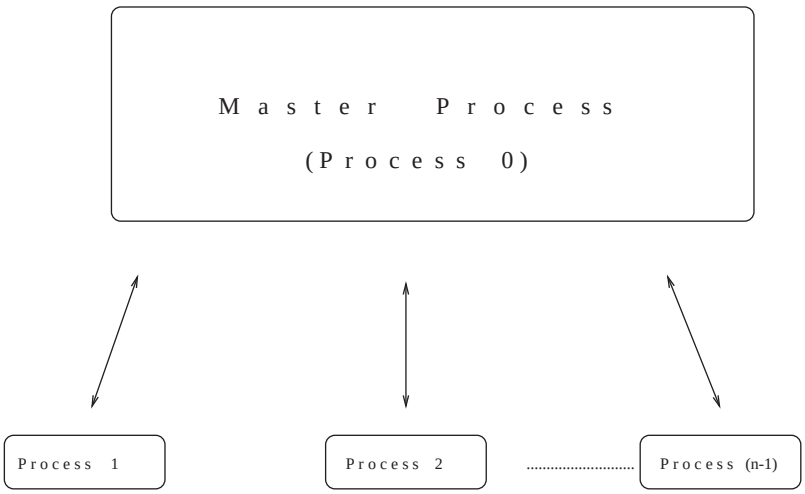

Fig. 1. The task farming model.

1.3. Parallelising semi-infinite programs

There are two sub-procedures in the full SIP algorithm that can be parallelised, namely the global optimisation in step 10, of Algorithm 2 and checking for violation of the constraints in step 11.

It is worth mentioning that stochastic approaches are becoming very important tools in a wide range of fields, starting from physics, chemistry or applied mathematics to social sciences. Bearing this in mind it is clear that an important aspect of such approaches is the source of pseudo random numbers. A scalable package for parallel pseudo random number generation (SPRNG) is being developed at Florida State University.[3] In this implementation however, a generator from NAG library (see footnote 2) is used, as the number of starting points generated is relatively small.

In the case of global optimisation, parallelisation is implemented as follows—the master process randomly chooses N_{st} starting points x_{st} and subsequently distributes them evenly ($\approx N_{st}/n-1$ points) to each of the remaining $n-1$ processors. Local optimisations are performed by each processor, each starting from one of the points in the starting point set x_{st}. The results are then sent back to the master process, where it is checked whether the newly obtained locally optimal points are disregarded or put in the recorded list of local minima.

A similar procedure is followed for the constraint violation problem. There are n_G constraints present, where each can be sent to a different process, which subsequently returns the information about the violation. Then, again, the master process decides whether to add a new constraint to the feasible region.

[3] Scalable Parallel Pseudo Random Number Generators Library, http://sprng.cs.fsu.edu/.

2. Applications

To illustrate the application of the algorithms discussed in this paper, two examples are presented, one from macroeconomics and the other from engineering, detailing and discussing their solutions.

2.1. An application in macroeconomics

In a recent paper, Orphanides and Wieland (2000) use a simple macroeconomics model of inflation, output gap and interest rates for inflation targeting. The policy instrument is the short term nominal interest rate. The dynamic structure of the model is represented by a single lag of inflation in the Phillips curve and a single lag of the output gap in the aggregate demand equation. It is appropriate, therefore, to interpret the length of a single period to be rather long, say half a year to a year.

In every period, the policymaker sets the interest rate, r_t. To describe the policymaker's welfare loss during a period t, a per-period loss function is specified as a weighted average of the deviation of inflation π from its desired target π^* and the output deviation from the economy's natural level y.

$$l_t = l(\pi_t, y_t) = \omega(\pi_t - \pi^*)^2 + (1-\omega)y_t^2, \quad \omega \in (0,1).$$

The state variables y_t and π_t, are defined by the linear Phillips curve:

$$\begin{aligned} y_t &= \delta + \rho y_{t-1} - \xi r_{t-1} + u_t, \\ \pi_t &= \pi_{t-1} + \alpha y_t + e_t, \end{aligned} \tag{4}$$

where $\delta, \alpha, \xi > 0$, $\rho \in [0,1)$ and e_t, u_t are zero-mean, noncorrelated random shocks.

An alternative approach, which will be used in this framework, is that of Tetlow and von zur Muehlen (2001). In that approach (see also Hansen et al. (1999)) the policymaker chooses the parameters x_1 and x_2 of the feedback law $r_t = x_1\pi_t + x_2 y_t$ to minimise welfare losses that are maximised over w_t. This rule is referred to as the feedback rule.

The objective function is defined in terms of a sum of per-period losses:

$$f(x_1, x_2, u, e) = \sum_{s=0}^{\infty} \beta^s l_{t+1+s}, \quad \beta \in (0,1),$$

where β is the discount factor.

In the worst-case approach, the aim is to minimise the discounted sum of future per-period losses with respect to the worst possible outcome of the uncertain variables u and e. Therefore, the policymaker chooses x_1, x_2 to minimise the sum of losses, where "nature" chooses u, e to maximise the sum.

Parameters α, ρ, and ξ that appear in the state equations (4) are usually estimated from the existing data (see Orphanides and Wieland, 2000); uncertainty can also be introduced on them, usually in the form of a posteriori probability distribution functions.

Therefore the most general optimisation problem can be defined as:

$$\begin{aligned} \min_{x_1,x_2} \quad & \max_{u,e,\alpha,\rho,\xi} \quad f(x_1,x_2,u,e,\alpha,\rho,\xi), \\ & s.t. \qquad l \leqslant u_t, e_t \leqslant u, \\ & \qquad\quad \underline{\alpha} \leqslant \alpha \leqslant \bar{\alpha}, \\ & \qquad\quad \underline{\rho} \leqslant \rho \leqslant \bar{\rho}, \quad \underline{\xi} \leqslant \xi \leqslant \bar{\xi}. \end{aligned}$$

Results for the following two problems are presented

$$\begin{aligned} \min_{x_1,x_2} \quad & \max_{u,e} f(x_1,x_2,u,e) \\ & -0.05 \leqslant u_t, \quad e_t \leqslant 0.05. \\ & x_1^* = 1.30, \quad x_2^* = 1.14. \end{aligned} \tag{5}$$

and

$$\begin{aligned} \min_{x_1,x_2} \quad & \max_{u,e} f(x_1,x_2,u,e) \\ & -0.05 \leqslant u_t, e_t \leqslant 0.05, \quad 0.21 \leqslant \alpha \leqslant 0.47 \\ & 0.66 \leqslant \rho \leqslant 0.88, \quad 0.3 \leqslant \xi \leqslant 0.5. \\ & x_1^* = 1.26, \quad x_2^* = 1.75. \end{aligned} \tag{6}$$

In both examples, three worst cases were found. Each of the worst cases actually represent possible outcomes of a different scenario. However, for all of the possible worst-case scenarios, the feedback rule is the same

$$r_t^* = x_1^* \pi_t + x_2^* y_t.$$

Table 1 contains CPU times and acceleration obtained due to parallelisation with an increasing number of processors for solving problems (5) and (6).

The acceleration is significant, considering the complexity of the problems presented here. As such, communication time does not play an important role for problems (5) and (6). The best acceleration is achieved for problem (5), where computational time has been reduced from almost 42 h to less than 4 h.

Table 1
Numerical solutions and speed-ups for problems (5) and (6)

Problem		Number of processors			
		1	4	8	16
Macro-1	CPU(h)	41.91	16.00	7.47	3.71
	speed-up	—	2.62	5.61	11.29
Macro-2	CPU(h)	7.75	2.66	1.28	0.71
	speed-up	—	2.92	6.07	10.91

2.2. Robust optimal design of dynamic experiments

To obtain predictive mathematical models of processing systems, one is faced with the problem of having to estimate several freely-varying parameters within the model from collected experimental data in order to statistically verify the model. In particular, systems modelled by general systems of mixed differential and algebraic equations are considered

$$f(x, \dot{x}, y, \theta, \phi, t) = 0,$$

$$g(x, y, t) = 0.$$

Due to the nonlinearity of these models, the ease at which verification can be accomplished not only depends on the parameterisation of the model (i.e., the mathematical form of the model), but also on "where" in the experiment space the data have been collected. The main question that arises when designing optimally informative experiments is—how should time-varying controls, initial conditions or the duration of the experiment be adjusted to generate the maximum amount of information for parameter identification and ultimately model verification?

The predicted amount of information contained within a set of experimental data can be used to design future experiments that are optimally rich in information for parameter estimation purposes. To quantify this, an information matrix for dynamic experiment design is defined as

$$M_I(\theta, \phi) \equiv \sum_{r=1}^{M} \sum_{s=1}^{M} \tilde{\sigma}_1^{rs} Q_r^{\mathrm{T}} Q_s,$$

where ϕ is a vector of experiment decision variables (i.e., sampling times of response variables to be predicted by the model, time-varying controls to be applied to the process, etc.), $\tilde{\sigma}_1^{rs}$ is the rsth element of the inverse of the estimate of the variance–covariance matrix $\tilde{\sum}_y = cov(y_r, y_s)$ and θ is the vector of model parameters to be estimated from data. The $(n_{sp} \times p)$ matrix Q_r is the matrix of first-order dynamic sensitivity coefficients of the rth response variable in the model computed at each of n_{sp} sampling points (the number of which is chosen a priori):

$$Q_r \equiv \begin{bmatrix} \dfrac{\partial y_r(\theta, \phi, t_1)}{\partial \theta_1} & \cdots & \dfrac{\partial y_r(\theta, \phi, t_1)}{\partial \theta_p} \\ \dfrac{\partial y_r(\theta, \phi, t_{n_{sp}})}{\partial \theta_1} & \cdots & \dfrac{\partial y_r(\theta, \phi, t_{n_{sp}})}{\partial \theta_p} \end{bmatrix}.$$

To design future experiments in the face of uncertainty in the parameters, θ, the following max-min optimisation problem is solved:

$$\phi_R = \arg\max_{\phi \in \Phi} \left\{ \min_{\theta \in \Theta} \{\det(M_I(\theta, \phi))\} \right\},$$

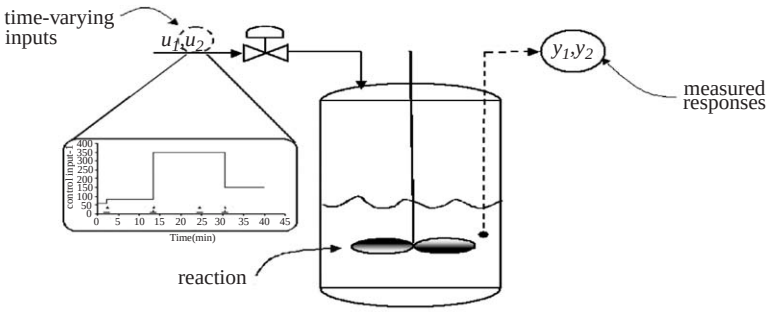

Fig. 2. A fed-batch reactor.

which may be rewritten in semi-infinite form as

$$\max_{\phi\in\Phi,\Psi\in R} \quad \Psi$$

$$s.t. \quad \Psi \leqslant \det(M_I(\theta,\phi)), \quad \forall\theta\in\Theta. \tag{7}$$

$\Phi \subset \mathbf{R}^n$ and $\Theta \subset \mathbf{R}^p$ represent feasible regions (upper and lower bounds on each element of ϕ and θ, respectively), and thus there are no functional constraints on the variables. In a typical situation, the dimension n of Φ is much larger than the dimension p of Θ, indicating that the problem of maximising Ψ is much more complex and computationally intensive.

A typical example is presented to illustrate the application of parametrically robust optimal experiment design. Consider a fed-batch reactor in which the fermentation of baker's yeast is carried out. To model this process, the following model $M_I(\theta,\mathbf{u},t)$ is proposed:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = (r - u_1 - \theta_4)y_1$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = -\frac{ry_1}{\theta_3} + u_1(u_2 - y_2)$$

$$r = \frac{\theta_1 y_2}{\theta_2 + y_2}.$$

Within this process, there are two time-varying controls $(u_1(t), u_2(t))$ and two measured concentrations $(y_1(t), y_2(t))$, as illustrated in Fig. 2.

Here, $\Phi \subset \mathbf{R}^{29}$—one varying initial condition, ten sampling times, and two time-varying inputs parameterised by piecewise constant trajectories, each delineated by five time intervals (representing 18 variables); and $\Theta \subset \mathbf{R}^4$ as there are four model parameters to be estimated from experimental data. In problems such as these, it seems rather obvious that the use of parallelisation will not gain much improvement, as the most expensive computation is that of the **max** problem, which is performed sequentially.

In Table 2 the solution to (7) for the model describing the semi-batch fermentation of baker's yeast (Asprey and Macchietto, 2000; Asprey et al., 2000) is shown. There

Table 2
Intermediate solutions when solving problem (7)

Iteration		Results	No. of processors 1	4	8	16
	max	39.38954				
I	min	−51.78269	1517 s	1365 s	1320 s	1303 s
	max	4.57472				
II	min	−6.73966	1677 s	1543 s	1508 s	1488 s
	max	4.48416				
III	min	−4.07338	1864 s	1640 s	1574 s	1548 s
	max	4.47273				
IV	min	0.00001	2564 s	2179 s	2067 s	2021 s
Total time (seq)			6221 s	6221 s	6221 s	6221 s
Total time (par)			1401 s	506 s	248 s	139 s
Speed-up (par)			—	2.77	5.76	10.08
Total time (seq+par)			7622 s	6727 s	6469 s	6360 s

are two timings shown—one for the sequential part, where only the **max** problem is solved and then timings obtained for solving the global optimisation problem. Although overall improvements are not significant (only about 15% for 16 processors), speed-ups obtained for the parallel computation are comparable to those presented in Table 1.

The interested reader is referred to Asprey and Macchietto (2002) for an in-depth comparison of using the semi-infinite programming approach to the robust design of dynamic experiments to that of using point estimates of the parameters alone, as well as an expected-value approach using a priori probability distributions for the uncertainty associated with each of the model parameters.

3. Conclusions

A parallelised semi-infinite programming algorithm has been presented in this paper. Two different sub-procedures—global optimisation and the checking of constraint violation have been parallelised. The parallelisation has been effected within the MPI (Message Passing Interface) framework (Gropp et al., 1994). The hardware used is Fujitsu AP3000 parallel server—a distributed memory parallel server based on powerful 64-bit UltraSPARC technology. The nodes of the AP3000 are connected by Fujitsu's AP-Net. This is a wormhole routed network connecting the processing nodes in a 2-D Torus configuration. The data transfer of the network is 200 MB/s × 2 (bi-directional) for each port (4 ports/node). ICPC's AP3000 has the configuration of 60 UltraSPARC 300 MHz processors (U300), local file space per node: 4GB, 15.5 GB main memory (varying between 128, 256 and 512 Mb).

Two complex problems from macro-economics and the optimal design of dynamic experiments are presented to illustrate the efficiency of the implementation of the semi-infinite programming problem. In these cases, the decrease in computational times

are very significant, as these problems depend on the actual computational time, rather than the time spent on communication between the processing nodes. The paper presents a promising start to accelerating the solution of challenging economic and engineering problems. Indeed, the experiment design problem can then be applied to very large problems such as the dynamic modelling of mammalian cells in culture; models comprising some 150 parameters and a DAE system of 200 equations.

References

Adjiman, C.S., Androulakis, I.P., Floudas, C.A., 1996. A global optimization method αbb, for process design. Comp. Chem. Eng. 20, S419–S424.

Adjiman, C.S., Androulakis, I.P., Floudas, C.A., 1998. A global optimization method αbb, for general twice-differentiable constrained NLPS. Comp. Chem. Eng. 22, 1159–1179.

Asprey, S.P., Macchietto, S., 2000. Statistical tools for optimal dynamic model building. Comp. Chem. Eng. 24, 1261–1267.

Asprey, S.P., Macchietto, S., 2002. Designing robust optimal dynamic experiments. J. Proc. Cont. 12, 545–556.

Asprey, S.P., Macchietto, S., Pantelides, C.C., 2000. In: Biegler, L.T., Brambilla, A., Scali, C. (Eds.), Proceedings of ADCHEM'2000, Vol. II, pp. 869–875 IFAC.

Blankenship, J.W., Falk, J.E., 1976. Infinitely constrained optimization problems. JOTA 19, 261–281.

Coope, I.D., Watson, G.A., 1985. A projected Lagrangian Algorithm for semi-infinite programming. Math. Programming 32, 337–356.

Darlington, J., Pantelides, C.C., Rustem, B., Tanyi, B.A., 1999. An algorithm for constrained nonlinear optimization under uncertainty. Automatica 35, 217–228.

Deep, K., Evans, D.J., 1994. A parallel random search global optimisation method, Computer Studies 882, Loughborough University of Technology.

Dixon, L.C.W., Szego, G.P. (Eds.), 1975. Towards Global Optimisation, North Holland, Amsterdam.

Dixon, L.C.W., Szego, G.P. (Eds.), 1978. Towards Global Optimisation, Vol. 2, North Holland, Amsterdam.

Gropp, W., Lusk, E., Skjellum, A., 1994. Using MPI. The MIT Press, Cambridge, Massachusetts.

Gustafson, S.A., 1981. A three-phase algorithm for semi-infinite programming. In: Fiacco, A.V., Kortanek, O. (Eds.), Semi-infinite Programming and Applications, Lecture Notes in Economics and Mathematical Systems, Vol. 215. Springer, Berlin, pp. 138–157.

Hansen, E., 1992. Global Optimisation Using Interval Analysis. Marcel Dekker, Inc., New York.

Hansen, L., Sargent, T.J., Tallarini, T.D., 1999. Robust permanent income and pricing. Rev. Econom. Stud. 66, 873–907.

Hettich, R., Kortanek, K.O., 1993. Semi-infinite Programming: Theory, Methods and Applications. SIAM Rev. 35, 380–429.

John, F., 1948. Extremum Problems With Inequalities as Subsidiary Condition, Studies and Essays, Courant Anniversary Volume. Wiley, New York.

Kwak, B.M., Haug, E.J., 1976. Optimum design in the presence of parametric uncertainty. JOTA 19, 527–546.

Lawrence, C.T., Tits, A.L., 1998. Feasible Sequential Quadratic Programming for Finely Discretized Problems from SIP. In: Reemtsen, R., Ruckmann, J.J. (Eds.), Semi-infinite Programming. Kluwer Academic Publishers, Dordrecht, pp. 159–193.

Mayne, D.Q., Polak, E., 1982. A quadratically convergent algorithm for solving infinite dimensional inequalities. Appl. Math. Optim. 9, 25–40.

Mayne, D.Q., Polak, E., Trahan, R., 1979. An outer approximation algorithm for computer-aided design problems. JOTA 28, 331–352.

Migdalas, A., Pardalos, P.M., Storoy, S. (Eds.), 1997. Parallel Computing in Optimization, Kluwer Academic Publishers, Dordrecht.

Moore, R.E., 1979. Methods and Applications of Interval Analysis. SIAM, Philadelphia.

Orphanides, A., Wieland, V., 2000. Inflation zone targeting. European Econom. Rev. 44, 1351–1387.
Pardalos, P.M., 1989. Parallel search algorithms in global optimization. Appl. Math. Comput. 29 (3), 219–229.
Pardalos, P.M., 1999. Parallel Processing of Discrete Problems, IMA Volumes in Mathematics and its Applications, Vol. 106, Springer, Berlin.
Pardalos, P.M., Philips, A., Rosen, J.B., 1992. Topics in Parallel Computing in Mathematical Programming. American Mathematical Society & Science Press, Providence, RI, Moscow.
Pardalos, P.M., Romeijn, H.E., Tuy, H., 2000. Recent developments and trends in global optimization. J. Comp. Appl. Math. 124, 209–228.
Polak, E., Mayne, D.Q., 1976. An algorithm for optimization problems with functional inequality constraints. IEEE Trans. Automat. Control AC-21, 184–193.
Rinnooy Kan, A., Timmer, G.T., 1987. Stochastic global optimization methods. Part II: Multilevel methods. Math. Programming 78, 39–57.
Rustem, B., Howe, M.A., 2001. Algorithms for Worst-case Design with Applications to Risk Management. Princeton University Press, Princeton, NJ.
Smith, P., Pantelides, C., 1997a. Global optimisation on nonconvex MINLPS. Comp. Chem. Eng. 21, S791–S796.
Smith, P., Pantelides, C., 1997b. A symbolic reformulation spatial branch-and-bound algorithms for the global optimisation of nonconvex MINLPS. Comp. Chem. Eng. 23, 457–478.
Tetlow, R.J., von zur Muehlen, P., 2001. Robust monetary policy with misspecified models: Does model uncertainty always call for attenuated policy?. J. Econom. Dynamics Control 25, 911–949.
Wolfe, M.A., 1996. Interval methods for global optimisation. Appl. Math. Comput. 75, 179–206.
Zakovic, S., 1997. Global optimization applied to an inverse light scattering problem, Ph.D. Thesis, University of Hertfordshire, UK.
Zakovic, S., Rustem, B. to appear. Semi-infinite programming and applications to minimax problems. Ann. Oper. Res.
Zakovic, S., Ulanowski, Z., Bartholomew-Biggs, M.C., 1998. Application of global optimization to particle identification using light scattering. Inverse Problems 14 (4), 1053–1067.