Limited Memory Solution of Complementarity Problems arising in Video Games *

Michael C. Ferris† Andrew J. Wathen‡

July 28, 2002

1 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 behaviour 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 prob-

*This material is based on research partially supported by the Smith Institute, EPSRC Grant GR/M59044, the National Science Foundation Grant CCR-9972372, the Air Force Office of Scientific Research Grant F49620-01-1-0040, and the Guggenheim Foundation
†Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD Permanent address: Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706, USA
‡Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD
lems 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

$$\text{min}(\frac{1}{2}x^T Ax + v_0^T x : l \leq x \leq h).$$

Here, $x$ is the vector to be found, $l$, $h$ and $v_0$ are given vectors, the bounds hold componentwise, and $A$ is a symmetric positive semidefinite matrix of the form $A = JM^{-1}J^T + D$. The matrix $A$ is not computed explicitly, but is given by $J$, $M^{-1}$ and $D$. Here $M = \text{diag}(M_1, M_2, \ldots, M_k)$ is block diagonal and each $M_i = \text{diag}(m_i, m_i, m_i, I_i)$ is $6 \times 6$ with $m_i$ and $I_i$ being the mass and inertia tensor for the $i$th physical body. In fact, $x$ is the vector of the impulses at each physical contact, $J^T x$ is the vector of the impulses applied to the bodies, $M^{-1}J^T x$ is the vector of velocity changes of the bodies, $JM^{-1}J^T x$ 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”. 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 organised as follows. In Section 2 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 3 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 4 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.

2 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:

$$\min(\frac{1}{2}x^T H x + q^T x : x \in B)$$

where

$$B = \{ x : 0 \leq x_B \leq h, 0 \leq x_L, x_f free \}.$$ 

The first order conditions of this optimization problem are both necessary and sufficient and constitute the following complementarity problem:

$$\begin{bmatrix} H_{BB} & H_{BC} & H_{BF} \\ H_{CB} & H_{CC} & H_{CF} \\ H_{FB} & H_{FC} & H_{FF} \end{bmatrix} \begin{bmatrix} x_B \\ x_L \\ x_F \end{bmatrix} + \begin{bmatrix} q_B \\ q_L \\ q_F \end{bmatrix} \perp x_B \in [0, h] \quad x_L \geq 0 \quad x_F$$

where $\perp$ is interpreted componentwise to mean that if the variable is at lower bound then the corresponding function is nonnegative, if at upper bound the corresponding function is nonpositive and if the variable is strictly between its bounds, the function is zero.

Introducing an explicit multiplier $\xi$ for the upper bound constraint on $x_B$
we arrive at the following mixed complementarity problem:

\[
\begin{bmatrix}
H_{BB} & H_{BC} & H_{BF} & I \\
H_{LB} & H_{LC} & H_{LF} & 0 \\
H_{FB} & H_{FC} & H_{FF} & 0 \\
-I & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_B \\
x_C \\
x_F \\
\xi
\end{bmatrix}
+
\begin{bmatrix}
q_S \\
q_C \\
q_F \\
h
\end{bmatrix}
= 0
\]

In the standard manner, we introduce nonnegative variables \( s_B, s_C \) and \( w \) to generate the following problem:

\[
0 \leq s_B := H_B x + \xi + q_B \quad \perp \quad x_B \geq 0 \\
0 \leq s_C := H_C x + q_C \quad \perp \quad x_C \geq 0 \\
0 = H_F x + q_F \quad \perp \quad x_F \geq 0 \\
0 \leq w := -x_B + h \quad \perp \quad \xi \geq 0
\]

We apply the Primal-Dual Framework for LCP to this problem (see [11, pg 158–160]). The critical system of linear equations (using the standard capitalization notation) that must be solved at each iteration is:

\[
\begin{bmatrix}
H_{BB} & H_{BC} & H_{BF} & I \\
H_{LB} & H_{LC} & H_{LF} & 0 \\
H_{FB} & H_{FC} & H_{FF} & 0 \\
-I & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x_B \\
\Delta x_C \\
\Delta x_F \\
\Delta \xi
\end{bmatrix}
=
\begin{bmatrix}
\Delta q_S \\
\Delta q_C \\
\Delta q_F \\
\Delta h
\end{bmatrix}
\]

where \( \sigma \in [0, 1] \),

\[
\Phi = \begin{bmatrix}
(H x + q)_B + \xi - s_B \\
(H x + q)_C - s_C \\
(H x + q)_F \\
\mu
\end{bmatrix}
\]

and \( \mu = \frac{\| (H x + q)_F \| + x_T s_B + x_T s_C + w_T \xi}{\| \mathcal{F} \| + \| \mathcal{L} \| + 2 \| B \|} \).

We first eliminate \( \Delta w, \Delta s_B, \Delta s_C \) and \( \Delta \xi \) from this system to recover the following problem:

\[
\begin{bmatrix}
H_{BB} + X_C^{-1} S_B + W^{-1} E \\
H_{LB} \\
H_{FB}
\end{bmatrix}
\begin{bmatrix}
X_B & H_{BC} & H_{BF} \\
H_{LC} & X_C^{-1} S_C & H_{LF} \\
H_{FC} & H_{FF}
\end{bmatrix}
\begin{bmatrix}
\Delta x_B \\
\Delta x_C \\
\Delta x_F
\end{bmatrix}
= -r
\]

4
Figure 1: Interior Point Iterations per Problem: the problems are arbitrarily numbered along the horizontal axis

where $r$ is easy to calculate from $\Phi$ as

$$r = \begin{bmatrix}
(Hx + q)_B + \xi - \sigma \mu X^{-1}_B e - W^{-1}(E(h - x_B) - \sigma \mu e) \\
(Hx + q)_F - \sigma \mu X^{-1}_F e \\
(Hx + q)_F
\end{bmatrix}.$$  

It is clear that (2) involves simply a diagonal perturbation of the original $H$, which we call $\theta$. Once this system is solved, we can recover all the required values using back substitution in (1).

Some points of note. We choose $\sigma = 0.1$ for all our tests and initialize the method (in the translated problem) at a point where $x_i = 1$, $i \in \mathcal{L}$, $x_i = h_i/2$, $i \in \mathcal{B}$, $x_i = 0$, $i \in \mathcal{F}$. The slack variables $s_i = \max\{0.1, H_i, x + q_i\}$, $i \in \mathcal{L} \cup \mathcal{B}$, $s_i = H_i, x + q_i$, $i \in \mathcal{F}$ and $w = h/2$, $\xi_i = 0.1$. We terminate the interior point method when $\mu \leq 10^{-8}$.

We show on our two suites of test problems that the interior point method is an effective solution approach. Figure 1 shows the number of interior point iterations in each of the problem classes. While there is some variation over the suite of problems, the number of iterations is bounded above by 35 over all test problems and the vast majority of the solutions occur in less than 25 iterations of the interior point method. This approach appears very
promising provided that we can solve the linear systems within the time and space constraints imposed by the application.

3 Linear Algebra

3.1 Primal System

We investigate the solution of the system (2), which we will refer to as the primal system, using an iterative method. Symmetry and positive (semi-) definiteness allow us to use preconditioned conjugate gradients (PCG) to solve (2). The work involved in an iteration of this method is one matrix-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 \( \tilde{J}M^{-1}\tilde{J}^T + D \), 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 [1]. Figure 2 shows that all the resulting problems are then solved, each graph indicating the total number of iterations (matrix-vector products) per problem to solve all the systems generated by the interior point method on each problem class. We experimented with the different diagonal preconditioner \( \text{diag}(\sum_j |H_{j,\cdot}|) + \theta \) that attempts to incorporate the off diagonal entries in \( H \), but the results are not as good as those of Figure 2. While additional memory requirements are small, the numbers of matrix-vector products is considered unacceptable for the application.

There is evidently significant non-local coupling than can not be ac-
Figure 2: Total conjugate gradient iterations per problem when solving (2) with preconditioner \( \text{diag}(H) + \theta \)

counted for by these simple diagonal scalings. We therefore resort to a more sophisticated preconditioner, namely an incomplete Cholesky factorization with a drop tolerance (see for example [10]).

Figure 3 shows the number of conjugate gradient iterations and the number of nonzeros in the incomplete Cholesky factor (with drop tolerance \(10^{-4}\)) that are required to solve each problem using the primal system. 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.

### 3.2 Dual System

Alternative approaches for solving (2) stem from the equivalent augmented system:

\[
\begin{bmatrix}
M & \tilde{J}^T \\
\tilde{J} & -D - \theta
\end{bmatrix}
\begin{bmatrix}
\Delta y \\
\Delta x
\end{bmatrix}
= \begin{bmatrix} 0 \\ r \end{bmatrix}.
\]
Figure 3: Total conjugate gradient iterations and number of nonzeros (nnz) in factor per problem when solving (2) with an incomplete Cholesky factorization preconditioner (drop tolerance of $10^{-4}$)
Figure 4: Total conjugate gradient iterations per problem when solving (3) with preconditioner diag(N)

This symmetric and indefinite system might be preconditioned using for example the results of [5, 8] but without making further assumptions, no effective technique of either type was found in this case. However, just as the original system (2) (the primal system) results from an elimination of \( \Delta y \) using the first equation, an alternative is to use the dual system where we first eliminate \( \Delta x \) using the second equation, then solve for \( \Delta y \) and finally use back substitution to calculate \( \Delta x \):

\[
N := (M + J^T(D + \theta)^{-1}J)
\]

\[
N\Delta y = J^T(D + \theta)^{-1}r
\]

\[
\Delta x = (D + \theta)^{-1}(J\Delta y - r).
\]  \hspace{1cm} (3)

Two simple diagonal preconditioners can again be used, namely diag(N) and diag(\( \sum_j |N_{jj}| \)). The results for the first are given in Figure 4, and remain very similar for the second (which are not shown).

Figure 5 shows the number of conjugate gradient iterations and the number of nonzero entries in the incomplete Cholesky factor when applied to the system (3). The first point to note is that both of these are reduced on the “topple” problem as compared to the results of Figure 3. 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 the worst case of both iteration count and number of nonzeros is increased when using the dual approach.

This suggests using an approach that switches between the two systems depending on known problem characteristics. We recall 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 $J$ are not known, we choose to solve the dual system if $1.2 \text{rows}(\tilde{J}) \geq \text{cols}(\tilde{J})$. 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 Figure 6. Note that the "topple" problem is always solved using the dual approach, but the "ball" problem switches between the two systems.

### 3.3 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 Figure 3 (c) and (d) and Figure 5 (c) and (d). 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 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 Figure 7. Note that the maximum number of PCG iterations required to solve the problem (over all problems in the respective set) using these preconditioners is given in the caption as cgmax.

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 as Figure 8 and Figure 9.

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 the number of interior point iterations and the number of
Figure 5: Total conjugate gradient iterations and number of nonzeros (nnz) in factor per problem when solving (3) with an incomplete Cholesky factorization preconditioner (drop tolerance of $10^{-4}$)
Figure 6: Total Conjugate Gradient Iterations using both primal and dual systems with an incomplete Cholesky factorization with drop tolerance of $10^{-4}$

Figure 7: Number of nonzeros in Cholesky factor, using symrcm ordering, a drop tolerance of $10^{-4}$ and primal/dual switching
Figure 8: Number of nonzeros in Cholesky factor, using symrcm ordering, a drop tolerance of 0 and primal/dual switching

(a) Ball ($c_{g_{max}} = 25$)  
(b) Topple ($c_{g_{max}} = 35$)

Figure 9: Number of nonzeros in Cholesky factor, using symrcm ordering, a drop tolerance of $10^{-3}$ and primal/dual switching

(a) Ball ($c_{g_{max}} = 370$)  
(b) Topple ($c_{g_{max}} = 158$)
conjugate gradient iterations remains (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.

### 3.4 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 (3) since $D + \theta$ may no longer be invertible. To protect against this, whenever we solve (2) by first transforming it to (3), we perturb $D + \theta$ to force every entry to be at least $10^{-8}$. The second caveat is that the resulting linear system (3) 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.

### 4 Other Approaches

We experimented with several other approaches to determine their applicability, including OOQP, PATH, SEMI, and LBFGS. The OOQP code did not solve all the problems in the suite so we have not reported results for that code here; 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.

4.1 PATH and SEMI

We configured the PATH solver [2, 4] 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 [3] 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 this approach.

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 3. We believe however, the method of Section 2 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 [7] is similar to the interior point approach that was outlined in Section 2 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 [9], which by default has a very small drop tolerance. We present the results (with termination criterion of $10^{-4}$) as Figure 11. 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 a topic for future research to determine if iterative techniques, more specialised 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.

Figure 11: SEMI iterations
4.2 LBFGS

The limited memory BFGS method [12] appears to be ideally suited to this application. This code has a limited memory requirement of $2mn + 4n + 12m$ where $m$ is the memory size of the problem and $n$ is the underlying problem dimension. We used the code on the application supplied suite of test problems giving rise to the iteration counts of the method shown in Figures 12, 13 and 14.

We used the medium accuracy termination requirements suggested by the authors of the code, and 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 iteration count. 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 ap-
Figure 13: LBFGS iterations with $m = 5$

Figure 14: LBFGS iterations with $m = 10$
plied 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 iterations with $m = 3$ and more than 500 with $m = 10$.

Another possibility is to use the TRON code [6] that implements a truncated Newton approach for this problem with linear systems solved by pre-conditioned conjugate gradients using a limited memory incomplete Cholesky factorization. This has not yet been applied to our test set. Obviously, both these techniques are confined to the optimization framework that we have considered here and seem non-trivial to generalize to the complementarity setting. However, when direct methods are applied to solve (2) or (3), the symmetry of the underlying systems is no longer as crucial - in fact $LU$ decomposition can be used in place of Cholesky factorization.

5 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) also process all the models at hand, but are considered inferior to the interior point approach for a variety of reasons.

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 $J$. 

19
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 (2) or (3) 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 and KNITRO 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.

References


