High Performance Realtime Convex Solver for Embedded Systems

Ichitaro Yamazaki, Saeid Nooshabadi, *Senior Member, IEEE*, Stanimire Tomov, and Jack Dongarra.

***Abstract*— Convex optimization solvers for embedded systems find widespread use. This letter presents a novel technique to reduce the run-time of decomposition of KKT matrix for the convex optimization solver for an embedded system, by two orders of magnitude. We use the property that although the KKT matrix changes, some of its block sub-matrices are fixed during the solution iterations and the associated solving instances.**

***Index Terms*—Realtime embedded convex optimization solver, KKT**

I. Introduction

Recent convex optimization advances [1] [2] have enabled their use as realtime solvers for embedded systems [3] [4] [5] [6] [7] [8]. Two recent developments – CVXGEN [9] and ECOS [10] – provide frameworks for code generation for realtime convex optimization solvers for embedded systems. There are also some attempts in code generation for small-sized basic linear algebra opeations like vector-matrix multiplication [11]. CVXGEN takes a high-level description of the optimization problem, and employs the CVX technique of disciplined convex programming (DCP) [12] to generate a flat, and library-free C code. The generated code can be compiled into a high performance solver for the specific problem family (*e.g.*, all the matrices have the same sparsity structure). Though CVXGEN [9] solver obtains a high performance thanks to the generation of a flat code, to meet the strict realtime constraint enforced on the solution time, the size of the problem had to be limited to 100 or so variables.

In DCP a quadratic programming (QP) convex problem statement is transformed into a standard form [9] [13] as,

$$\begin{array}{ll} \text{minimize} & (1/2)x^{\mathrm{T}}Qx + q^{\mathrm{T}}x \\ \text{subject to} & Gx \preceq h \quad \text{and} \quad Ax = b, \end{array} \tag{1}$$

where vector variable $x \in \mathbf{R}^n$, $Q \in \mathbf{S}^n_+ \succeq 0$ (a square symmetric positive semidefinite matrix), $A \in \mathbf{R}^{m\times n}$, $Q \in \mathbf{R}^{p\times n}$, $b \in \mathbf{R}^m$, and $h \in \mathbf{R}^p$. With each instance of the solve, the solver goes through several solution iterations until it meets a certain level of pre-determined accuracy or the maximum number of iterations are reached. In each iteration the core of the optimization solver, where it spends almost its entire solution time, is the solution of a family of Karush Kuhn Tucker (KKT) [2] [14] linear system of equations $Kx = c$, whose coefficient matrices K all have the following block structure:

$$K = \left(\begin{array}{cc|cc} Q & 0 & G^{\mathrm{T}} & A^{\mathrm{T}} \\ 0 & S^{-1}Z & I_p & 0 \\ \hline G & I_p & 0 & 0 \\ A & 0 & 0 & 0 \end{array}\right), \tag{2}$$

where I_p is the identity matrix of size $(p \times p)$. $S = \mathrm{diag}(s) \in \mathbf{R}^{p\times p}$ and $Z = \mathrm{diag}(z) \in \mathbf{R}^{p\times p}$ are diagonal matrices (for the case of QP). Vectors $s \in \mathbf{R}^p$ and $z \in \mathbf{R}^p$, respectively, represent the slack variables and inequality multipliers in the KKT condition [14].

The solution to the linear system $Kx = c$ ($c \in \mathbf{R}^{n+m+2p}$) is found through the LDL$^{\mathrm{T}}$ decomposition of $PKP^{\mathrm{T}} = LDL^{\mathrm{T}}$ [15], where P is a permutation matrix, L a lower triangular matrix with diagonal elements equal to one, and D is a diagonal matrix. Using the LDL$^{\mathrm{T}}$ decomposition of K, the solution to $Kx = b$ is found through the sequence of forward, scaling, and back substitution.

The current generation of embedded convex optimization solvers, while good at taking the advantage of the structure of the problem family (*e.g.* sparsity), fail to take advantage of the fact that several blocks in the KKT matrix are fixed and do not change during the iterations of a given solve instance. In most practical signal processing applications, such as linearizing pre-equalizer, Kalman filtering, sliding window smoothing, and sliding window estimation [16], only the vectors q, h or b in (1) change from one solve instance to the next. These vectors only contribute to the make up of c vector in $Kx = c$. In the online array weight design adaptive filtering, only matrix G in (1) changes from one solve instance to the next. In all many cases that we have studies only the submatrix $S^{-1}Z$ changes with the solution iterations for each solve instance.

This letter proposes a technique to reduce the time to solve the family of the KKT linear systems by exploiting the fact that several blocks in the KKT matrix K are fixed and do not change during the solution iterations of a given problem instance.

II. Algorithms

We consider two cases: A) change in the KKT matrix that only persists during a single solution iteration in one instance of the solve, and B) change that persists across all solution iterations in one instance of the solve.

Saeid Nooshabadi is with the Department of Electrical and Computer Engineering, Michigan Tech, Houghton, MI, e-mail:{saeid}@mtu.edu; Ichitaro Yamazaki, Stanimire Tomov and Jack Dongara are with Innovative Computing Laboratory (ICL), the University of Tennessee, Knoxville, e-mail{iyamazak, tomov, dongarra}@icl.utk.edu

Manuscript received July 2016; revised:

A. Variable $S^{-1}Z$

Between the solution iterations, it is often that the blocks Q, A, and G are fixed, and only the diagonal block $S^{-1}Z$ changes. In order to take advantage of this feature we solve an equivalent, *implicitly*-reordered, linear system, $\widehat{K}\widehat{x} = \widehat{b}$, *i.e.*,

$$\left(\begin{array}{c|c|c|c} Q & A^{\mathrm{T}} & 0 & G^{\mathrm{T}} \\ \hline A & 0 & 0 & 0 \\ \hline 0 & 0 & S^{-1}Z & I_p \\ \hline G & 0 & I_p & 0 \end{array}\right) \left(\begin{array}{c} x_1 \\ \hline x_4 \\ \hline x_2 \\ \hline x_3 \end{array}\right) = \left(\begin{array}{c} c_1 \\ \hline c_4 \\ \hline c_2 \\ \hline c_3 \end{array}\right). \quad (3)$$

We emphasize that the matrix is not explicitly reordered, but we factorize and solve the linear system in the order given in (3) as follows:

a) Initial off-line setup: As part of our setup phase, we partially factorize the matrix such that $\widehat{K} = \widehat{L}\widehat{D}\widehat{L}^T$, where

$$\widehat{L} = \left(\begin{array}{c|c|c|c} L_{1,1} & 0 & 0 & 0 \\ \hline L_{2,1} & L_{2,2} & 0 & 0 \\ \hline 0 & 0 & I_p & 0 \\ \hline L_{4,1} & L_{4,2} & 0 & I_p \end{array}\right), \quad \widehat{D} = \left(\begin{array}{c|c|c|c} D_{11} & 0 & 0 & 0 \\ \hline 0 & D_{22} & 0 & 0 \\ \hline 0 & 0 & S^{-1}Z & I_p \\ \hline 0 & 0 & I_p & C \end{array}\right) \quad (4)$$

We compute this partial LDL$^{\mathrm{T}}$ factorization as follows:

1) We compute the LDL$^{\mathrm{T}}$ factorization of Q such that
$$Q = L_{1,1}D_{1,1}L_{1,1}^{\mathrm{T}}.$$
2) We compute the off-diagonal blocks in the first block column of $\widehat{L}$, *i.e.*,
$$L_{2,1} := A(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1} \quad \text{and} \quad L_{4,1} := G(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}.$$
3) We compute the LDL$^{\mathrm{T}}$ factorization of the second diagonal block to obtain $L_{2,2}$ and $D_{2,2}$,
$$\widetilde{K}_{2,2} = L_{2,2}D_{2,2}L_{2,2}^{\mathrm{T}},$$
where $\widetilde{K}_{2,2} := -(L_{2,1}D_{1,1}L_{2,1}^{\mathrm{T}})$.
4) We compute the off-diagonal block in the second block column of $\widehat{L}$, *i.e.*,
$$L_{4,2} := \widetilde{K}_{4,2}(D_{2,2}L_{2,2}^{\mathrm{T}})^{-1},$$
where $\widetilde{K}_{4,2} := -(L_{4,1}D_{1,1}L_{2,1}^{\mathrm{T}})$.
5) We compute the last block of $\widehat{D}$,
$$C := -(L_{4,1}D_{1,1}L_{4,1}^{\mathrm{T}}) - (L_{4,2}D_{2,2}L_{4,2}^{\mathrm{T}}).$$

b) On-line Factorization: At each step of the convex optimization solution, we can cheaply factorize the last two diagonal blocks of $\widehat{D}$ with the new $S^{-1}Z$, *i.e.*,

$$\begin{cases} D_{3,3} & := & S^{-1}Z \\ L_{4,3} & := & D_{3,3}^{-1} = Z^{-1}S, \end{cases}$$

$L_{4,4}$ and $D_{4,4}$ are computed from the LDL^T factorization of the last diagonal block as,

$$\widetilde{C} = L_{4,4}D_{4,4}L_{4,4}^{\mathrm{T}},$$

where $\widetilde{C} := C - (Z^{-1}S)$. Since both Z and S are diagonal, the most computationally expensive part of the on-line factorization is the LDL^T factorization of $\widetilde{C}$. In other words, in the on-line factorization stage we only factorize a matrix of dimension S, and this algorithm allows us to solve the realtime convex optimization where p, instead of $n + m + 2p$, is the largest dimension of the matrix that must be factorized within the realtime constraint.

In this letter, we only consider the QP problem, *i.e.*, diagonal $S^{-1}Z$. However, the algorithm can be trivially extended to a more general case such as the second order cone programming (SOCP) [2], where $S^{-1}Z$ is of the form $S^{-1/2}ZS^{-1/2}$.

B. Variable G

For the case that the submatrix G changes between the solve instances, but stays fixed during the solution iterations of one instance of the solve, we introduce an intermediate off-line update step, where we only recompute the steps that use G.

a) Initial off-line setup:

1) We compute the LDL$^{\mathrm{T}}$ factorization of Q such that
$$Q = L_{1,1}D_{1,1}L_{1,1}^{\mathrm{T}}.$$
2) We compute the off-diagonal blocks in the first block column of $\widehat{L}$, *i.e.*,
$$L_{2,1} := A(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}.$$
3) We compute the LDL$^{\mathrm{T}}$ factorization of the second diagonal block to obtain $L_{2,2}$ and $D_{2,2}$,
$$\widetilde{K}_{2,2} = L_{2,2}D_{2,2}L_{2,2}^{T},$$
where $\widetilde{K}_{2,2} := -(L_{2,1}D_{1,1}L_{2,1})$.
4) We partially compute the off-diagonal block in the second block column of $\widehat{L}$, *i.e.*,
$$H := -D_{1,1}L_{2,1}^{\mathrm{T}}(D_{2,2}L_{2,2})^{-1}.$$

b) Off-line update: To solve each convex optimization problem with a new submatrix G, we complete the off-line factorization as follows:

1) We compute the off-diagonal blocks in the first block column of $\widehat{L}$, *i.e.*,
$$L_{4,1} := G(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}.$$
2) We compute the off-diagonal block in the second block column of $\widehat{L}$, *i.e.*,
$$L_{4,2} := L_{4,1}H.$$
3) We compute the last block of $\widehat{D}$,
$$C := -(L_{4,1}D_{1,1}L_{4,1}^{\mathrm{T}}) - (L_{4,2}D_{2,2}L_{4,2}^{\mathrm{T}}).$$

c) On-line Factorization: Finally, with each solution iteration with the new diagonal submatrix $S^{-1}Z$, we can cheaply complete the factorization as in Section II-A.

III. Experiments

All our experiments were conducted on MacBook Pro®, using just one core of 2.7 GHz Intel® Core™ i7. Our codes were compiled using `gcc` of Apple® LLVM version 5.1 with the optimization flag `-Os`. For our experiments, we linked our codes to BLAS and LAPACK provided in the Apple's Accelerate Framework [17], but on any other embedded system, it could be statically linked to open-source reference implementations of BLAS [18] and LAPACK [19]. The test problem that we used for this study is the largest size standard convex optimization problem in (1) that CVXGEN can handle. The KKT matrix is of dimension 131, and the respective dimensions of the submatrices Q, Z, G, and A are 95×95, 12×12, 95×12, and 95×12.

A. Results with variable $S^{-1}Z$

Table I presents the run times of several techniques for factorizing the KKT matrix K when only the submatrix $S^{-1}Z$ changes. It also shows the run times for the solve of the linear system $Kx = c$ through the forward and backward substitutions. The performance results shown are for the case where the submatrix Q is dense.

For this particular test matrix, the LAPACK's solvers were faster than the CVXGEN generated solver even though they ignore any structure of the KKT matrix and perform dynamic pivoting to ensure the numerical stability of the factorization. The CVXGEN generated solver uses regularization [9] that avoids the small diagonal entries through small shifts during the factorization, and is typically less stable but more efficient than the dynamic pivoting.

Our off-line factorization that takes advantage of the structure was slower than "`dsytrf`" of LAPACK. This is because our implementation is based on LAPACK that does not provide flexible enough interface to take full advantage of the symmetry in the KKT matrix. For example, LAPACK does not provide a subroutine to solve only with the lower-triangular matrix L of the LDL^T factorization. Hence, we compute the non-symmetric LDU factorization of the KKT matrix, $\widehat{K} = \widehat{L}\widehat{D}\widehat{U}$, where $\widehat{L}$ and $\widehat{D}$ have the same block structures as those in (4), and $\widehat{U}$ has the same structure as that of $\widehat{L}^T$. For this, at Step 2 of initial off-line setup in Section II-A, we compute,

$$L_{2,1} := AU_{1,1}^{-1} \quad \text{and} \quad L_{4,1} := GU_{1,1}^{-1},$$

where $U_{1,1} = Q$, and in addition, we let $L_{1,1} = I$, $L_{2,2} = I$, $D_{1,1} = I_n$, $D_{2,2} = I_m$, and $U_{1,2} = A^T$ and $U_{1,4} = G^T$. We use the LDL^T factorization of Q to apply $U_{1,1}^{-1}$.

Then at Steps 3 and 4, we compute,

$$L_{4,2} := -(L_{4,1}U_{1,2})(U_{2,2})^{-1} \text{ and } U_{2,4} := L_{2,2}^{-1}(L_{2,1}U_{1,4}),$$

TABLE I
Run Times for facotorizing and solving the KKT linear system with variable $S^{-1}Z$ and Dense Q (in seconds).

Here unroll is the generated solver from CVXGEN, `dgetrf` and `dsytrf` are the LAPACK's unsymmetric and symmetric solvers applied on the KKT matrix, and `dsgesv` is the LAPACK's mixed-precision solver, where the matrix is factorized in the single precision, and the iterative refinements are used to compute the solution in double precision.

Technique	Factor	Solve
unroll	$5.25 \cdot 10^{-4}$	$1.70 \cdot 10^{-5}$
`dgetrf`	$3.03 \cdot 10^{-4}$	$2.20 \cdot 10^{-5}$
`dsytrf`	$1.78 \cdot 10^{-4}$	$1.90 \cdot 10^{-5}$
`dsgesv`	$2.71 \cdot 10^{-4}$ (2 iters)	
fixed Q,A,G: initial off-line setup		
total	$1.85 \cdot 10^{-4}$	
▷ step 1	$7.40 \cdot 10^{-5}$	
▷ step 2	$3.90 \cdot 10^{-5} + 4.00 \times 10^{-5}$	
▷ step 3	$1.10 \cdot 10^{-5} + 3.00 \times 10^{-6}$	
▷ step 4	$3.00 \cdot 10^{-6} + 3.00 \times 10^{-6}$	
▷ step 5	$3.00 \cdot 10^{-6} + 2.00 \times 10^{-6}$	
on-line factor and solve	$4.00 \cdot 10^{-6}$	$2.30 \cdot 10^{-5}$

where $U_{2,2} := -(L_{2,1}U_{1,2})$. Finally, at Step 5 we compute $C := -(L_{4,1}U_{1,4}) - (L_{4,2}U_{2,4})$.

Though our particular implementation of the solver does not exploit the symmetry in the KKT matrix, by taking advantage of the fixed submatrices, it significantly reduces the on-line factorization time, with speedups of about $131.3\times$ and $44.5\times$, respectively, over the CVXGEN generated solver, and the LAPACK's `dsytrf` solver.

Table II shows the similar results, where the submatrix Q is considered to be diagonal. As the baseline performance to compare against, we used the CVXGEN ordering shown in (2), and formed the Schur complement consisting of the last two diagonal blocks. Since Q is diagonal, the Schur complement can be computed with the combination of simple diagonal scaling and the matrix-matrix multiply "`dgemm`" from BLAS. Then, we used different LAPACK solvers on the Schur complement. Since the submatrix Q does not have to be factorized, the performance improvement obtained by taking advantage of the fixed submatrices Q, A, and G is smaller than those in Table I. However, we still obtained the speedup of about $6.5\times$ over our baseline implementation using `dsytrf`.

B. Results with variable G

Table III shows the performance of the off-line update when only the submatrix G changes between the instances of the solve. Just like in Section III-A, our implementation of the initial off-line setup and off-line update compute the LDU factorization. Overall, even with these extra operations required for the non-symmetric factorization, we see that taking the advantage of the structure reduces the time of the initial off-line setup in Table I by a factor of about $2.6\times$.

IV. Future Work

As a next logical step we will consider integration of the proposed solver into a code generation platform such as CVXGEN or ECOS. We also plan to modify and tune the

TABLE II
RUN TIMES FOR FACTORIZING AND SOLVING THE KKT SYSTEM WITH DIAGONAL Q (IN SECONDS).

On the first three rows `dgemm` is used to form the Schur complement of last two diagonal blocks of the KKT matrix in (2). Next LAPACKS's `dgetri`, `dgetrf`, or `dsytrf` is used to solve the dense Schur complement system.

Technique	Factor	Solve
`dgetri` on Schur comp		
total	$3.40 \cdot 10^{-5}$	$1.80 \cdot 10^{-6}$
`dgemm`	$1.70 \cdot 10^{-5}$	
`dgetrf`	$5.00 \cdot 10^{-6}$	
`dgetri`	$7.00 \cdot 10^{-6}$	
`dgetrf` on Schur comp		
total	$2.50 \cdot 10^{-5}$	$3.00 \cdot 10^{-6}$
`dgemm`	$1.50 \cdot 10^{-5}$	
`dgetrf`	$5.00 \cdot 10^{-6}$	
`dsytrf` on Schur comp		
total	$2.60 \cdot 10^{-5}$	$5.00 \cdot 10^{-6}$
`dgemm`	$1.50 \cdot 10^{-5}$	
`dsytrf`	$6.00 \cdot 10^{-6}$	
fixed Q,A,G: initial off-line setup		
total	$3.90 \cdot 10^{-5}$	
▷ step 1	0.00	
▷ step 2	$1.00 \cdot 10^{-6} + 1.00 \times 10^{-6}$	
▷ step 3	$1.30 \cdot 10^{-5} + 4.00 \times 10^{-6}$	
▷ step 4	$5.00 \cdot 10^{-6} + 5.00 \times 10^{-6}$	
▷ step 5	$5.00 \cdot 10^{-6} + 1.00 \times 10^{-6}$	
on-line factor and solve	$4.00 \cdot 10^{-6}$	$6.00 \cdot 10^{-6}$

TABLE III
RUN TIMES (IN SECONDS) WITH VARIABLE G AND DENSE Q

Technique	Factor
initial off-line setup	
total	$1.21 \cdot 10^{-4}$
▷ step 1	$7.40 \cdot 10^{-5}$
▷ step 2	$3.70 \cdot 10^{-5}$
▷ step 3	$1.00 \cdot 10^{-5} + 2.00 \times 10^{-6}$
▷ step 4	– –
off-line update	
total	$7.00 \cdot 10^{-5}$
▷ step 1	$4.35 \cdot 10^{-5}$
▷ step 2	$1.10 \cdot 10^{-5} + 3.00 \times 10^{-6}$
▷ step 3	$5.00 \cdot 10^{-6} + 1.00 \times 10^{-6}$

LAPACK subroutines for specializing them to factorize the KKT matrix.

Here, we only considered diagonal $S^{-1}Z$ where the KKT duality gap condition requires $S^{-1}Z = 0$ [14]. We typically choose a value of $||S^{-1}Z||_2 \leq 10^{-6}$ as a the stopping criterion. With the small change in the numerical values of the diagonal elements of $S^{-1}Z$, there is a possibility to update the block matrices $L_{4,4}$ and $D_{4,4}$ *in situ* in parallel, with no need for re-factorization.

Further, simplification to the decomposition can be made considering some special properties of some of the submatrices. For example in some applications we have $Q = H^{-\mathrm{T}}H$ where H is a trapezoidal matrix.

V. CONCLUSION

In this letter a reordering technique for the decomposition of KKT matrix for the convex optimization solver for an embedded system that reduces the run time by two orders of magnitude was proposed. This technique takes advantage of the fact that many block matrices in the KKT matrix do not change during the iterations of one instance of the solve.

REFERENCES

[1] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, Belmont, MA., 2015.

[2] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, Cambridge, 2012.

[3] J. Mattingley and S. Boyd, "Realtime convex optimization in signal processing," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 50–61, May 2010.

[4] E. Hazen, "Efficient algorithms for online convex optimization and their applications," PhD dissertation, Department of Computer Science, Princeton University, Tech. Rep., 2006.

[5] I. Das and J. W. Fuller, "Real-time quadratic programming for control of dynamical systems," US Patent 7328074, Tech. Rep., 2008.

[6] S. P. Boyd, L. Vandenberghe, and M. Grant, "Efficient convex optimization for engineering design," in *Proceedings IFAC Symposium on Robust Control Design*, Rio de Janeiro, Brazil, Sept. 2004.

[7] D. Burns, W. Weiss, and M. Guay, "Realtime setpoint optimization with time-varying extremum seeking for vapor compression systems," in *American Control Conference (ACC), 2015*, July 2015, pp. 974–979.

[8] G. M. Shaver, "Stability analysis of residual-affected HCCI using convex optimization," *Control Engineering Practice*, vol. 17, no. 12, pp. 1454–1460, May 2009.

[9] J. Mattingley and S. Boyd, "CVXGEN – code generation for convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, March 2012. [Online]. Available: http://http://cvxgen.com/

[10] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference (ECC)*, Zurich, Switzerland, July 2013, pp. 3071–3076. [Online]. Available: https://www.embotech.com/ECOS

[11] A. Stojanov, G. Ofenbeck, T. Rompf, and M. Püschel, "Abstracting vector architectures in library generators: Case study convolution filters," in *ACM International Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY)*, 2014, p. 14.

[12] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: From Theory to Implementation (Nonconvex Optimization and Its Applications)*, L. Liberti and N. Maculan, Eds. Springer Science and Business Media, 2006, pp. 155–210.

[13] M. Grant and S. Boyd, "CVX: matlab software for disciplined convex programming version 2.1," 2015. [Online]. Available: http://cvxr.com/cvx

[14] J. Nocedal and S. Wright, *Numerical Optimization, 2nd ed.* Springer Verlag, 2006.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, 3rd Ed.* Cambridge University Press, Cambridge, 20007.

[16] J. Mattingley and S. Boyd, "Automatic code generation for real-time convex optimization," in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge: Cambridge University Press, 2009, ch. 1, pp. 1–41.

[17] Apple Inc, "Accelerate Framework," 2016. [Online]. Available: https://developer.apple.com/reference/accelerate

[18] "BLAS Basic Linear Algebra Subprograms," 2012. [Online]. Available: http://www.netlib.org/blas/

[19] "APACKLinear Algebra PACKage," 2016. [Online]. Available: http://www.netlib.org/lapack/