RAIRO Operations Research
RAIRO Oper. Res. **41** (2007) 19–34
DOI: 10.1051/ro:2007009

LIMITED MEMORY SOLUTION OF BOUND CONSTRAINED CONVEX QUADRATIC PROBLEMS ARISING IN VIDEO GAMES [*]

Michael C. Ferris[1, 2], Andrew J. Wathen[1] and Paul Armand[3]

Abstract. We describe the solution of a bound constrained convex quadratic problem with limited memory resources. The problem arises from physical simulations occurring within video games. The motivating problem is outlined, along with a simple interior point approach for its solution. Various linear algebra issues arising in the implementation are explored, including preconditioning, ordering and a number of ways of solving an equivalent augmented system. Alternative approaches are briefly surveyed, and some recommendations for solving these types of problems are given.

Keywords. Interior point method, nonlinear complementarity problem, bound constrained problem, limited memory method.

Mathematics Subject Classification. 90C20, 90C51.

Received March 30, 2005. Accepted June 23, 2006.

[*] *This material is based on research partially supported by the Smith Institute, EPSRC Grant GR/M59044, the National Science Foundation Grant CCR-9972372 and DMS-0427689, the Air Force Office of Scientific Research Grant FA9550-04-1-0192, the Guggenheim Foundation, and the Centre National de la Recherche Scientifique.*

[1] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK.

[2] *Permanent address:* Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706, USA.

[3] Laboratoire XLIM, Université de Limoges, Faculté des Sciences et Techniques, 123, avenue Albert Thomas, 87060 Limoges, France ; `armand@unilim.fr`

Introduction

This paper describes the solution of a problem arising in the application of complementarity for physical simulations that occur within video games. The size of the problems is typically not very large, ranging from around 20 variables to a current limit of around 400 variables. The computational time available to solve each instance of the problem is limited by the frame rate of the simulation, and the memory allowed to solve each problem is severely restricted by the hardware available on many of the existing game (console) platforms. The typical time frame is of the order of milliseconds, while the amount of fast RAM available is 4-16K. A further important feature of the solution technique is that worst case behavior is very important – if large spikes in computation occur this can lead to loss of frames and jumpy screen animations.

While these problems are clearly not large scale, the limited memory requirement means that techniques normally associated with large scale problems are pertinent. In particular, limited memory methods and conjugate gradient techniques would appear to be applicable.

We now describe some mathematical background on the problem. To handle collisions in physical simulation, it is normally necessary to solve Linear Complementarity Problems (LCP's) very efficiently. While more general formulations are typically of interest, the form of the LCP considered here is a bound constrained convex quadratic program

$$\min \left\{ \frac{1}{2} x^\top A x + v_0^\top x : l \leq x \leq \tilde{h} \right\}.$$

Here, x is the vector of length n to be found, l, $\tilde{h}$ and v_0 are given vectors, the bounds hold componentwise, and A is a symmetric positive semidefinite matrix of the form $A = JM^{-1}J^\top + D$. The matrix A is not computed explicitly, but is given by J, M^{-1} and D. Here $M = \operatorname{diag}(M_1, \ldots, M_k)$ is block diagonal and each $M_i = \operatorname{diag}(m_i, m_i, m_i, I_i)$ is 6×6 with m_i and I_i being the mass and inertia tensor for the ith physical body. In fact, x is the vector of the impulses at each physical contact, $J^\top x$ is the vector of the impulses applied to the bodies, $M^{-1}J^\top x$ is the vector of velocity changes of the bodies, $JM^{-1}J^\top x$ is the vector of relative velocity changes at the physical contacts, and (if we ignore D) $Ax + v_0$ is the vector of relative velocities at the physical contacts.

The matrix J is sparse and represents collisions. If bodies i and j are capable of colliding then J has a set of rows with nonzero entries only for those bodies; thus it has a (collisions $\times$ bodies) block structure.

The matrix D is diagonal with small positive or zero diagonal elements. Physically, a positive element would correspond to a small springiness in the constraints, but they are not put in to represent a physical effect. They are sometimes added to make A positive definite and guarantee uniqueness in x. Note that if A is only semidefinite then $Ax + v_0$ will still be unique, even if x is not.

We have around 1800 test problems from the application each of which is a convex quadratic optimization with simple bound constraints. They are set up as two suites of problems, one labelled "ball" and the other labelled "topple". The order of the problems is due to the visualization being performed. A ball is rolling and bumping into things, a stack of bricks is toppling. The results show that the difficulty varies as the simulation ensues. While the original problem description specifies bounds on all the variables, it was decided to treat any bounds with magnitude over 10^{20} as infinite. The resulting problems then have some doubly bounded variables, some singly bounded variables and some free variables.

The remainder of this paper is organized as follows. In Section 1 we describe a simple interior point approach to solving the problem and show this to be effective in terms of iteration count on the problems at hand. Section 2 explores the linear algebra issues that arise in an implementation of the interior point approach for this application. In particular, we investigate preconditioning, ordering and various ways of solving an equivalent augmented system. Section 3 briefly surveys other approaches that were considered and discusses the pros and cons of them compared to the interior point approach. The paper concludes with some recommendations for solving these types of problems and indicates some thoughts for future research.

1. Interior point method

For computational ease, the problems were transformed by a simple linear transformation so that doubly bounded variables lie between 0 and a finite upper bound and singly bounded variables are simply nonnegative. Such changes clearly do not affect the convexity properties of the objective function and result in some simple shifts coupled with a multiplication by -1 of the rows and columns of A (or equivalently the rows of J) corresponding to singly upper bounded variables. The resulting problem is thus:

$$\min \left\{ \frac{1}{2} x^\top H x + q^\top x : x \in \mathbf{B} \right\}$$

where

$$\mathbf{B} = \{x : 0 \le x_{\mathcal{B}} \le u, 0 \le x_{\mathcal{L}}, x_{\mathcal{F}} \,\text{free}\}.$$

Introducing multipliers $s_{\mathcal{B}}$, $s_{\mathcal{L}}$ and ξ for the bound constraints, the first order optimality conditions of this convex problem are both necessary and sufficient and can be written (see [23], for example)

$$\begin{array}{rcl} g := Hx + q - \left[\begin{array}{c} s_{\mathcal{B}} - \xi \\ s_{\mathcal{L}} \\ 0 \end{array} \right] & = & 0 \\ 0 \le x_{\mathcal{B}} & \perp & s_{\mathcal{B}} \ge 0 \\ 0 \le x_{\mathcal{L}} & \perp & s_{\mathcal{L}} \ge 0 \\ 0 \le u - x_{\mathcal{B}} & \perp & \xi \ge 0 \end{array}$$

where the perp notation means that in addition to the nonnegative bounds, primal and dual variables are orthogonal.

We apply the Primal-Dual Framework for LCP to this problem (see [23], pp. 158–160). The critical system of linear equations (using the standard capitalization notation) that must be solved at each iteration is:

$$\begin{bmatrix} H_{\mathcal{BB}} & H_{\mathcal{BL}} & H_{\mathcal{BF}} & -I & & I \\ H_{\mathcal{LB}} & H_{\mathcal{LL}} & H_{\mathcal{LF}} & & -I & \\ H_{\mathcal{FB}} & H_{\mathcal{FL}} & H_{\mathcal{FF}} & & & \\ S_{\mathcal{B}} & & & X_{\mathcal{B}} & & \\ & S_{\mathcal{L}} & & & X_{\mathcal{L}} & \\ -\Xi & & & & & U - X_{\mathcal{B}} \end{bmatrix} \begin{bmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{L}} \\ \Delta x_{\mathcal{F}} \\ \Delta s_{\mathcal{B}} \\ \Delta s_{\mathcal{L}} \\ \Delta \xi \end{bmatrix} = -\Phi$$

where

$$\Phi = \begin{bmatrix} g \\ X_{\mathcal{B}} s_{\mathcal{B}} - \sigma\mu e \\ X_{\mathcal{L}} s_{\mathcal{L}} - \sigma\mu e \\ (U - X_{\mathcal{B}})\xi - \sigma\mu e \end{bmatrix}$$

with

$$\sigma \in [0,1] \quad \text{and} \quad \mu = \frac{x_{\mathcal{B}}^\top s_{\mathcal{B}} + x_{\mathcal{L}}^\top s_{\mathcal{L}} + (u - x_{\mathcal{B}})^\top \xi}{\|\mathcal{L}\| + 2\|\mathcal{B}\|},$$

where $\|\cdot\|$ denotes the cardinal of a finite set.

We first eliminate $\Delta s_{\mathcal{B}}$, $\Delta s_{\mathcal{L}}$ and $\Delta\xi$ from this system to recover the following problem:

$$(H + \theta)\Delta x = -r \tag{1}$$

where

$$\theta = \operatorname{diag}(X_{\mathcal{B}}^{-1} s_{\mathcal{B}} + (U - X_{\mathcal{B}})^{-1}\xi, X_{\mathcal{L}}^{-1} s_{\mathcal{L}}, 0_{\mathcal{F}})$$

and

$$r = Hx + q - \sigma\mu \begin{bmatrix} X_{\mathcal{B}}^{-1} e - (U - X_{\mathcal{B}})^{-1} e \\ X_{\mathcal{L}}^{-1} e \\ 0 \end{bmatrix}.$$

Once this system is solved, we can recover all the required values using back substitution.

Some points of note. We use two stepsizes, for primal and dual variables, each one been computed by means of the fraction to the boundary rule with 0.9995 as parameter value. We choose $\sigma = 0.1$ as initial value of the centering parameter. Whenever one of the stepsize becomes too small (less than 0.1), the centering parameter is set to 0.3, otherwise it is set to $\max\{\sigma, (0.1+\sigma)/2\}$. We initialize the method (for the translated problem) at a point where $x_i = 1$, $i \in \mathcal{L}$, $x_i = u_i/2$, $i \in \mathcal{B}$ and $x_i = 0$, $i \in \mathcal{F}$. The dual variables are set to

$$s_i = \begin{cases} 0.1 + \max\{0, H_{i\cdot}x + q_i\} & i \in \mathcal{B}, \\ \max\{0.1, H_{i\cdot}x + q_i\} & i \in \mathcal{L} \end{cases} \quad \text{and } \xi_i = s_i - (H_{i\cdot}x + q_i), i \in \mathcal{B},$$

where $H_{i\cdot}$ stands for the i-th row of H. We terminate the interior point method when

$$\mu \leq 10^{-8} \quad \text{and} \quad \|g\| \leq (1 + \|q\|)10^{-6}.$$

The experiments in Matlab on our two suites of test problems show that the interior point method is an effective solution approach. While there is some variation over the suite of problems, the maximum number of iterations is 22 for all test problems and the vast majority of the solutions occur in no more than 20 iterations of the interior point method. The average number of interior point iterations is about 13. This approach appears very promising provided that we can solve the linear systems within the time and space constraints imposed by the application.

2. Linear algebra

In this section we explore the linear algebra issues to solve the critical linear system at each interior point iteration. We use a preconditioned conjugate gradients method (PCG) (see [22], pp. 244–250) to solve the system (1) which we will refer to as the primal system. We also investigate the possibility to solve an equivalent augmented system which we will refer to as the dual system. In both cases, we examine different preconditioners (diagonal preconditioners and incomplete Cholesky factorization) and also the possibility of reordering the system.

2.1. Primal system

Symmetry and positive (semi-)definiteness allow us to use PCG to solve the primal system (1). The work involved in an iteration of this method is one matrix$\times$vector multiplication plus 5 vector operations (3 vector updates and 2 dot products) and 3 additional vectors require storage. The key issue is to determine an effective preconditioner for $H + \theta$ so that the number of linear iterations needed is small, but little additional memory is required. Note that H has the form $\tilde{J}M^{-1}\tilde{J}^\top + D$, where $\tilde{J}$ incorporates the change of signs on the rows corresponding to upper bounded variables.

We use the default convergence tolerance of a reduction in Euclidean norm of the residual in the linear system by 10^{-6} in all runs of the standard Matlab conjugate gradient code. Despite the relatively small dimension of the linear systems, the conjugate gradient method without preconditioning fails to converge on some of the resulting systems. This is even the case if we relax the convergence tolerance. Given that termination in exact arithmetic should occur in at most n steps, this indicates the poor conditioning of some of the linear systems. Thus we only report results for PCG, and only choose options for which the method achieves the above tolerance. This results in an (almost) identical sequence of iterations of the interior point method.

The first preconditioner is the simple diagonal preconditioner $\text{diag}(H) + \theta$ as suggested in [3]. Table 1 reports the computer time (Time), the choice of the preconditioner (Precond), the number of conjugate gradient iterations (CG Iter) and

TABLE 1. Results when solving (1) with the diagonal preconditioner diag(H)+θ (diag) and an incomplete Cholesky factorization (chol). The results are reported under the form mean/max.

Problem	Precond	Time	CG Iter	Nnz
Ball	diag	0.30/0.53	1223/2057	67/118
	chol	0.06/0.12	47/74	1222/1846
Topple	diag	0.25/1.97	724/3107	71/288
	chol	0.07/0.38	47/125	876/5111

the additional memory storage (Nnz, number of nonzeros in factor) per problem to solve all the systems generated by the interior point method on each problem class.

We experimented with the diagonal preconditioner $\theta + \text{diag}(\sum_j |H_{\cdot j}|)$ that attempts to incorporate the off diagonal entries in H, but the results are not as good as those of Table 1. While additional memory requirements are small, the numbers of matrix-vector products are considered unacceptable for the application.

There is evidently significant non-local coupling that can not be accounted for by these simple diagonal scaling. We therefore resort to a more sophisticated preconditioner, namely an incomplete Cholesky factorization with a drop tolerance (see for example [22]). Table 1 shows the computer time, the number of conjugate gradient iterations and the number of nonzeros in the incomplete Cholesky factor with a drop tolerance of 10^{-4}, that are required to solve each problem using the primal system. Increasing the drop tolerance led to significant increases in the number of conjugate gradient iterations and was deemed unacceptable. It is clear that the number of conjugate gradient iterations is decreased to a very reasonable number using this approach, but that the size of the factor is too large for the application. Note that the results for the "topple" problem are particularly bad for this approach. We therefore investigate alternative linear systems in order to generate preconditioners with smaller memory requirements.

2.2. DUAL SYSTEM

Alternative approaches for solving (1) stem from the equivalent augmented system:

$$\begin{bmatrix} M & \tilde{J}^\top \\ \tilde{J} & -D-\theta \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix}.$$

This symmetric and indefinite system might be preconditioned using for example the results of [15] and [20], but without making further assumptions, no effective technique of either type was found in this case. However, just as the original system (1) (the primal system) results from an elimination of Δy using the first equation, an alternative is to use the dual system where we first eliminate Δx using the second equation, then solve for Δy and finally use back substitution to

Table 2. Results when solving (2) with the diagonal preconditioner diag(N) (diag) and with an incomplete Cholesky factorization preconditioner (drop tolerance of 10^{-4}).

Problem	Precond	Time	CG Iter	Nnz
Ball	diag	0.31/0.91	1242/3494	66/102
	chol	0.06/0.15	43/103	1284/2564
Topple	diag	0.10/0.56	364/1783	29/108
	chol	0.04/0.13	30/64	242/1340

calculate Δx:

$$\begin{aligned} N &:= (M + \tilde{J}^\top (D+\theta)^{-1} \tilde{J}) \\ N\Delta y &= \tilde{J}^\top (D+\theta)^{-1} r \\ \Delta x &= (D+\theta)^{-1}(\tilde{J}\Delta y - r). \end{aligned} \tag{2}$$

In this formulation, we implicitly assume that $(D+\theta)$ is invertible. This is the case in our application examples, for which D is a positive diagonal matrix. The case where $D = 0$ is discussed in Section 2.4. Two simple diagonal preconditioners can again be used, namely diag(N) and diag($\sum_j |N_{\cdot j}|$). The results for the first are given in Table 2, and remain very similar for the second (which are not shown). The results when an incomplete Cholesky factor is used to solved the system (2) are reported in Table 2. The first point to note is that both of these are reduced on the "topple" problem as compared to the results of Table 1. This is essentially due to the fact that on these problems the size of the dual system is smaller than that of the primal system. The results for the "ball" problem are less conclusive and show that the worse case of both iteration count and number of nonzeros is increased when using the dual approach.

Figure 1 summarizes the first experiments with two performance profiles [10] on the amount of computer time for each problem suite. Each curve represents the fraction p of problems for which the method is within a factor t of the best method. A method with a large factor p (top curve) is to be preferred. It is clear that the use of an incomplete factorization outperforms the diagonal preconditioning. The solution on the dual system gives always better results for the "topple" suite, but for the "ball" suite, a part of problems is more efficiently solved with the primal system. This suggests using an approach that switches between the two systems depending on known problem characteristics. We note that for all the "topple" problems J has more rows than columns so that the dual system will be smaller than the primal system. Since in general the relative dimensions of $\tilde{J}$ are not known, we choose to solve the dual system if the number of rows of $\tilde{J}$ is greater than its number of columns. This is an empirical choice based on the number of nonzeros in the resulting (primal or dual) linear system. Using this heuristic switching mechanism, the number of conjugate gradient iterations remains at a reasonable number as shown in Table 3. Note that the "topple" problem is always

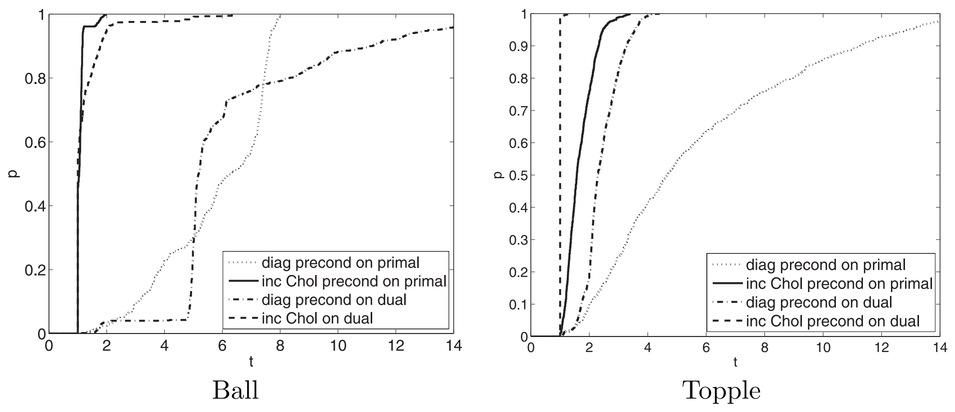

Figure 1. Performance profiles on computer time. The profiles correspond to the results reported in Tables 1–2.

Table 3. Results when using both primal and dual systems with an incomplete Cholesky factorization preconditioner (drop tolerance of 10^{-4}).

Problem	Time	CG Iter	Nnz
Ball	0.05/0.15	37/102	1151/2023
Topple	0.04/0.13	30/64	242/1340

solved using the dual approach, but the "ball" problem switches between the two systems.

2.3. Orderings and drop tolerances

If we choose to perform an incomplete factorization then we require the incomplete factors to be reasonably sparse. The storage required for an incomplete Cholesky factorization with drop tolerance of 10^{-4} is shown in Table 3. Ideally, we would like to limit the amount of storage that is available to the preconditioner to 1000 double precision entries and neither approach achieves this goal.

We have investigated ordering the systems at hand to reduce the size of these factors. We found the use of a symmetric reverse Cuthill McKee ordering (symrcm, see [13]) to be helpful in this respect. If we perform a symmetric reverse Cuthill McKee ordering of $H + \theta$ or N beforehand (and solve the permuted problem explicitly), the size of preconditioners generated is shown in Table 4. However, it should be noted that while the effects of drop tolerances are dramatic in the number of conjugate gradient iterations required, their effect on the number of nonzeros in the factors is somewhat limited. We show two extremes in Table 4.

Note that only one ordering needs to be carried out, but that an incomplete factorization must be formed at every iteration of the interior point method. Since

TABLE 4. Results when using both primal and dual systems, with symrcm ordering and incomplete Cholesky factorization preconditioner.

Problem	Drop Tol	Time	CG Iter	Nnz
Ball	1e-3	0.06/0.22	60/330	543/703
	1e-4	0.05/0.11	32/80	563/811
	1e-8	0.04/0.08	12/24	575/945
Topple	1e-3	0.05/0.15	46/108	198/836
	1e-4	0.04/0.13	27/60	215/902
	1e-8	0.04/0.12	13/22	233/982

the number of interior point iterations and the number of conjugate gradient iterations remain (small and) unchanged, we believe that preconditioning with a very small drop tolerance is an efficient and effective way to solve the problem at hand.

We have also investigated the possibility of reusing the incomplete factors. However, while this has proven effective in other applications, it just served to dramatically increase the number of conjugate gradient iterations required in this application.

2.4. Effect of D

In both application examples, the matrix D has positive entries, ensuring that the problem at hand has a unique solution. We carried out a limited number of experiments where we set $D = 0$. Since in many cases the matrix J is quite rank-deficient, this can lead to substantial difficulties in our dual approach.

However, subject to the following caveats, all problems were solved by our approach. The first caveat is that in the dual approach, it may no longer be possible to form (2) since $D + \theta$ may no longer be invertible. To protect against this, whenever we solve (1) by first transforming it to (2), we perturb $D + \theta$ to force every entry to be at least 10^{-8}. The second caveat is that the resulting linear system (2) is very hard to solve and requires an exact factor. For even very small values of the drop tolerance the PCG method failed.

Thus to gain robustness in the method, we suggest using the adaptive primal/dual approach (possibly with very small perturbations), combined with a symmetric reverse Cuthill McKee ordering and solution using exact factorization. While it is possible (particularly in the primal system) to employ a drop tolerance in an incomplete Cholesky factorization preconditioner, the gain in terms of memory is very small compared to the increase in conjugate gradient iterations.

3. Other approaches

We experimented with several other approaches to determine their applicability, including OOQP [12], PATH [8,11], SEMI [19], L-BFGS-B [24], SPG [5] and TRON [17].

TABLE 5. PATH Results (the memory storage is for the LU factorization only).

Problem	Time	Iter	Mem
Ball	0.004/0.01	30/84	2230/4184
Topple	0.01/0.13	70/355	2574/19766

TABLE 6. SEMI Results (an iteration is a Newton step, not a LSQR step).

Problem	Time	Iter	Mem
Ball	0.01/0.14	6/15	2018/3824
Topple	0.1/1.77	11/30	2312/13900

The OOQP code did not solve all the problems in the suite so we have not reported results for that code here. The failures are due to difficulties in the factorization, as the matrix becomes more ill conditioned. However, we note that in the cases successfully solved, the number of interior point iterations used by OOQP was very similar to the code we outlined above.

3.1. PATH AND SEMI

We configured the PATH solver [8, 11] to act as a modified Lemke code (using a regular start instead of a ray start) with termination criteria of 10^{-4}. Note that each iteration corresponds to a pivot, implementable by a rank-1 update. While the use of a crash procedure [9] does reduce the number of pivots required to solve the problem, we have not reported these results here since we attempted to reduce the overall complexity of the approach. We present the results in Table 5. The iterations required by this approach are quite small, although for the larger test problems in the "topple" suite it increases to nearly 400. This approach could be implemented using the problem specific linear algebra techniques outlined in Section 2. We believe however, the method of Section 1 will perform better on this application due to the small fluctuation in iteration count in the context of the interior point method.

The semismooth approach described in [19] is similar to the interior point approach that was outlined in Section 1 in that it generates a small number of linear systems to solve, except that it typically destroys the symmetry properties of H. We used an option file that configured the code to carry out monotone linesearches without a crash procedure and using a Fischer merit function, as these options greatly improved the performance of the code on these problems. The implementation uses an incomplete LU factorization preconditioner to the LSQR iterative solver [21], which by default has a very small drop tolerance. We present the results (with termination criterion of 10^{-4}) in Table 6.

While the iteration count is very impressive, the number of nonzeros in the LU factors approaches 14,000 even when the Markovitz ordering is used. It remains

TABLE 7. L-BFGS-B results.

Problem	m	Time	Feval	Mem
Ball	3	0.075/0.180	1302/2287	822/1324
	5	0.066/0.149	858/1449	1309/2012
	10	0.097/0.683	757/1422	2947/4152
Topple	3	0.038/0.559	296/1332	855/3024
	5	0.035/0.453	207/666	1356/4392
	10	0.039/0.466	162/ 534	3027/8232

a topic for future research to determine if iterative techniques, more specialized reformulations or other orderings can reduce this memory requirement to an acceptable level. A (predictor-corrector) smoothing method implemented within Matlab performs very similarly to the SEMI code.

An advantage of the techniques of this section over the other ones outlined in this paper is that they would perform just as well on convex LCP's that are not derived directly as the optimality conditions of a convex quadratic optimization problem.

3.2. L-BFGS-B

The limited memory BFGS method [24] appears to be ideally suited to this application. This code has a limited memory requirement of $2mn + 4 * n + 12 * m^2 + 12m$ where n is the underlying problem dimension and m is a parameter that determines the number of BFGS corrections saved in memory. We used the code on the application supplied suite of test problems giving rise to the results shown in Table 7. The mean/max numbers of function evaluations and memory requirement are reported in columns Feval and Mem.

The run was terminated when the infinity norm of the projected gradient becomes smaller than a tolerance of 10^{-4}. We experimented with 3 values of the parameter m. Note that each iteration approximately solves a model problem determined by the current limited memory approximation of the Hessian matrix H.

Clearly, these results show that the benefit of $m > 3$ is not substantial in terms of computer time. Furthermore, the amount of memory required when $m = 3$ is much smaller, and is therefore preferred. However, even in this case, the method takes over $6n$ additional memory, and is thus more memory intensive that the problem specific approaches outlined above.

However, as a general purpose approach to these problems this technique has several benefits. Firstly, it does not require the matrix H to be formed explicitly but uses reverse communication to request objective function and gradient evaluations. Thus, H does not need to be formed, it can be applied using J, M^{-1} and D. Furthermore, the memory required is known in advance and is not affected by changes in density of J. However, the cost of this method is the number of iterations that are needed - the method often requires more than 1000 function evaluations with $m = 3$ and more than 500 with $m = 10$.

TABLE 8. Results when solving the linear systems (1) or (2) with a L-BFGS preconditioner.

Problem	m	Time	CG Iter	$2n(m+1)$
Ball	4	0.60/2.94	892/3400	623/1020
	8	0.65/3.57	769/3247	1121/1836
	16	0.62/3.94	598/2737	2118/3468
Topple	4	0.21/1.49	317/1634	295/1080
	8	0.22/1.66	267/1447	531/1944
	16	0.21/1.81	214/1206	1002/3672

3.3. L-BFGS PRECONDITIONING

Limited memory BFGS updates can also be used to compute a preconditioner for the conjugate gradient method [18]. The method is designed for solving a sequence of slowly varying linear systems. It uses information from the resolution of the current linear system to build an L-BFGS approximation matrix as a preconditioner for the next linear system. The benefits of this approach are the same as the L-BFGS-Bq algorithm, that is, low memory storage, an amount of memory known in advance and a Hessian matrix H that does not need to be formed but can simply be applied.

We experimented with a slightly different approach than the one described by [18]. Since the matrices of the linear systems are of the form $H + \theta$, where θ is diagonal and is the only term varying during the interior point iterations, we perform an L-BFGS approximation of H, say M, and use $M+\theta$ as preconditioner. Formulas and the algorithm for the computation of the matrix-vector products $(M+\theta)^{-1}v$ are obtained from a compact representation of M [7] combined with the Sherman-Morrison-Woodbury formula (see [1]). The memory storage requirement is of $2n(m+1)+O(m^2)$, where m is the number of vector pairs of length n kept in memory. The choice of the vector pairs at one iteration is carried out using the SAMPLE algorithm described by [18].

Table 8 shows the mean and maximum amount of computer time and number of conjugate gradient iterations that are required to solve the linear systems for each class of problems. We used the same primal-dual switching mechanism as described in Section 2.2. The parameter m indicates the number of vectors pairs kept in memory and the last column indicates the amount of additional memory required to store the vector pairs. We observed a linear decrease of the number of conjugate gradient iterations for increasing values of m, but the counterpart is that the computational cost increases at each iteration, so that the overall computer time does decrease. Note that the computer time are not directly comparable with those of the L-BFGS-B results, because of the use of different programming languages.

Table 9. SPG Results.

Problem	Time	Iter	Feval
Ball	0.21/1.00	4085/9995	6929/+15000
Topple	0.12/4.15	892/9990	1390/+15000

3.4. SPG

SPG [5] implements a gradient projection method combined with a nonmonotone line search. It is derived from the Barzilai-Borwein algorithm [2] and is globally convergent for the minimization of a nonlinear differentiable function over a convex set [4]. SPG is particularly well suited to our application because it needs only to store two gradient vectors and the projected direction of line search. Moreover the inner cost of one iteration is nearly reduced to the computation of two scalar products.

We used the Fortran 77 code supplied by the authors with the default parameters settings, except that we set the maximum number of function evaluations to 15000 and we used the same stopping criterion as with our experiments with L-BFGS-B, that is the infinity norm of the projected gradient must be smaller than 10^{-4}. Table 9 shows the computer time, the number of iterations and function evaluations to solve both problem suites.

SPG is a gradient algorithm and so we observed that sometimes a very great number of iterations are needed to obtain a sufficiently accurate solution. In particular we observed that to attain the given accuracy the number of function evaluations is greater than 15 000 for 23 problems. By relaxing the upper bound on the number of function evaluations, we observed more than 100 000 function evaluations for 6 problems. Even when the stopping tolerance is relaxed to 10^{-2}, the number of function evaluations is still greater than 15,000 for these problems. Though our implementation of the computation of function and gradient values is not optimal and can be certainly improved for a real application, we think that the inherent slow convergence of the gradient method is a serious drawback for this application.

3.5. TRON

TRON [17] implements a truncated Newton method for solving bound constrained problems. It uses a gradient projection method and a preconditioned conjugate gradient method with an incomplete Cholesky factorization to solve the linear systems. The factorization uses a technique proposed by Lin and Moré [16] and depends on a parameter p that specifies the amount of memory to store the preconditioner. An advantage of the preconditioning technique is to predict the amount of memory used by the application, what is not allowed with the drop tolerance strategy used in the preconditioning of our interior point approach.

We performed the experiments with the Fortran 77 code supplied by the authors. Table 10 shows the results for both suites of test problems. The TRON

TABLE 10. Tron results.

Problem	Iter	Time	CG Iter	Nnz
Ball	6.6/10	0.036/0.065	40/91	866/2129
Topple	5.6/13	0.040/0.204	27/ 286	1035/6867

approach seems to be very efficient in terms of number of iterations. The overall number of main iterations (Iter) is smaller than the one of the interior point approach. The computer time (Time) is of the same order as the interior point and L-BFGS-B, though it is not directly comparable to the former because of different programming languages. While the number of conjugate gradient iterations (CG Iter) remains reasonable for each problem, in particular for the "ball" suite, the amount of memory to store the preconditioner is prohibitive for the application. This is due to the fact that the number of nonzero elements of the preconditioner is equal to $n_{nz} + n(p+1)$ where n_{nz} is the number of nonzero elements of the strict lower triangular part of the factorized matrix and n is the problem size. For a part of the test problems, the Hessian matrix of the quadratic function is not sparse.

4. Conclusions

This paper has provided a case study of the solution of a particular suite of test problems arising from physical simulations in the video game industry. The paper has proposed an interior point method for solution and has investigated the use of iterative methods to solve the systems of linear equations that arise. Several other codes (PATH, SEMI, L-BFGS-B, SPG, TRON) also process all the models at hand, but are considered inferior to the interior point approach because they are too memory consuming or CPU time consuming or both. Obviously, the optimization that we have considered here, seem non-trivial to generalize to the complementarity setting. However, when direct methods are applied to solve (1) or (2), the symmetry of the underlying systems is no longer as crucial – in fact LU decomposition can be used in place of Cholesky factorization.

The interior point approach is theoretically guaranteed to process problems of the type described here and in practice takes very few iterations (linear solves) to generate accurate solutions. The method is easily generalizable to unsymmetric complementarity problems, relying on corresponding changes to the methods needed to solve the resulting linear systems.

We have proposed a primal/dual switching mechanism for solving the underlying linear systems in our method as a means to reduce memory requirements. This, coupled with a symmetric reverse Cuthill McKee ordering, allows all the systems to be solved with preconditioner memory requirements of less than 1000 double precision entries. The preconditioner recommended is an incomplete Cholesky factorization with very low drop tolerance. It is interesting to note that in many cases the nonzeros in the preconditioner are fewer than the nonzeros in the matrix J.

Unfortunately, the benefits of this preconditioner when coupled with the other features of our solution approach are limited, and direct solution does not take much more memory than the preconditioner, and can lead to more robustness on problems that are not strongly convex. In both cases, we acknowledge the potential drawback of needing to form the matrix used in (1) or (2) in order that the (incomplete) factorization can be carried out.

We believe that the use of iterative methods within the context of interior point methods remains a topic of future research. Newly developed codes such as GALAHAD [14] and KNITRO [6] for (quadratic and) nonlinear programming already incorporate conjugate gradient techniques for subproblem solution in large scale settings.

Limited memory solution of complementarity problems remains an open area for general problems. This paper has shown it to be difficult to use iterative linear equation solvers in the context of convex quadratic programs with simple bounds. Complementarity problems are considered more difficult than these problems for iterative solvers since they generate unsymmetric systems. We remain hopeful that careful study of particular applications, coupled with a more complete understanding of the interplay between theory and implementation will lead to advances in this area.

Acknowledgements. The authors wish to thank J.M. Martínez and an anonymous referee for their useful remarks and improvements.

References

[1] P. Armand and P. Ségalat, *A limited memory algorithm for inequality constrained minimization.* Technical Report 2003-08, University of Limoges (France) 2003.

[2] J. Barzilai and J.M. Borwein, Two-point step size gradient methods. *IMA J. Numer. Anal.* **8** (1988) 141–148.

[3] L. Bergamaschi, J. Gondzio and G. Zilli, Preconditioning indefinite systems in interior point methods for optimization. *Comput. Optim. Appl.* **28** (2004) 149–171.

[4] E.G. Birgin, J.M. Martínez and M. Raydan, Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Optim.* **10** (2000) 1196–1211.

[5] E.G. Birgin, J.M. Martinez and M. Raydan, Algorithm 813: Spg – software for convex-constrained optimization. *ACM Trans. Math. Software* **27** (2001) 340–349.

[6] R.H. Byrd, M.E. Hribar and J. Nocedal, An interior point algorithm for large-scale nonlinear programming. *SIAM J. Optim.* **9** (1999) 877–900 (electronic).

[7] R.H. Byrd, J. Nocedal and R.B. Schnabel, Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.* **63** (1994) 129–156.

[8] S.P. Dirkse and M.C. Ferris, The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optim. Meth. Software* **5** (1995) 123–156.

[9] S.P. Dirkse and M.C. Ferris, Crash techniques for large-scale complementarity problems, in *Complementarity and Variational Problems: State of the Art*, edited by M.C. Ferris and J.S. Pang. Philadelphia, Pennsylvania. SIAM Publications (1997) 40–61.

[10] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002) 201–213.

[11] M.C. Ferris and T.S. Munson, Complementarity problems in GAMS and the PATH solver. *J. Econ. Dyn. Control* **24** (2000) 165–188.

[12] E.M. Gertz and S.J. Wright, Object-oriented software for quadratic programming. *ACM Trans. Math. Software* **29** (2003) 58–81.

[13] John R. Gilbert, Cleve Moler and Robert Schreiber, Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl.* **13** (1992) 333–356.

[14] N.I.M. Gould, D. Orban and P.L. Toint, GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Software* **29** (2003) 353–372.

[15] C. Keller, N.I.M. Gould and A.J. Wathen, Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. Appl.* **21** (2000) 1300–1317.

[16] C.J. Lin and J. Moré, Incomplete Cholesky factorizations with limited memory. *SIAM J. Sci. Comput.* **21** (1999) 24–45.

[17] C.J. Lin and J. Moré, Newton's method for large bound-constrained optimization problems. *SIAM J. Optim.* **9** (1999) 1100–1127.

[18] J.L. Morales and J. Nocedal, Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Optim.* **10** (2000) 1079–1096.

[19] T.S. Munson, F. Facchinei, M.C. Ferris, A. Fischer and C. Kanzow, The semismooth algorithm for large scale complementarity problems. *INFORMS J. Comput.* **13** (2001) 294–311.

[20] M.F. Murphy, G.H. Golub and A.J. Wathen, A note on preconditioning for indefinite linear systems. *SIAM J. Sci. Comput.* **21** (2000) 1969–1972.

[21] C.C. Paige and M.A. Saunders, LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software* **8** (1982) 43–71.

[22] Y. Saad, *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston, Massachusetts (1996).

[23] S.J. Wright, *Primal–Dual Interior–Point Methods.* SIAM, Philadelphia, Pennsylvania (1997).

[24] C.Y. Zhu, R. Byrd, P. Lu and J. Nocedal, L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Trans. Math. Software* **23** (1997) 550–560.