NUMERICAL EXPERIMENTS IN SEMI-INFINITE PROGRAMMING

by

C.J. Price and I.D. Coope
Department of Mathematics and Statistics,
University of Canterbury, Christchurch, New Zealand

No. 91 June, 1993

Abstract – A quasi-Newton algorithm for semi-infinite programming using an L_∞ exact penalty function is described, and numerical results are presented. Comparisons with three Newton algorithms and one other quasi-Newton algorithm show that the algorithm is very promising in practice.

AMS classifications: 65K05,90C30.

Keywords: semi-infinite programming, nonlinear optimisation, L_∞ exact penalty function.

Numerical Experiments in Semi-Infinite Programming

C. J. Price and I. D. Coope,
Department of Mathematics and Statistics,
University of Canterbury, Private Bag 4800,
Christchurch 8001, New Zealand.

Abstract

A quasi-Newton algorithm for semi-infinite programming using an L_∞ exact penalty function is described, and numerical results are presented. Comparisons with three Newton algorithms and one other quasi-Newton algorithm show that the algorithm is very promising in practice.

AMS classifications: 65K05,90C30.

Keywords: semi-infinite programming, nonlinear optimisation, L_∞ exact penalty function.

1 Introduction

Semi-Infinite Programming (SIP) problems occur in a wide variety of fields, such as computer aided design, and pollution control. Several globally convergent schemes for solving SIP problems have been proposed [1, 2, 5, 10, 11, 12]. A common approach yielding global convergence is the use of sequential quadratic programming techniques in conjunction with an Exact Penalty Function (EPF). It is shown by Tanaka et al [11] (see also [10] for a graphical example) that, in the context of semi-infinite programming, the L_∞ EPF is preferable to the L_1 EPF. In [10] it is shown that the theoretical results for an algorithm based on the L_∞ EPF are applicable to C^1 problems; in contrast, for the L_1 EPF some functions must be C^2. This paper presents the results of numerical experiments with the quasi-Newton algorithm for SIP described in [9, 10]. The theoretical properties of this algorithm are discussed in [9, 10].

The problem considered herein is:

$$\min_{x \in R^n} f(x) \quad \text{subject to} \tag{1}$$

$$g(x,t) \leq 0 \ \ \forall\, t \in T. \tag{2}$$

Here $f(x)$ and $g(x,t)$ are continuously differentiable functions mapping $R^n \rightarrow R$ and $R^n \times T \rightarrow R$ respectively, and $T \subset R^p$ is a Cartesian product of closed intervals. For convenience only one semi-infinite constraint (0.2) has been considered, and auxiliary finite constraints have been omitted. The algorithm is applicable to problems with finite numbers of semi-infinite and ordinary constraints.

Rather than solve the SIP directly, the problem of minimising an exact penalty function $\phi(x)$ over $x \in R^n$ is solved, where ϕ is chosen so that the solution points of the SIP coincide with those of the Penalty Function Problem (PFP). The penalty function used is

$$\phi(\mu, \nu; x) = f(x) + \mu\theta + \tfrac{1}{2}\nu\theta^2 \quad \text{where} \quad \theta = \max_{t \in T} [g(x,t)]_+ . \tag{3}$$

The penalty parameters μ and ν are restricted to $\mu > 0$, and $\nu \geq 0$. Clearly $\theta(x)$ is the infinity norm of the constraint violations, hence ϕ is continuous $\forall x \in R^n$.

As f and g are only required to be C^1, the problem of finding a local minimum of the SIP is replaced by that of finding a stationary point.

Definition 1.1 *Let $x^* \in R^n$ satisfy the constraint (0.2), and let there exist $t_1, \ldots, t_m \in T$ and non-negative numbers $\gamma^*, \lambda_1^*, \ldots, \lambda_m^*$ such that*

$$g(x^*, t_i) = 0 \ \ \forall i \in 1, \ldots, m,$$

$$\text{and} \ \ \gamma^* \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla_x g(x^*, t_i) = 0.$$

Then x^* is a stationary point of the SIP.

Assumption 1.2 *At each stationary point of interest, an unspecified constraint qualification holds which implies $\gamma^* \neq 0$. For convenience $\gamma^* = 1$ is assumed.*

The stationary points satisfying assumption (0.1.2) will be referred to as Karush-Kuhn-Tucker (KKT) points. The solution points of the PFP are characterized as follows.

Definition 1.3 *For fixed values μ_0 and ν_0 of μ and ν, a point x_0 is a* ***critical point*** *of $\phi(\mu_0, \nu_0; x)$ iff at x_0 the directional derivative of $\phi(\mu_0, \nu_0; x)$ with respect to x along every direction is non-negative.*

It can be shown [9, 10] that if x^* is a KKT point of the SIP, then satisfaction of the condition

$$\mu > \|\lambda^*\|_1, \tag{4}$$

where $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)^T$, ensures x^* is also a critical point of $\phi(\mu, \nu; x)$. Conversely, if x^* is feasible, and is a critical point of ϕ for some $\mu > 0$ and $\nu \geq 0$, then x^* is also a solution point of the SIP. The following assumption, which ensures the SIP is tractable, is made.

Assumption 1.4 *For each $x \in R^n$, the set of global maximisers $\Gamma(x)$ of $g(x,t)$ over T is finite.*

Using this assumption, for any $x \in R^n$ a continuous piecewise quadratic ψ approximating ϕ about x can be constructed. Specifically,

$$\psi(x, \mathcal{A}; \mu, \nu; s) = f(x) + s^T \nabla f(x) + \tfrac{1}{2} s^T H s + \mu\zeta(s) + \tfrac{1}{2}\nu\zeta^2(s),$$

$$\text{where } \zeta(s) = \max_{t \in \mathcal{A}} [g(x,t) + s^T \nabla_x g(x,t)]_+,$$

where H is positive definite, and where $\mathcal{A} \subset T$ is finite. The matrix H is used to include second derivative information, and it is updated at each iteration. Clearly ψ is strictly convex in s, and has a unique minimum with respect to s over $s \in R^n$. It can be shown [10] that if $\Gamma(x) \subseteq \mathcal{A}$ then, in the limit $s \to 0$

$$\phi(\mu, \nu; x + s) = \psi(x, \mathcal{A}; \mu, \nu; s) + o(\|s\|).$$

Therefore, for any s_0 satisfying $\psi(s_0) < \psi(0)$, the convexity of ψ implies the directional derivative of ϕ at x in the direction s_0 (hereafter $D_{s_0}\phi(x)$) is strictly negative. Such an s_0 exists unless the minimiser s^* of ψ is zero. If $s^* \neq 0$, then x is not a critical point of ϕ and s^* is a descent direction for ϕ at x. This provides the basis for the algorithm described in the next section.

The problem of minimising ψ is an L_∞ Quadratic Programme (L_∞QP), and can be rewritten as the Quadratic Programme (QP)

$$\min_{s\in R^n,\zeta\in R} s^T\nabla f + \tfrac{1}{2}s^T Hs + \nu\zeta + \tfrac{1}{2}\nu\zeta^2 \text{ subject to } \zeta \geq 0 \text{ and}$$

$$g(x,t) + s^T\nabla_x g(x,t) - \zeta \leq 0 \ \ \forall t \in \mathcal{A},$$

and solved accordingly. The Lagrange multipliers $\lambda^{(k)}$ from this QP are used as estimates of the optimal Lagrange multipliers λ^* when updating H and the penalty parameters.

2 Description of the Algorithm

The basic structure of each iteration of the algorithm is as follows. The superscript (k) denotes the iteration number. First the locally approximating L_∞QP about the current iterate $x^{(k)}$ is constructed. This L_∞QP is then solved to yield the proposed step $s^{(k)}$. If this step yields a sufficient reduction in the penalty function it is accepted. Otherwise a second order correction $c^{(k)}$ is calculated. The purpose of this second order correction is to prevent the Maratos effect from occurring. An Armijo linesearch is then conducted along the arc $q^{(k)}(\alpha) = x^{(k)} + \alpha s^{(k)} + \alpha^2 c^{(k)}$ for the next iterate. The process is repeated until a sufficiently accurate approximation to a critical point of the penalty function is found.

Algorithm Summary:

1. The global, and some local maximisers of $g(x^{(k)},t)$ with respect to t are found. Let $\mathcal{A}^{(k)}$ denote this set of points. This subproblem is referred to as the *multi-local optimisation* subproblem.

2. The approximating L_∞QP at $x^{(k)}$ is formed, and its solution $s^{(k)}$ is calculated. If $\theta^{(k)}$ exceeds some positive parameter θ_{cap}, then the *capping constraint* $\zeta(s^{(k)}) \leq \zeta(0)$ is imposed on the L_∞QP. If $s^{(k)} = 0$, but the Lagrange multiplier estimates indicate that the penalty parameters are too small then the penalty

parameters are updated as described in step 4 and the L_∞QP is re-solved — such an occurrence is called a 'short' iteration. If the capping constraint is active, then the penalty parameters are increased according to the rules given in step 4, except $\|\lambda^{(k)}\|_1$ is replaced by the quantity $\mu^{(k)} + \nu^{(k)}\theta^{(k)} + |\zeta|$, where ζ is the Lagrange multiplier for the capping constraint from the L_∞QP's solution. The L_∞QP is then re-solved with these new penalty parameter values. With these new penalty parameter values the capping constraint will be inactive and the algorithm will proceed to the next step.

3. If $x^{(k)} + s^{(k)}$ does not satisfy the sufficient descent condition:

$$\phi(x^{(k)}) - \phi(x^{(k)} + s^{(k)}) \geq \rho\alpha[\psi(x^{(k)}; 0) - \psi(x^{(k)}; s^{(k)})],$$

calculate $c^{(k)}$, and perform the arc search. Here $0 < \rho < \frac{1}{2}$. For all results presented herein, $\rho = 0.33$ was used.

4. Estimate the optimal Lagrange multipliers at the new iterate. If θ is less than some positive parameter $\theta_{\text{crossover}}$, and if $\mu \leq \kappa_1\|\lambda^{(k)}\|_1$, then μ is increased to $\kappa_2\|\lambda^{(k)}\|_1$, where $\kappa_2 > \kappa_1 > 1$ are fixed parameters. Related research [3] suggests that $\kappa_2 < 2$ may be desirable. If $\theta \geq \theta_{\text{crossover}}$, and $\mu + \nu\theta \leq \kappa_3\|\lambda^{(k)}\|_1$, then ν is adjusted to give $\mu + \nu\theta = \kappa_4\|\lambda^{(k)}\|_1$, where $\kappa_4 > \kappa_3 > 1$. Herein $\kappa_1 = 1.2$, $\kappa_2 = 1.5$, $\kappa_3 = 1.2$, and $\kappa_4 = 4$ have been used.

5. Update H using the BFGS update provided the following condition (0.5) is satisfied, otherwise do not update H.

$$\forall x \in R^n - \{0\},\ \forall k,\ 0 < x^T H^{(k)} x \leq \gamma x^T x, \text{ where } \gamma > 0. \qquad (5)$$

Here $\gamma = 10^8$ was used.

6. If sufficient accuracy has not been attained, another iteration is begun.

The vector $c^{(k)}$ is that of [8], and is determined as follows: The multi-local optimisation subalgorithm is applied to $g(x^{(k)} + s^{(k)}, t)$, yielding the set $\mathcal{A}^{(k)}_{\text{soc}}$. Let $\mathcal{Q}^{(k)}$ denote the set of elements $t \in \mathcal{A}^{(k)}$ which give rise to the optimal active set of

constraints for $L_\infty \mathrm{QP}^{(k)}$. Each member of $\mathcal{Q}^{(k)}$ is then matched to the closest point in $\mathcal{A}_{\mathrm{soc}}^{(k)}$. If some element of $\mathcal{A}_{\mathrm{soc}}^{(k)}$ is matched with more than one member of $\mathcal{Q}^{(k)}$ then $c^{(k)} = 0$ is used; otherwise $c^{(k)}$ is found as described in [8], where the above matching is used to pair the constraint values at $x^{(k)} + s^{(k)}$ with the constraint gradients at $x^{(k)}$. If $\|c^{(k)}\| \geq \|s^{(k)}\|$, then $c^{(k)}$ is reset to 0.

The purpose of the capping constraint in step 2 is to prevent directions of ascent for θ being chosen as $s^{(k)}$ whenever $\theta^{(k)}$ is large. Without this precaution, it is possible that the sequence $\{\theta^{(k)}\}$ diverges to infinity, because an increase in infeasibility may be offset by a reduction in the objective function. The capping constraint was found to be necessary to solve problem 13 (see section 3), for precisely this reason.

For the $p = 1$ case, each multi-local optimisation was performed by calculating $g(x^{(k)}, t)$ at a number of equally spaced points $t_0, \ldots, t_N$ in T. A local search was then performed in each interval $[t_{i-1}, t_{i+1}]$ containing a local maximum.

For the case when $p > 1$, $g(x^{(k)}, t)$ was calculated at a number of points in T, where these points were generated using a Halton sequence. These test points were then grouped into clusters, such that the algorithm considers that local searches started at any two test points will find the same point if, and only if the two test points lie in the same cluster. One local search is then performed for each cluster, using a quasi-Newton algorithm. The highest test point in each cluster is used as the starting point for that cluster's local search.

These multi-local optimisation algorithms are described in more detail in [9].

3 Numerical Results

The algorithm was tested on the 14 test problems of Watson and Coope [13, 5] (hereafter known as the Watson set of problems). The results for these problems are summarised in Table 1. Here j and h respectively denote the number of iterations and the number of multi-local optimisations performed in solving the problem. Φ' is the magnitude of the minimum directional derivative of the L_1 exact penalty function at the final iterate. The subscripts CW, TFI, B, and P respectively refer

to results by Coope and Watson [5], Tanaka et al [11], Bell [1], and by the algorithm presented herein. The superscript $\sharp$ denotes values taken by the various quantities in the final iteration of the SIP algorithm.

All computations were performed on a VAX 3100 workstation in double precision arithmetic. This gives approximately 16 digits accuracy; the machine precision being 1.39E-17.

3.1 Results for the Watson problem set.

For the problems in the Watson set with $p = 1$ (ie numbers 1–6, and 14), $\theta_{\text{cap}} = \theta_{\text{crossover}} = 1$ was used. The capping constraint was struck only on problem 6. The trust region $\|s\|_\infty \leq 2$ was also used. On problem 4 with $n > 3$, performance was improved by replacing this simple bound with

$$\|s^{(k)}\|_\infty \leq \|x^{(k)} - x^{(k-1)}\|_\infty. \tag{6}$$

All problems were solved to an accuracy of 10^{-5} except for problem 4 with $n = 8$: for this problem an accuracy of 10^{-8} was sought.

On problem 2 the algorithm converged to a different solution than that given by Watson [13], as did the algorithm of Tanaka et al [11]. Following Tanaka et al, on using $x^{(0)} = 0$ the algorithm found the solution listed by Watson. The results for the original, and altered starting points are listed in rows 2 and 3 of Table 2 respectively.

The algorithm was also tested on its ability to cope with remote starting points on problem 2. The algorithm took 12 iterations and 12 function evaluations to find the same solution from 100 times the original starting point. It took 21 iterations, 23 function evaluations, and one short iteration to find the same solution from 10,000 times the original starting point.

The capping constraint and quadratic penalty term played only passive roles in the solution of each 1 dimensional problem, excluding problem 6: for the remaining problems in the Watson set it was decided to reduce the values of θ_{cap} and $\theta_{\text{crossover}}$ from 1 in order to increase their influence on the algorithms behaviour. These

problem	n	p	$\lvert\Gamma^*\rvert$	j_P	h_P	Φ'_P	j_{TFI}	h_{TFI}	Φ'_{TFI}	j_{CW}	Φ'_{CW}
1	2	1	1	17	21	8.2E-6	17	19	4.8E-7	16	8.2E-6
2	2	1	2	8	10	1.4E-8	5	11	2.7E-8	7	1.4E-8
	2	1	2	7	8	4.9E-7	-	-	-	-	-
3	3	1	1	11	23	1.3E-6	9	12	5.5E-8	10	6.2E-12
4	3	1	2	10	11	1.9E-6	5	15	2.7E-7	5	5.4E-8
	6	1	4	57	119	7.7E-6	8	27	7.7E-6	20	6.4E-6
	8	1	5	84	164	1.0E-7	3	14	3.4E-6	16	7.4e-6
5	3	1	2	8	14	6.2E-6	4	9	6.8E-7	4	6.9E-6
	8	1	2	7	13	4.3E-6	2	6	1.2E-6	4	2.3E-8
	10	1	2	7	13	2.2E-6	2	6	7.1E-7	4	1.2E-9
	12	1	2	7	14	8.7E-6	3	7	9.2E-8	4	1.8E-8
	15	1	2	8	13	3.8E-6	3	7	6.2E-8	4	1.3E-9
6	2	1	1	27	87	5.2E-6	16	19	1.3E-18	9	1.1E-8
7	3	2	1	9	14	7.0E-9	2	4	0.0	3	0.0
8	6	2	4	34	40	4.1E-8	11	41	1.1E-7	9	1.1E-8
	10	2	5	21	27	6.7E-7	12	56	3.4E-6	-	-
	15	2	?	-	-	-	10	57	3.8E-6	-	-
9	6	2	∞	41	192	-	2	6	0.0	18	4.8E-2
10	3	2	1	2	3	1.2E-6	2	3	8.1E-7	3	2.8E-7
11	3	2	2	10	18	9.8E-7	7	18	1.6E-14	12	2.2E-7
12	3	2	1	9	17	3.8E-6	3	5	3.0E-12	4	1.7E-11
13	3	2	1	11	22	7.5E-6	4	6	2.1E-15	4	3.5E-7
14	2	1	1	6	7	8.1E-6	5	8	3.4E-7	5	8.2E-7

Table 1: *A comparison of results with those obtained by Tanaka et al.*

problems were run with $\theta_{\text{cap}} = 0.01$ and $\theta_{\text{crossover}} = 0.1$.

The only published results from the solution of SIP problems using a quasi-Newton algorithm known to the authors are those by Bell [1]. These comprise problem 4 of the Watson set with $n = 3, \ldots, 6$, and are as follows: for $n = 3$, $j_B = 29$ and $j_P = 10$; for $n = 4$, $j_B = 41$ and $j_P = 22$; for $n = 5$, $j_B = 81$ and $j_P = 32$; and for $n = 6$, $j_B = 100$ and $j_P = 57$. Bell's algorithm takes more iterations to reach a solution than the one presented herein. This is hardly surprising as Bell's algorithm starts by using quite coarse approximations to the global maximisers in the early iterations, and increases the accuracy required of these approximations as the solution process proceeds.

The three algorithms with which almost all comparisons are made here are Newton type algorithms. On the easier problems (1, 2, 3, 4 with $n = 3$, 5, 7, 10, 11, and 14) the algorithm performed well. In most cases the number of iterations taken was at most double that required by any of the Newton type algorithms, and the number of multi-local optimisations performed was less than twice the number of multi-local optimisations that the algorithm of Tanaka et al. required. The more non-linear problems (6, 12, and 13) produced greater discrepancies, but the algorithm had no difficulty in solving them.

The extended version of problem 4 was much more testing: the algorithm was able to solve it for the various values of n, however many more iterations and multi-local optimisation calls were needed than for the Newton type algorithms. In particular, the algorithm of Coope and Watson was able to achieve a higher accuracy on this problem (with $n = 8$) in lower precision arithmetic than the algorithm presented here. Both of these algorithms were able to locate all global maximisers, as was that of Tanaka et al. Watson's [13] algorithm respectively missed 1, and 2 of the global maximisers on the $n = 6$ and $n = 8$ problems.

The margin between the Newton type algorithms, and that described here was greatest on problem 8. The algorithm coped quite well with the $n = 6$ case, requiring one less multi-local optimisation call than the algorithm of Tanaka et al. although

many more iterations were taken. The $n = 10$ case was very different: the local search procedure used in the multi-local optimisations experienced much difficulty in accurately calculating the local maximisers of the constraint function. Convergence was obtainable, but only by using the $n = 6$ solution as a starting point.

3.2 Results for C^1 problems.

The algorithm is designed to be capable of solving C^1 problems. To test its ability on such problems it was applied to the following three problems. The results are listed with each problem.

Problem L.

$$f = (x_1 + x_2 - 2)^2 + (x_1 - x_2)^2 + 30(\min\{0, x_1 - x_2\})^2$$

$$g(x,t) = x_1 \cos(t) + x_2 \sin(t) - 1 \qquad T = [0, \pi]$$

$$x^{(0)} = (0, -0.1)^T$$

The objective function of this problem has discontinuous second derivatives at the solution. The following solution was found in 11 iterations and 17 multi-local optimisations:

$$x^\sharp = (0.7071, 0.7071)^T \quad f^\sharp = 0.3431; \quad \Gamma^\sharp = \{0.7854\}; \quad \mathcal{A}^\sharp - \Gamma^\sharp = \emptyset.$$

$$\Phi' = 5.3E-6; \;\; \theta^\sharp = 9.5E-14; \;\; \mu^\sharp = 2.5909; \;\; \nu^\sharp = 0.1;$$

The sequence of iterates crossed the line $x_1 = x_2$ (along which the second derivatives are discontinuous) three times whilst solving this problem.

Problem M.

$$f = (x_1 - 2)^2 + x_2^2$$

$$g(x,t) = x_1 \cos(t) + x_2 \sin(t) - 1 \qquad T = [0, \pi]$$

$$\text{and} \quad \|x\|_\infty \leq 1, \quad \text{where} \quad x^{(0)} = (0, 0.1)^T$$

In this problem strict complementarity fails to hold for the global maximiser of g at $x = x^*$. Accordingly, at x^* the boundary of the feasible region changes over from the envelope $g(x, \arctan(x_2/x_1)) = 0$ to the ordinary constraint $g(x, 0) = 0$. The join between these two pieces is C^1, but not C^2. The problem was solved in 4 iterations with 4 multi-local optimisations being made. In addition to this one 'short' iteration was also performed. The solution found is:

$$x^\sharp = (1,0)^T \quad f^\sharp = 1; \quad \Gamma^\sharp = \{0\}; \quad \mathcal{A}^\sharp - \Gamma^\sharp = \emptyset.$$

$$\Phi' = 6.8E-16; \quad \theta^\sharp = 0; \quad \mu^\sharp = 3.5147; \quad \nu^\sharp = 0.1;$$

Problem N.

$$f = x_2$$

$$g(x,t) = 2.0x_1^2t^2 + t^4 + x_1^2 - x_2 \qquad T = [-1,1]$$

$$x^{(0)} = (0.5, 0.5)^T$$

In this problem the implicit function theorem fails to hold for the global maximiser of g at the SIP's solution x^*. Actually, for $x_1 > x_1^*$ there are two global maximisers which combine into one at $x_1 = x_1^*$. For $x_1 \leq x_1^*$ there is only one global maximiser. The following solution was found in 9 iterations and 11 multi-local optimisations:

$$x^\sharp = (0,0)^T \qquad f^\sharp = 0;$$

$$\Phi' = 2.5E-6; \quad \theta^\sharp = 6.9E-16; \quad \mu^\sharp = 1.5270; \quad \nu^\sharp = 0.1;$$

$$\Gamma^\sharp = \{0\}; \quad \mathcal{A}^\sharp - \Gamma^\sharp = \emptyset.$$

In solving this problem the number of local maximisers changed four times.

4 Higher Dimensional Problems.

Three problems involving constraint index sets of dimension greater than two were looked at. The first (problem S) was designed to be a non-trivial problem, but one which was not overly difficult. The second (problem T) was chosen to be quite testing

Problem	n	p	$\lvert\Gamma^*\rvert$	j	h	cpu time
S	4	3	1	24	60	64.05
S	4	4	1	20	37	152.36
S	4	5	1	21	36	331.59
S	4	6	1	23	43	1081.71
T	4	3	4	23	48	125.52
T	4	4	4	20	39	456.29
T	4	5	4	26	68	988.23
T	4	6	4	26	64	5855.90
U	4	6	2	17	18	414.11

Table 2: *Results for the higher T dimensional problems.*

of the algorithm's ability to keep track of local maximisers which merge into one another, and then split apart as the iteration number k is increased. Fortuitously, this problem is also a good test of an algorithm's ability to cope with a constraint function which has an almost flat region taking values close to the global maximum.

On all runs performed on the higher dimensional problems the trust region (0.6) was used, as was $\theta_{\text{cap}} = \theta_{\text{crossover}} = 1$. A summary of the results for these three problems is given in Table 0.2. The symbol $|\Gamma^*|$ denotes the actual number of global maximisers active at the solution x^*. The cpu time is in seconds, and includes input/output time which is of the order of 4 to 10 seconds.

The results for problems S and T show a steady and large increase in computational time as p is increased. This follows from the increased effort needed to solve the multi-local optimisation subproblems as the dimension of T increases.

Problem S.

$$\begin{aligned} f(x) &= x_1x_2 + x_2x_3 + x_3x_4 \\ g(x,t) &= 2(x_1^2 + x_2^2 + x_3^2 + x_4^2) - 6 - 2p \end{aligned}$$

$$+ \sin(t_1 - x_1 - x_4) + \sin(t_2 - x_2 - x_3)$$
$$+ \sin(t_3 - x_1) + \sin(2t_4 - x_2) + \sin(t_5 - x_3) + \sin(2t_6 - x_4)$$

$$T = [0,1]^p$$

$$x^{(0)} = (1,1,1,1)^T$$

This problem was solved for $p = 3$, 4, 5, and 6. The results are as follows:

For $p = 3$:

$$x^\sharp = (0.894135, -1.290617, 1.235788, -0.748821)^T$$

$$f^\sharp = -3.674298; \quad \theta^\sharp = 0; \quad \mu^\sharp = 1.012; \quad \nu^\sharp = 1.0$$

$$\Gamma^\sharp = \{(1.7161, 1.5160, 2.0000)^T\}; \quad \mathcal{A}^\sharp - \Gamma^\sharp = \emptyset$$

For $p = 4$:

$$x^\sharp = (0.948247, -1.361576, 1.300981, -0.787553)^T$$

$$f^\sharp = -4.087086; \quad \theta^\sharp = 2.689491E-8; \quad \mu^\sharp = 0.932; \quad \nu^\sharp = 1.0$$

$$\Gamma^\sharp = \{(1.7315, 1.5102, 2.0000, 0.1046)^T\};$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \{(1.7315, 1.5102, 2.0000, 2.0000)^T\}$$

For $p = 5$:

$$x^\sharp = (0.913759, -1.391873, 1.516069, -0.868445)^T$$

$$f^\sharp = -4.698634; \quad \theta^\sharp = 0; \quad \mu^\sharp = 0.7335; \quad \nu^\sharp = 1.0$$

$$\Gamma^\sharp = \{(1.6161, 1.6950, 2.0000, 0.0895, 2.0000)^T\};$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \{(1.6161, 1.6950, 2.0000, 2.0000, 2.0000)^T\}$$

For $p = 6$:

$$x^\sharp = (0.960921, -1.456291, 1.581476, -0.905873)^T$$

$$f^\sharp = -5.135086; \quad \theta^\sharp = 2.254294E-10; \quad \mu^\sharp = 1.013; \quad \nu^\sharp = 1.0$$

$$\Gamma^\sharp = \{(1.6258, 1.6960, 2.0000, 0.0573, 2.0000, 0.3325)^T\}$$

$$\begin{aligned}\mathcal{A}^\sharp - \Gamma^\sharp &= \{(1.6258, 1.6960, 2.0000, 2.0000, 2.0000, 2.0000)^T \\ &\quad (1.6258, 1.6960, 2.0000, 2.0000, 2.0000, 2.0000)^T \\ &\quad (1.6258, 1.6960, 2.0000, 0.0573, 2.0000, 0.3325)^T \\ &\quad (1.6258, 1.6960, 2.0000, 0.0573, 2.0000, 2.0000)^T\}\end{aligned}$$

Problem T.

$$\begin{aligned}f(x) &= \sum_{i=1}^{4} x_i^2 - x_i \\ g(x,t) &= -\sum_{i=1}^{4} x_i^2 + \sum_{i=1}^{4} \frac{1}{1+w_i} \\ T &= [-3,3]^p \\ x^{(0)} &= (-2.25, -2.5, -2.75, -3.0)^T\end{aligned}$$

where

$$\begin{aligned}w_1 &= \sum_{j=1}^{p} [t_j - x_1]^2 \\ w_2 &= \sum_{j=1}^{p} [t_j - x_2(-1)^j]^2 \\ w_3 &= \sum_{j=1}^{p} [t_j - x_3(-1)^{j \operatorname{div} 2}]^2 \\ w_4 &= \sum_{j=1}^{p} [t_j - x_4(-1)^{(j+1) \operatorname{div} 2}]^2\end{aligned}$$

This problem was solved for values of p ranging from 3 to 6. For each value of p there are four global maximisers active at the solution x^*. Lagrange multiplier estimates indicate that at most two of the four global maximisers are needed to satisfy the first order KKT conditions at x^*.

For $p = 3$:

$$x^\sharp = (0.659449, 0.659446, 0.659446, 0.659441)^T$$

$$f^\sharp = -0.898308; \quad \theta^\sharp = 0; \quad \mu^\sharp = 2.128; \quad \nu^\sharp = 1.0$$

$$\begin{aligned}\Gamma^\sharp &= \{(0.4502, 0.4502, 0.4502)^T, (0.4502, -0.4502, -0.4502)^T, \\ &\quad (-0.4502, 0.4502, -0.4502)^T\}\end{aligned}$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \{(0.0000, 0.0000, 0.0000)^T\}$$

In this problem the multi-local optimisation algorithm actually misses the global maximiser at $(-0.4502, -0.4502, 0.4502)^T$. The value taken by $g(x^\sharp, .)$ at the origin is -0.00382. The closeness of this value to zero indicates that $g(x^\sharp, t)$ is very nearly flat in the region 'between' the four global maximisers. This near flatness, and the fact that all the global maximisers lie in a small part of T make them quite difficult to locate. Similar remarks apply to the constraint function at x^* for $p = 4, 5$, and 6.

For $p = 4$:

$$x^\sharp = (0.659442, 0.659450, 0.659448, 0.659443)^T$$

$$f^\sharp = -0.898308; \quad \theta^\sharp = 1.734531E - 6; \quad \mu^\sharp = 2.317; \quad \nu^\sharp = 1.0$$

$$\begin{aligned} \Gamma^\sharp \quad = \quad & \{(0.4502, 0.4502, 0.4502, 0.6594)^T, \\ & (-0.4502, -0.4502, 0.4502, 0.6594)^T, \\ & (0.4502, -0.4502, -0.4502, 0.6594)^T\} \end{aligned}$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \{(0.0000, 0.0000, 0.0000, 0.6594)^T\}$$

Once again one of the global maximisers has been missed by the algorithm.

For $p = 5$:

$$x^\sharp = (0.636215, 0.636215, 0.636216, 0.636215)^T$$

$$f^\sharp = -0.925782; \quad \theta^\sharp = 0.0; \quad \mu^\sharp = 2.232; \quad \nu^\sharp = 1.0$$

$$\begin{aligned} \Gamma^\sharp \quad = \quad & \{(0.5420, 0.4941, 0.4941, 0.6362, 0.5420)^T, \\ & (-0.5420, 0.4941, -0.4941, 0.6362, -0.5420)^T, \\ & (-0.5420, -0.4941, 0.4941, 0.6362, -0.5420)^T, \\ & (0.5420, -0.4941, -0.4941, 0.6362, 0.5420)^T\} \end{aligned}$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \emptyset$$

For $p = 6$:

$$x^\sharp = (0.617580, 0.617580, 0.617579, 0.617580)^T$$

$$f^\sharp = -0.944700; \quad \theta^\sharp = 0.0; \quad \mu^\sharp = 2.287; \quad \nu^\sharp = 1.0$$

$$\begin{aligned}\Gamma^\sharp = \{&(-0.5410, -0.5410, 0.5227, 0.6176, -0.5410, -0.5410)^T,\\ &(0.5410, 0.5410, 0.5227, 0.6176, 0.5410, 0.5410)^T,\\ &(-0.5410, 0.5410, -0.5227, 0.6176, -0.5410, 0.5410)^T,\\ &(0.5410, -0.5410, -0.5227, 0.6176, 0.5410, -0.5410)^T\end{aligned}$$

$$\mathcal{A}^\sharp - \Gamma^\sharp = \emptyset$$

Problem U.

$$\begin{aligned} f(x) &= \sum_{i=1}^{4} \frac{1}{10} x_i^2 - x_i \\ g(x,t) &= \frac{x_4}{5} \sin\left(30t_1 \sin(x_1) + 30t_2 \cos(x_2)\right) \\ &\quad + \frac{x_3}{10} \sin\left(\frac{t_1 t_2}{10}\right) + t_3 x_1 + t_4 x_2 + t_5 x_3 + t_6 x_4 - 4 \\ T &= [-1, 1]^6 \\ x^{(0)} &= (3, 2, 1, 0)^T \end{aligned}$$

The linearity of g in t_3, t_4, t_5, and t_6 means that finding the maximisers of g over T can be reduced from a search over six dimensions to a search over two dimensions. This feature was put in the problem to make it possible to check the algorithm's answer by hand. The algorithm made no allowance for the fact that the number of dimensions over which the local and global maximisers of g are sought can be reduced from six to two.

The results for $p = 6$ are

$$x^\sharp = (1.173288, 1.179673, 1.142275, 0.412150)^T$$

$$f^\sharp = -3.483097; \quad \theta^\sharp = 2.374750E - 11; \quad \mu^\sharp = 1.462; \quad \nu^\sharp = 1.0$$

$$\Gamma^\sharp = \{(1,1,1,1,1,1)^T, (-0.8928, -1, 1, 1, 1, 1)^T\}; \quad \Gamma^\sharp - \mathcal{A}^\sharp = 11 \text{ local maximisers.}$$

5 Using an NLP First Phase.

A two phase approach was examined. In the first phase a discretized version of the SIP was solved using an NLP algorithm. The NLP's solution was then used as the starting point for the SIP algorithm in the second phase.

The objective function of the NLP was identical to that of the SIP. The set of constraints of the NLP was $\{g(x,t) \leq 0 : t \in \mathcal{H}_m\}$, where $\mathcal{H}_m$ is the set of the first m test points generated.

The algorithm used to solve the NLP was identical to that used to solve the SIP, except that $\mathcal{A}^{(k)} = \mathcal{H}_m$ was used at each iteration instead of choosing $\mathcal{A}^{(k)}$ as the set of global (and other local) maximisers of $g(x^{(k)}, t)$.

Once the NLP is solved to the required accuracy, the SIP algorithm is applied with the NLP solution as the starting point. No alterations were made to the SIP algorithm. It did, however, use as starting values the first phase's final values of the penalty parameters and the estimate of the Hessian.

Problem S with $p = 4$ was used to test this two phase algorithm. Results were generated for various accuracies required of the NLP solution, and also for various values of m, where m is the number of constraints in the NLP. These are listed in Tables 0.3 and 0.4. The two phase algorithm found the same solution to that listed in section 4 in each case.

In Table 0.3, the parameter *Tol* represents the accuracy required of the NLP's solution. More precisely, *Tol* is both the maximum NLP constraint violation permitted, and the maximum (2-norm) residual of the derivative of the NLP's Lagrangian allowed at an acceptable solution to the NLP. The row labelled $Tol = \infty$ in Table 0.3 contains the results obtained by applying the SIP algorithm proper without an NLP first phase. The rest of the legend for Tables 0.3 and 0.4 is as follows: j_1 and j_2 are respectively the number of iterations performed in solving the NLP, and the SIP; h_1 is the number times the set of NLP constraints is evaluated, and h_2 is the number of multi-local optimisation calls made in solving the SIP; $f^{\sharp}_{NLP}$ is the value of f at the solution of the NLP; and $\|x^{\sharp}_{NLP} - x^{\sharp}\|$ is the Euclidean distance between the NLP's

Tol	First Phase					second phase			combined
	j_1	h_1	time	$f^\sharp_{NLP}$	$\|x^\sharp_{NLP} - x^\sharp\|$	j_2	h_2	time	cpu time
∞	0	0	0.00	+3.000	2.9775	20	37	152.36	152.36
1.0E-1	24	43	20.80	−4.139	0.2875	9	13	30.29	51.09
1.0E-2	26	45	21.72	−4.132	0.3645	10	14	32.59	54.31
1.0E-5	28	47	22.54	−4.132	0.3641	10	14	32.02	54.56

Table 3: *Results for a two phase algorithm on problem S with p = 4. Here the number of constraints in the first phase has been fixed at 160, and the accuracy to which the NLP was solved has been varied.*

m	First Phase					second phase			combined
	j_1	h_1	time	$f^\sharp_{NLP}$	$\|x^\sharp_{NLP} - x^\sharp\|$	j_2	h_2	time	cpu time
50	19	23	6.46	−4.340	0.7170	13	17	45.11	51.57
160	28	47	22.54	−4.132	0.3641	10	14	32.02	54.56
500	21	30	51.89	−4.128	0.1080	10	15	34.50	86.39
1600	21	25	173.58	−4.128	0.1080	11	14	33.12	206.70

Table 4: *Results for a two phase algorithm on problem S with p = 4. Here the number of constraints in the NLP has been varied, and each NLP was solved to an accuracy of 1.0E-5.*

and SIP's solutions. For the case when $Tol = \infty$, $x^{\sharp}_{NLP} = x^{(0)}$ and $f^{\sharp}_{NLP} = f(x^{(0)})$ have been used. The cpu times required to complete the first, and the second phases are listed in the two columns headed 'time.' The total time required to solve the problem is listed under the heading 'combined cpu time.' Unfortunately it was not possible to separate the input/output times from the cpu times. The input/output times are of the order of 4 to 10 seconds for the test runs listed here, with the input/output time for each run being approximately proportional to $j_1 + j_2$. In spite of this uncertainty in the times, they still provide useful information on the effects of using an NLP first phase.

The results show that the use of a first phase reduces the total time required to solve the problem. A relatively coarse discretization of the semi-infinite constraint performed better than a finer discretization. As the discretization became finer the time taken to complete the first phase increased accordingly. Curiously, the time taken to complete the second phase was relatively independent of the discretization; the second phase times for $m = 160$, 500, and 1600 being very similar.

The results for $m = 160$ and varying values of Tol also show that there is little to be gained by solving the NLP to great accuracy. Discretizing the semi-infinite constraint introduces an error between the solution of the NLP (x^*_{NLP}), and x^*. There is little point in reducing the error in the calculated value of $x^{\sharp}_{NLP}$ too much below $\|x^*_{NLP} - x^{\sharp}\|$.

6 Advantages of an Extra Penalty Parameter.

The penalty function ϕ (0.3) is a hybrid of the standard Single Parameter Exact non-differentiable Penalty Function (SPEPF) and the classical Quadratic Penalty Function (QPF). These are respectively ϕ with $\nu \equiv 0$, and with $\mu \equiv 0$. The characteristics of this hybridization are investigated by varying the threshold parameter $\theta_{\text{crossover}}$. When θ is above this threshold value, any adjustments to the penalty parameters are made to ν; below this threshold the adjustments are made to μ. If $\theta_{\text{crossover}}$ is very large, then the algorithm's behaviour imitates that of an algorithm

based on a SPEPF. If θ is very small, then the algorithm mimics a QPF based algorithm.

Problem 6 was chosen as the test problem on which to explore the effects of altering $\theta_{\text{crossover}}$. The results are presented in Table 0.5. The first and last rows of Table 0.5 list the results obtained by using a SPEPF ($\nu \equiv 0$), and a QPF ($\mu \equiv 0$) respectively. For these two rows the initial penalty parameter values were $\mu = 0.1$ and $\nu = 1.0$ respectively. For all other rows, $\mu = 0.1$ and $\nu = 1.0$ were the initial values, with $\theta_{\text{crossover}}$ as listed. Two sets of results were generated: the first set was computed using the algorithm without a capping constraint, and the second set was calculated by the algorithm with a capping constraint set at $\theta_{\text{cap}} = 1$.

The results show that without the capping constraint, the pure non-differentiable penalty function needed over twice as many iterations, and more than four times as many multi-local optimisation calls as the hybrid penalty function with $\theta_{\text{crossover}} = 1$. With $\theta_{\text{crossover}} = 100$, the algorithm did not alter ν, in which case ϕ was effectively the sum of the SPEPF and a $+\frac{1}{2}\theta^2$ term. Even this simple alteration produced a significant improvement in performance. Using lower values of $\theta_{\text{crossover}}$ improved performance further.

The SPEPF performed so poorly without either a non-zero ν or a capping constraint because many iterations are needed before a sufficiently large value of μ is obtained. With $\nu \equiv 0$, and without a capping constraint, $\mu^{(k)}$ can be at most $\kappa_2 \mu^{(k-1)}$, where $\kappa_2 = 1.5$ was used. This is a consequence of using the Lagrange multiplier estimates from the L_∞QP, which means that $\|\lambda^{(k-1)}\|_1$ is bounded above by $\mu^{(k-1)}$. The updating scheme for the penalty parameters is designed to ensure that $\mu^{(k)}$ is at most $\kappa_2 \|\lambda^{(k-1)}\|_1$. So, if $\mu^{(0)}$ is small, many iterations may be needed before a reasonable value of μ is reached. If $\nu > 0$ then $\mu^{(k)} \leq \kappa_2(\mu^{(k-1)} + \nu^{(k-1)}\zeta^{(k-1)})$ and μ can grow faster than for the SPEPF.

One might expect that the QPF's performance would be much worse than that of the hybrid penalty function. However the results did not bear this out. All calculations in all test runs were performed in double precision. This was enough to cope with the ill-conditioning arising from the high value of ν, whilst still achieving

the required accuracy of about five digits. However the deficiencies of the QPF are well known.

With the capping constraint in place, the differences between the various penalty functions were not great. The result for $\theta_{\text{crossover}} = 10\check{0}$ appears to be something of an anomaly. For $\theta_{\text{crossover}} \leq 10$ the uncapped algorithm consistently performed better than the capped algorithm; the difference however was not large.

6.1 Unrestricted increases in μ and ν

To investigate the relative merits of the SPEPF and the hybrid penalty function further, the algorithm was modified to permit arbitrarily large increases in the penalty parameters. This was accomplished by solving the L_∞QP subproblem with μ reset to a very large number: here $1.0E8$ was used. The Lagrange multiplier estimates calculated whilst solving this L_∞QP were then used to update the penalty parameter values in accordance with the rules given in section 2. The search direction was then calculated by re-solving the L_∞QP with the new penalty parameter values. The relevant results are presented in Table 0.6. In these, the SPEPF does better than the hybrid penalty function with $\theta_{\text{crossover}} = 1$. An examination of the sequences of iterates generated shows that the hybrid penalty function with $\theta_{\text{crossover}} = 1$ allows the sequence of iterates to penetrate deeper into the infeasible region than does the SPEPF. The deeper forays into the infeasible region take longer to correct. The presence or absence of a capping constraint had no effect on these numerical results.

Allowing arbitrarily large increases in μ and ν does not *quite* make the capping constraint irrelevant. The method used to estimate the Lagrange multipliers when unlimited increases are permitted ensures that the capping constraint will never be active at the solution of the L_∞QP; the capping constraint itself becomes redundant. However, using the capping constraint also imposes the extra requirement on the line search: 'if $\theta^{(k)} > \theta_{\text{cap}}$ then $\theta^{(k+1)} \leq \theta^{(k)}$.' This extra condition is still able to influence how the algorithm selects each iterate.

6.2 Decreasing the penalty parameters

Additionally, the possibility of allowing reductions in the penalty parameter values as well as unlimited increases was also looked at. To stop the algorithm from endlessly increasing and decreasing the penalty parameters it was necessary to assign μ and ν minimum values $\mu_{\min}$ and $\nu_{\min}$: initially $\mu_{\min} = 0.1$ and $\nu_{\min} = 1.0$ were used. Each time a penalty parameter was decreased, the corresponding minimum value was subsequently doubled.

The necessary changes were implemented as follows. Firstly, $\lambda^{(k)}$ was calculated as described earlier for the case of arbitrarily large increases. Any consequent increases in the penalty parameters were then made. Immediately following this, if $\theta^{(k)} \leq \theta_{\text{crossover}}$ then decreasing either or both of the penalty parameters was considered. If

$$\mu^{(k)} > \max\left(1.8\|\lambda^{(k)}\|_1, \mu_{\min}\right)$$

then the following adjustments were made, in this order:

$$\nu^{(k)} \longleftarrow \nu^{(k)} + \frac{\mu^{(k)} - \max\left(1.5\|\lambda^{(k)}\|_1, \mu_{\min}\right)}{\theta_{\text{crossover}}}$$

$$\text{and} \quad \mu^{(k)} \longleftarrow \max\left(1.5\|\lambda^{(k)}\|_1, \mu_{\min}\right).$$

The first adjustment ensures $\mu + \nu\theta$ is decreased only on the part of the infeasible region where $\theta < \theta_{\text{crossover}}$. For many problems this is the part of the infeasible region which borders on the feasible region. If

$$\nu^{(k)}\theta_{\text{crossover}} > \max\left(50\|\lambda^{(k)}\|_1, \nu_{\min}\theta_{\text{crossover}}\right),$$

then $\nu^{(k)}$ was reset as follows:

$$\nu^{(k)} \longleftarrow \max\left(\frac{4\|\lambda^{(k)}\|_1 - \mu^{(k)}}{\theta_{\text{crossover}}}, \nu_{\min}\right).$$

If $\theta > \theta_{\text{crossover}}$ then the penalty parameters were not reduced.

The results for this are presented in Table 0.7. They show that allowing decreases in the penalty parameters led to improvements in the performance of the algorithm in most cases. Once again the SPEPF did better than the hybrid penalty function.

$\theta_{\text{crossover}}$	Not Capped				Capped			
	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$
SPEPF	42	130	977.1	0	21	40	334.0	0
100	35	99	16970	1.0	16	25	431.0	1.0
10	16	34	210.8	24.64	19	39	429.4	86.97
1	16	31	46.49	234.5	17	34	75.13	86.97
0.1	16	31	22.42	877.0	18	35	7.583	86.97
0.01	17	32	11.79	877.0	20	38	7.341	1182
1.0E-4	19	34	7.398	55760	21	39	7.381	78030
1.0E-6	21	36	7.383	1.5E+7	24	42	7.383	5.2E+6
QPF	22	42	0	1.0E+9	22	39	0	2.1E+6

Table 5: *Variations of the algorithm's performance on problem 6 with respect to changes in* $\theta_{crossover}$.

The best result is that of the original algorithm, with $\theta_{\text{cap}} = 1$ and $\theta_{\text{crossover}} = 100$. Other than this apparently rather anomalous result, the best results were obtained using the hybrid penalty function with only restricted increases in the penalty parameters permitted, and without a capping constraint.

6.3 The effects of excessive penalty parameter values

Problem K was used to investigate the effects of excessively high values of the penalty parameters.

Problem K.

$$\begin{aligned} f(x) &= x_2^2 - 4x_2 \\ g(x,t) &= x_1\cos(t) + x_2\sin(t) - 1 \\ T &= [0,\pi] \\ x^{(0)} &= (0.9,0)^T \end{aligned}$$

$\theta_{\text{crossover}}$	Not Capped				Capped			
	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$
SPEPF	21	40	334.4	0	21	40	334.4	0
100	21	40	334.4	1.0	21	40	334.4	1.0
1	28	66	1625	3.2E+7	28	66	1625	3.2E+7
0.01	28	68	407.9	3.2E+7	28	68	407.9	3.2E+7

Table 6: *The hybrid PF, and the SPEPF with unlimited increases in the penalty parameters permitted.*

$\theta_{\text{crossover}}$	Not Capped				Capped			
	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$	j_P	h_P	$\mu^\sharp$	$\nu^\sharp$
SPEPF	19	34	6.490	0	19	36	7.006	0
100	19	34	6.490	1.142	19	36	7.006	1.237
1	24	37	6.781	4.000	26	41	6.711	10.57
0.01	32	65	7.365	19.64	29	63	7.340	19.57

Table 7: *The hybrid PF, and the SPEPF with unlimited increases, and with decreases in the penalty parameters permitted.*

The *exact* solution of this problem is:

$$x^* = (0,1)^T; \quad f^* = -3;$$

$$\Gamma^* = \left\{\frac{\pi}{2}\right\}; \quad \text{and} \quad \mathcal{A}^* - \Gamma^* = \emptyset.$$

Also $\mu^\sharp > 2$ is required for x^* to be a local minimum of the penalty function.

This problem contains a single convex constraint. The initial point lies near this constraint, and the solution lies on it. Between the initial point and the solution the gradient of the objective function points into the constraint. This problem tests an algorithm's ability to generate a sequence of iterates which efficiently skirts around the convex constraint to the solution. As the penalty parameters are increased, the constraint becomes more nearly impenetrable — forcing the algorithm to generate iterates which are either feasible, or only marginally infeasible.

Results were generated for a variety of values of μ and ν. These parameters were kept constant during each run of the algorithm. The results are listed in Table 0.8 in two groups. The first is for the SPEPF: $\nu \equiv 0$ is used for each of these runs. The second group is for the hybrid penalty function. In the latter group $\mu = 3$ has been used, as this is κ_2 (=1.5) times the minimum value of μ needed to make the solution of problem K a local minimum of ϕ.

The results show that the number of iterations and multi-local optimisation calls required to solve the problem rises with increasing values of either penalty parameter. The degradation in the SPEPF algorithm's performance caused by increasing μ by a factor γ is roughly the same as the degradation in the hybrid penalty function algorithm's performance caused by scaling ν by γ^2.

7 Discussion

The $\frac{1}{2}\nu\theta^2$ penalty term of (0.3) was included to provide a mechanism for reducing the risk that μ would be set at an excessively high value (in fact it has also proved to be advantageous in the NLP case [4]). Problem K was designed specifically to test the effects of including the second penalty term. As expected [3], excessively high values

μ	ν	j_P	h_P
3	0	8	15
10	0	9	17
30	0	16	43
100	0	28	82
300	0	35	124
1000	0	44	139
3	1	8	15
3	10	8	15
3	1E+2	9	17
3	1E+3	22	60
3	1E+4	20	54
3	1E+5	38	105
3	1E+6	40	152
3	1E+7	53	152

Table 8: *Results for problem K with various values of the penalty parameters. Both penalty parameters were fixed during each run.*

of either penalty parameter impair the algorithm's performance. These results also show that an excessively high value of ν degrades the algorithm's performance less than a correspondingly high value of μ. Accordingly, the scheme used to update the penalty parameters should try to avoid selecting unnecessarily large values, particularly for μ. Unfortunately such values may be unavoidable for a variety of reasons, notably:

- Reductions in the penalty parameters are not permitted, and the initial values of the penalty parameters are excessive.
- A highly infeasible iterate is encountered, and one or other penalty parameter must be large if near feasibility is to be subsequently attained.
- The Lagrange multiplier estimates are highly inaccurate.

The inclusion of the second penalty parameter does reduce the susceptibility to the last two causes listed. However if $\mu^{(0)}$ is excessive, then the $\frac{1}{2}\nu\theta^2$ term is of little use. In spite of the results, permitting only restricted increases of the penalty parameters could easily lead to excessive values on some problems, especially as a result of the second reason listed. Many iterations may be wasted before μ and ν are large enough to achieve feasibility. In addition, the restrictions on the increases in μ and ν are a product of using the Lagrange multiplier estimates from the L_∞QP's solution. If the Lagrange multiplier estimates are calculated in some other way (for example, first order estimates are used) then any restriction of the form $\mu^{(k)} \leq \kappa_2\mu^{(k-1)}$ becomes essentially ad hoc in nature. Hence permitting both increases and decreases is apparently advantageous. It should be noted that on some problems, permitting decreases in μ and ν may allow the algorithm to cycle until the minimum values of the penalty parameters become high enough to force convergence. In such cases the early iterations are likely to achieve little other than waste time. It appears there is no 'right' strategy: the best scheme depends on the nature of the problem being solved. It is reasonable to expect that, on average, allowing both increases and decreases would be the better strategy on more difficult problems.

The best choice for the capping constraint value θ_{cap} varies from problem to problem. If θ_{cap} is too small then the sequence of iterates may be forced to follow closely a tightly curving constraint: a task that can require many iterations. In contrast, if θ_{cap} is too large, then it is possible for the sequence of iterates to penetrate deeply into the infeasible region. This risks having to set one or other penalty parameter to a large value in order to regain near feasibility. More seriously, it is possible that $\theta(x)$ has strict local minimisers in the infeasible region. For sufficiently large μ and ν, there will be corresponding infeasible local minimisers in ϕ. Convergence to such a local minimiser is tantamount to failure of the algorithm. An appropriate value of θ_{cap} may lessen the risk of an infeasible local minimiser of ϕ 'trapping' the sequence of iterates.

8 Conclusion

The numerical results show that the algorithm is effective on a wide variety of problems, including those which are C^1 but not C^2. In contrast, the algorithms of Watson, Coope and Watson, and Tanaka et al require second derivatives. The differences in the performances of the quasi-Newton algorithm and of the Newton type algorithms are typical of those for finite nonlinear programmes.

The results for problems S, T, and U indicate that the main increase in computational effort as p increases is due to the increasing computational expense of each multi-local optimisation; the number of multi-local optimisations did not change much as p increased. The development of efficient multi-local optimisation algorithms is crucial if SIP problems with p large are to be solved in a reasonable amount of time.

On problem S, with $n = 4$, the use of an NLP first phase reduced the time required to solve the problem by almost a factor of three. The best results were obtained by using a coarse discretization of the semi-infinite constraint, and then calculating the solution of the resultant NLP to a low accuracy only. The accuracy of the approximation to the solution found by the first phase could be improved

by either using a finer discretization of the semi-infinite constraint, or solving the resulting NLP to a higher accuracy, or both. However, the benefits of a more accurate initial value for the second phase were more than offset by the extra effort required to obtain it.

The work of this paper shows that the time taken to solve an SIP can be reduced by employing a two phase approach: for instance, the discretization strategy described by Hettich and Gramlich [6, 7] could be used as a first phase, followed by the algorithm presented herein as a second phase.

References

[1] Bell, B. M., *Global convergence of a Semi-Infinite Optimisation method*, Appl. Math. Opt. 21, pp 89–110 (1990).

[2] Conn, A. R. and N. I. M. Gould, *An exact penalty function for semi-infinite programming*, Math. Prog. 37, pp 19–40 (1987).

[3] Coope, I. D., *The Maratos effect in sequential quadratic programming algorithms using the L_1 exact penalty function*, Technical Report CS85-32, Computer Science Department, University of Waterloo (1985).

[4] Coope, I. D. and C. J. Price, *A two parameter Exact Penalty Function for Nonlinear Programming*, to appear.

[5] Coope, I. D. and G. A. Watson, *A projected Lagrangian algorithm for semi-infinite programming*, Math. Prog. 32, pp 337–356 (1985).

[6] Hettich R., *An implementation of a discretization method for semi-infinite programming*, Math. Prog. 34, pp 354–361 (1986).

[7] Hettich R. and G. Gramlich, *A note on an implementation of a method for semi-infinite quadratic programming*, Math. Prog. 46, pp 249–254 (1990).

[8] Mayne D. Q. and E. Polak, *A superlinearly convergent algorithm for constrained optimisation problems*, Math. Prog. Study 16, pp 45–61 (1982).

[9] Price, C. J., *Nonlinear Semi-Infinite Programming*, Ph.D. Thesis, Department of Mathematics, University of Canterbury, New Zealand (1992).

[10] Price, C. J. and I. D. Coope, *An exact penalty function algorithm for semi-infinite programmes*, BIT 30, pp 723–734 (1990).

[11] Tanaka, Y., M. Fukushima, and T. Ibaraki, *A globally convergent SQP method for semi-infinite nonlinear optimisation*, J. Comp. Appl. Math. 23, pp 141–153 (1988).

[12] Watson, G. A., *Globally convergent methods for semi-infinite programming*, BIT 21, pp 362–373 (1981).

[13] Watson, G. A., *Numerical experiments with globally convergent methods for semi-infinite programming problems*, pp 193–205 in 'Semi-infinite programming and applications,' A. V. Fiacco and K. O. Kortanek (Eds.), Proceedings of an international symposium, ©Springer-Verlag Berlin 1983.