SIAM J. SCI. COMPUT.
Vol. 44, No. 1, pp. A103–A127

LARGE-SCALE OPTIMIZATION WITH LINEAR EQUALITY CONSTRAINTS USING REDUCED COMPACT REPRESENTATION[*]

JOHANNES J. BRUST[†], ROUMMEL F. MARCIA[‡], COSMIN G. PETRA[§], AND MICHAEL A. SAUNDERS[¶]

Dedicated to Dr. Oleg Burdakov, 1953–2021

Abstract. For optimization problems with linear equality constraints, we prove that the (1,1) block of the inverse KKT matrix remains unchanged when projected onto the nullspace of the constraint matrix. We develop *reduced compact representations* of the limited-memory inverse BFGS Hessian to compute search directions efficiently when the constraint Jacobian is sparse. Orthogonal projections are implemented by a sparse QR factorization or a preconditioned LSQR iteration. In numerical experiments two proposed trust-region algorithms improve in computation times, often significantly, compared to previous implementations of related algorithms and compared to IPOPT.

Key words. large-scale optimization, compact representation, trust-region method, limited memory, LSQR, sparse QR

AMS subject classifications. 68Q25, 68R10, 68U05

DOI. 10.1137/21M1393819

1. Introduction.

Linear equality constrained minimization problems are formulated as

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x) \quad \text{subject to} \quad Ax = b, \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $A \in \mathbb{R}^{m\times n}$. We assume that the number of variables n is large, $g(x) = \nabla f(x)$ is available, A is sparse, and that the initial guess x_0 is feasible: $Ax_0 = b$. If A has low rank, one can obtain a full-rank matrix by deleting rows in A that correspond to small diagonals of the triangular matrix in a sparse QR factorization of $A^\top$. Our methods here use the rank information contained in sparse QR factors, and thus we assume that A has full rank until implementation

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section January 27, 2021; accepted for publication (in revised form) September 8, 2021; published electronically January 13, 2022. LLNL Release Number: LLNL-JRNL-818401. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-accessplan
https://doi.org/10.1137/21M1393819

Funding: This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract DE-AC02-06CH11357 at Argonne National Laboratory. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. The work of the second author was partially supported by NSF grant IIS 1741490.

[†]Department of Mathematics, University of California San Diego, La Jolla, CA 92093 USA (formerly Argonne National Laboratory) (jjbrust@ucsd.edu).
[‡]Department of Applied Mathematics, University of California Merced, Merced, CA 95343 USA (rmarcia@ucmerced.edu).
[§]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550 USA (petra1@llnl.gov).
[¶]Department of Management Science and Engineering, Stanford University, Stanford, CA 94305-4121 USA (saunders@stanford.edu).

details are described in Appendix B. For large problems, computing the Hessian $\nabla^2 f(x) \in \mathbb{R}^{n\times n}$ is often not practical, and we approximate this matrix using a limited-memory BFGS (Broyden–Fletcher–Goldfarb–Shanno [2, 16, 20, 29]) quasi-Newton matrix $B_k \approx \nabla^2 f(x_k)$. Starting from x_0, we update iterates according to $x_{k+1} = x_k + s_k$. The step s_k is computed as the solution of a quadratic trust-region subproblem, in which the quadratic objective is defined as $q(s) \equiv s^\top g_k + \frac{1}{2} s^\top B_k s$ with $g_k \equiv g(x_k)$. For a given trust-region radius $\Delta > 0$ and norm $\|\cdot\|$, the trust-region subproblem is

$$\underset{\|s\|\le\Delta}{\text{minimize}}\; q(s) \quad \text{subject to} \quad As = 0, \tag{1.2}$$

which ensures that each search direction is in the nullspace of A, and thus each iterate x_k is feasible.

1.1. Background. Large problems of the form (1.1) are the focus of recent research because large statistical- and machine-learning problems can be cast in this way. As such, (1.1) constitutes the backbone of the alternating direction method of multipliers [1], with applications to optimal exchange problems, consensus and sharing problems, support-vector machines, and more. Recent work [18] emphasizes methods that use gradients of f and suggest accelerations via quasi-Newton approximations. Quasi-Newton methods estimate Hessian matrices using low-rank updates at each iteration (typically rank-1 or rank-2). Starting from an initial matrix, the so-called *compact representation* of quasi-Newton matrices [8] is a matrix representation of the recursive low-rank updates. Because the compact representation enables effective limited-memory implementations, which update a small number of previously stored vectors, these methods are well suited to large problems. Trust-region and line-search methods are standard for determining search directions for smooth problems, and each approach has its own merits. Combinations of trust-region methods and quasi-Newton compact representations have been developed in [3, 4, 5, 7]. Widely used quasi-Newton line-search methods are [9, 24, 31, 32]. The main ideas in this article are applicable to both trust-region and line-search methods.

1.2. Compact representation. A storage-efficient approach to quasi-Newton matrices is the compact representation of Byrd, Nocedal, and Schnabel [8], which represents the BFGS matrices in the form

$$B_k = \gamma_k I + J_k M_k J_k^\top \tag{1.3}$$

with scalar $\gamma_k > 0$. The history of vectors $\{s_k\} = \{x_{k+1} - x_k\}$ and $\{y_k\} = \{g_{k+1} - g_k\}$ is stored in rectangular $S_k \equiv \begin{bmatrix} s_0, \ldots, s_{k-1} \end{bmatrix} \in \mathbb{R}^{n\times k}$ and $Y_k \equiv \begin{bmatrix} y_0, \ldots, y_{k-1} \end{bmatrix} \in \mathbb{R}^{n\times k}$. The matrices

$$J_k \equiv \begin{bmatrix} S_k & Y_k \end{bmatrix}, \tag{1.4}$$

$$S_k^\top Y_k \equiv L_k + D_k + \bar{T}_k, \tag{1.5}$$

$$M_k \equiv -\begin{bmatrix} \delta_k S_k^\top S_k & \delta_k L_k \\ \delta_k L_k^\top & -D_k \end{bmatrix}^{-1} \tag{1.6}$$

are defined with $\delta_k = 1/\gamma_k$, where L_k and $\bar{T}_k$ are the strictly lower and upper triangular parts of $S_k^\top Y_k$ and D_k is the diagonal. For large problems, limited-memory versions store only a small subset of recent pairs $\{s_i, y_i\}_{i=k-l}^{k-1}$, resulting in storage-efficient matrices $J_k \in \mathbb{R}^{n\times 2l}$ and $M_k \in \mathbb{R}^{2l\times 2l}$ where $l \ll n$. Following Byrd, Nocedal, and Schnabel [8, Theorem 2.2], the inverse BFGS matrix has the form

$$B_k^{-1} = \delta_k I + J_k W_k J_k^\top, \tag{1.7}$$

where $W_k \in \mathbb{R}^{2l\times 2l}$ is given by

$$W_k = \begin{bmatrix} T_k^{-\top}(D_k + \delta_k Y_k^\top Y_k)T_k^{-1} & -\delta_k T_k^{-\top} \\ -\delta_k T_k^{-1} & 0_{l\times l} \end{bmatrix}. \tag{1.8}$$

The diagonal matrix D_k (and hence the upper triangular matrix $T_k \equiv D_k + \bar{T}_k$) are nonsingular as long as B_k is also.

1.3. Outline. Section 2 describes our contributions in the context of large problems, while section 3 motivates our proposed representations. Section 4 develops the reduced compact representation and updating techniques that enable efficient implementations. Section 5 describes computations of orthogonal projections and the trust-region strategy for optimization. Section 6 gives an efficient method when an ℓ_2-norm trust-region subproblem is used. Sections 7 and 8 develop an effective factorization and a method that uses a shape-changing norm in the trust-region subproblem. Numerical experiments are reported in section 9, and conclusions are drawn in section 10.

2. Contributions.

The first-order necessary conditions for the solution of problem (1.2) without the norm constraint are characterized by the linear system

$$\begin{bmatrix} B_k & A^\top \\ A & 0_{m\times m} \end{bmatrix} \begin{bmatrix} s_E \\ \lambda_E \end{bmatrix} = \begin{bmatrix} -g_k \\ 0_m \end{bmatrix}, \tag{2.1}$$

where $\lambda_E \in \mathbb{R}^m$ is a vector of Lagrange multipliers and s_E denotes the "equality" constrained minimizer of (1.2). Adopting the naming convention of [27, section 16.1, page 451], we refer to (2.1) as the KKT system (a slight misnomer, as use of the system for the equality constrained setting predates the work of Karush, Kuhn, and Tucker). For large n, compact representations of the (1,1) block in the inverse KKT matrix were recently proposed by Brust, Marcia, and Petra [6]. Two limited-memory trust-region algorithms, LTRL2-LEC and LTRSC-LEC (which we refer to as TR1 and TR2 in the numerical experiments in section 9), use these representations to compute search directions efficiently when A has relatively few rows. This article develops efficient algorithms when the number of equality constraints is large and the constraint matrix is sparse. In particular, by exploiting the property that part of the solution to the KKT system is unaltered when it is projected onto the nullspace of A, we develop *reduced compact representations*, which need a small amount of memory and lead to efficient methods for solving problems with many constraints (large m and n) and possibly many degrees of freedom (large $n - m$). In numerical experiments when solving large problems, the proposed methods are often significantly more efficient than both our previous implementations and IPOPT [30].

3. Motivation.

The solution s_E in (2.1) can be computed from only the (1,1) block of the inverse KKT matrix, as opposed to both the (1,1) and (1,2) blocks, because of the zeros in the right-hand side. Let V_k be the (1,1) block of the inverse KKT matrix (obtained, for example, from a block LDU factorization). It is given by

$$V_k \equiv (B_k^{-1} - B_k^{-1}A^\top(AB_k^{-1}A^\top)^{-1}AB_k^{-1}) \tag{3.1}$$

and then $s_E = -V_k g_k$. At first sight the expression in (3.1) appears to be expensive to compute because of the multiple inverse operations and matrix-vector products. However, as $B_k^{-1} = \delta_k I + J_k W_k J_k^\top$, we can exploit computationally useful structures. Specifically, with $G_k \equiv (AB_k^{-1}A^\top)^{-1}$ and $C_k \equiv AJ_kW_k$, [6, Lemma 1] describes the expression

$$V_k = \delta_k I + \begin{bmatrix} A^\top & J_k \end{bmatrix} \begin{bmatrix} -\delta_k^2 G_k & -\delta_k G_k C_k \\ -\delta_k C_k^\top G_k & W_k - C_k^\top G_k C_k \end{bmatrix} \begin{bmatrix} A \\ J_k^\top \end{bmatrix}. \tag{3.2}$$

For large n, once the components of the middle matrix in (3.2) are available, this compact representation of V_k enables efficient computation of a matrix-vector product $V_k g_k$, hence the solution of (2.1), and an economical eigendecomposition $V_k = U \Lambda U^\top$. However, unless m is small (there are few rows in A), multiplying with the $(m+2l) \times (m+2l)$ middle matrix is not practical.

With large n and m in mind, we note that the solution s_E is unchanged if instead of g_k a projection of this vector onto the nullspace of A is used, or if s_E is projected onto the nullspace of A. This is a consequence of the properties of V_k. To formalize these statements, let the orthogonal projection matrix onto null(A) be $P = I_n - A^\top(AA^\top)^{-1}A$. Since the columns of the (1,1) block of the inverse from (2.1) (namely, columns of V_k) are in the nullspace of A, the orthogonal projection onto null(A) acts as an identity operator on the vector space spanned by V_k:

$$V_k = V_k P = P^\top V_k = P^\top V_k P. \tag{3.3}$$

Relation (3.3) can equivalently be derived from (3.1), the expression for P, and the equality $V_k A^\top = 0$. The methods in this article are based on representations of projected matrices $P^\top V_k P \in \mathbb{R}^{n \times n}$, whose properties enable desirable numerical advantages for large n and m. Instead of multiplying with the possibly large $G_k \in \mathbb{R}^{m \times m}$ and $C_k \in \mathbb{R}^{m \times 2l}$ in (3.2), we store the matrices $S_k \in \mathbb{R}^{n \times l}$ and $Z_k \equiv PY_k \in \mathbb{R}^{n \times l}$ and small square matrices that depend on the memory parameter l but not on m. The columns of Z_k are defined as $z_k = Py_k = P(g_{k+1} - g_k)$, and they are contained in the nullspace of A.

With (3.1) and (3.2) we motivated the solution of (1.2) without the norm constraint (giving the equality constrained step s_E). Computing s_E is important for the implementation of practical algorithms, but it is even more important to solve (1.2) efficiently with the norm constraint. In section 6, using the ℓ_2-norm, we develop a modified version of V_k as a function of a scalar parameter $\sigma > 0$, i.e., $V_k(\sigma)$. In sections 7 and 8, we describe how the structure of V_k can be exploited to compute an inexpensive eigendecomposition that, when combined with a judiciously chosen norm (the shape-changing infinity norm from [7, section 4.2.1]), provides a search direction by an analytic formula. Note that the representation of V_k is not specific to the limited-memory BFGS (L-BFGS) matrix, and other compact quasi-Newton matrices could be used (Byrd, Nocedal, and Schnabel [8], DeGuchy, Erway, and Marcia [14]).

4. Reduced compact representation.

This section describes a computationally effective representation of (3.3), which we call the *reduced compact representation* (RCR). In section 4.1, the RCR is placed into historical context with reduced Hessian methods. Subsequently, sections 4.2–4.4 develop the specific formulas that enable effective computations.

4.1. Reduced Hessian.

The name *reduced compact representation* is related to the term *reduced Hessian* [19], where $\hat{Z} \in \mathbb{R}^{n \times (n-m)}$ denotes a basis for the nullspace of A (satisfying $A\hat{Z} = 0$). In turn, $\hat{Z}$ defines the so-called reduced Hessian matrix as $\hat{Z}^\top \nabla^2 f_k \hat{Z}$ or $\hat{Z}^\top B_k \hat{Z}$. In order to compute an equality constrained step s_E, a reduced Hessian method solves $(\hat{Z}^\top B_k \hat{Z})\hat{s}_E = -\hat{Z}^\top g_k$ and computes $s_E = \hat{Z}\hat{s}_E$. Known computational challenges with reduced Hessian methods are that a desirable basis $\hat{Z}$ may be expensive to compute, the condition number of the reduced linear system may be larger than the original one, and the product $\hat{Z}^\top B_k \hat{Z}$ is not necessarily sparse even if the matrices themselves are. For large-scale problems, these challenges can result in significant computational bottlenecks. In what follows we refer to $P^\top V_k P$ as an RCR because it has a reduced memory footprint compared to V_k in (3.2) (although the matrices have the same dimensions). We also note that V_k and $P^\top V_k P$ have the same condition, and $P^\top V_k P$ has structure that enables efficient implementations.

4.2. RCR. To simplify (3.2), we note that $V_k = P^\top V_k P$, that $P^\top A^\top = 0$, and that $P^\top J_k = \begin{bmatrix} S_k & Z_k \end{bmatrix}$ (where $P^\top Y_k \equiv Z_k$ by definition), so that

$$P^\top V_k P = \delta_k P + \begin{bmatrix} S_k & Z_k \end{bmatrix} (W_k - C_k^\top G_k C_k) \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top.$$

In Appendix A we show that $C_k^\top G_k C_k$ simplifies to $C_k^\top G_k C_k = \left[\begin{smallmatrix} (C_k^\top G_k C_k)_{11} & 0 \\ 0 & 0 \end{smallmatrix}\right]$ with $(C_k^\top G_k C_k)_{11} = \delta_k T_k^{-\top} Y_k^\top A^\top (AA^\top)^{-1} A Y_k T_k^{-1}$. Based on this, we derive an RCR of V_k.

Lemma 1. *The RCR of* V_k *in* (3.2) *for the L-BFGS matrix is given by*

$$V_k = \delta_k I + \begin{bmatrix} A^\top & S_k & Z_k \end{bmatrix} \begin{bmatrix} -\delta_k (AA^\top)^{-1} & \\ & N_k \end{bmatrix} \begin{bmatrix} A \\ S_k^\top \\ Z_k^\top \end{bmatrix}, \tag{4.1}$$

where

$$N_k = \begin{bmatrix} T_k^{-\top}(D_k + \delta_k Z_k^\top Z_k) T_k^{-1} & -\delta_k T_k^{-\top} \\ -\delta_k T_k^{-1} & 0_{k\times k} \end{bmatrix}.$$

Proof. Multiplying V_k in (3.2) from the left and right by $P^\top$ and P yields $V_k = \delta_k P + \begin{bmatrix} S_k & Z_k \end{bmatrix} (W_k - C_k^\top G_k C_k) \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top$. Since only the (1,1) block in $C_k^\top G_k C_k$ is nonzero, we consider only the (1,1) blocks, namely,

$$(W_k)_{11} - (C_k^\top G_k C_k)_{11} = T_k^{-\top}(D_k + \delta_k(Y_k^\top Y_k - Y_k^\top A^\top (AA^\top)^{-1} A Y_k)) T_k^{-1}.$$

Since $Y_k^\top P^\top Y_k = Y_k^\top P^\top P Y_k = Z_k^\top Z_k$, we obtain the (1,1) block in N_k. Subsequently, by factoring P as

$$P = I - \begin{bmatrix} A^\top & S_k & Z_k \end{bmatrix} \begin{bmatrix} -\delta_k (AA^\top)^{-1} & \\ & 0_{2k\times 2k} \end{bmatrix} \begin{bmatrix} A \\ S_k^\top \\ Z_k^\top \end{bmatrix},$$

we see that

$$P^\top V_k P = \delta_k I + \begin{bmatrix} A^\top & S_k & Z_k \end{bmatrix} \begin{bmatrix} -\delta_k (AA^\top)^{-1} & \\ & W_k - C_k^\top G_k C_k \end{bmatrix} \begin{bmatrix} A \\ S_k^\top \\ Z_k^\top \end{bmatrix}.$$

Because all blocks of $W_k - C_k^\top G_k C_k$ except for the (1,1) block are equal to those in W_k, all blocks in N_k are fully specified and representation (4.1) is complete. ∎

Note that $S_k^\top Y_k = D_k + L_k + \bar{T}_k = S_k^\top Z_k$, which means that D_k and $T_k = D_k + \bar{T}_k$ can be computed from S_k and Z_k alone and that G_k and C_k need not be explicitly computed. Therefore, for the RCR, only S_k, Z_k, T_k, and D_k are stored. An addition is the scalar δ_k, which is typically set to be $\delta_k = s_k^\top y_k / y_k^\top y_k = s_k^\top z_k / y_k^\top y_k$ and may depend on the most recent y_k. As $PJ_k = \begin{bmatrix} S_k & Z_k \end{bmatrix}$, we note a key advantage of the RCR: that (4.1) can be written as

$$V_k = \delta_k P + PJ_k N_k J_k^\top P^\top = \delta_k P + \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k \begin{bmatrix} S_k^\top \\ Z_k^\top \end{bmatrix}. \tag{4.2}$$

By storing a few columns of $\begin{bmatrix} S_k & Z_k \end{bmatrix} \in \mathbb{R}^{n \times 2l}$ (as described in section 4.4), which in turn define a small matrix $N_k \in \mathbb{R}^{2l \times 2l}$ (cf. Lemma 1), we can separate the solves with $AA^\top$ from other calculations. Concretely, note that solves with $AA^\top$ only occur as part of the orthogonal projection P, which can be represented as a linear operator and does not need to be explicitly formed. Also note that (1.7) and (4.2) are related, with the difference being that Y_k and $\delta_k I$ in (1.7) are replaced by Z_k and $\delta_k P$ in (4.2). Hence for large n and m, computation with (4.2) is efficient and requires little memory, provided orthogonal projections with P are handled effectively (as described in section 5). On the other hand, the compact representation in (3.2) does not neatly decouple solves with $AA^\top$ and results in perhaps prohibitively expensive computations for large m. In particular, G_k in the middle matrix of (3.2) is defined by $G_k \equiv (AB_k^{-1}A^\top)^{-1} \in \mathbb{R}^{m \times m}$, which interleaves solves with $AA^\top$ and other terms. Therefore, the RCR in (4.1)–(4.2) is recognizably more practical for large n and m than (3.2). We apply V_k from (4.2) to a vector g as

$$h = \begin{bmatrix} S_k^\top \\ Z_k^\top \end{bmatrix} g, \qquad V_k g = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k h + \delta_k P g. \tag{4.3}$$

4.3. Computational complexity. With adequate precomputation and storage, the cost of the matrix-vector product (4.3) is often inexpensive. If the columns of Z_k are stored, updating the small $2l \times 2l$ matrix N_k does not depend on solves with $AA^\top$. Moreover, factors of P can be precomputed once at $k = 0$ and reused. In particular, suppose that a (sparse) QR factorization $A^\top = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}$ is obtained once, with $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ being sparse, such that the product $Q^\top g$ takes $\mathcal{O}(rn)$ multiplications, where r is constant. Subsequently, the projection $Pg = g - Q_1 Q_1^\top g$ can be computed in $\mathcal{O}(n + 2rn)$ multiplications (or $Pg = Q_2 Q_2^\top g$ in $\mathcal{O}(2rn)$ multiplications). Thus, we summarize the multiplications in (4.3) as follows: h with $2nl$, $N_k h$ with negligible $(2l)^2$, $\begin{bmatrix} S_k & Z_k \end{bmatrix} N_k h$ with $2nl$, and Pg with, say, $2nr$. The total, without negligible terms, is $\mathcal{O}(2n(2l + r))$. The multiplications scale linearly with n, are related to the sparsity in A, and are thus suited for large problems.

4.4. Updating. We store and update the columns of $Z_k = \begin{bmatrix} z_{k-l} & \cdots & z_{k-1} \end{bmatrix}$ one at a time and recall that $z_k = Pg_{k+1} - Pg_k$. Based on this, no additional solves with $AA^\top$ are required to represent the matrix V_{k+1}. Specifically, suppose that we computed and stored Pg_k at the end of the previous iteration and that we compute Pg_{k+1} at the end of the current iteration. We can use this vector in two places: first to represent Z_{k+1} with $z_k = Pg_{k+1} - Pg_k$ and hence V_{k+1}, and secondly in the computation of $V_{k+1} g_{k+1}$. Thus only one solve with $AA^\top$ per iteration is necessary to update V_{k+1} and to compute a step of the form $s = -V_{k+1} g_{k+1}$.

For large problems, the limited-memory representation in (4.1) is obtained by storing only the last l columns of S_k and Z_k. With $1 \le l \ll n$, limited-memory strategies enable computational efficiencies and lower storage requirements [26]. Updating S_k and Z_k requires replacing or inserting one column at each iteration. Let an underline below a matrix represent the matrix with its first column removed. That is, $\underline{Z}_k$ represents Z_k without its first column. With this notation, a column update of a matrix Z_k by a vector z_k is defined as

$$\text{colUpdate}\left(Z_k, z_k\right) \equiv \begin{cases} [\, Z_k \; z_k \,] & \text{if } k < l, \\ [\, \underline{Z}_k \; z_k \,] & \text{if } k \geq l. \end{cases}$$

Such a column update either directly appends a column to a matrix or first removes a column and then appends one. This column update will be used, for instance, to obtain Z_{k+1} from Z_k and z_k, i.e., $Z_{k+1} = \text{colUpdate}(Z_k, z_k)$. Next, let an overline above a matrix represent the matrix with its first row removed. That is, $\overline{S_k^\top Z_k}$ represents $S_k^\top Z_k$ without its first row. With this notation, a product update of $S_k^\top Z_k$ by matrices S_k and Z_k and vectors s_k and z_k is defined as

$$\text{prodUpdate}\left(S_k^\top Z_k, S_k, Z_k, s_k, z_k\right) \equiv \begin{cases} \begin{bmatrix} S_k^\top Z_k & S_k^\top z_k \\ s_k^\top Z_k & s_k^\top z_k \end{bmatrix} & \text{if } k < l, \\ \begin{bmatrix} \underline{\left(\overline{S_k^\top Z_k}\right)} & \underline{S}_k^\top z_k \\ s_k^\top \underline{Z}_k & s_k^\top z_k \end{bmatrix} & \text{if } k \geq l. \end{cases}$$

This product update is used to compute matrix products such as $S_{k+1}^\top Z_{k+1}$ with $\mathcal{O}(2ln)$ multiplications, instead of $\mathcal{O}(l^2 n)$ when the product $S_k^\top Z_k$ is stored and the vectors s_k and z_k have been computed. Note that a diagonal matrix can be updated in this way by setting the rectangular matrices S_k and Z_k to zero and $D_{k+1} = \text{prodUpdate}(D_k, 0, 0, s_k, z_k)$. An upper triangular matrix can be updated in a similar way, e.g., $T_{k+1} = \text{prodUpdate}(T_k, S_k, 0, s_k, z_k)$. To save computation, products with zero matrices are never formed explicitly.

5. Computing projections.

With $P = I_n - A^\top (AA^\top)^{-1} A$, projections $z = Py$ can be computed by direct or iterative methods. Their efficiency depends on the sparsity of A.

5.1. QR factorization.

When A has full row rank and the QR factorization

$$A^\top = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R \tag{5.1}$$

is available, the projection operator becomes $P = I - Q_1 Q_1^\top = Q_2 Q_2^\top$. Thus, $z = Py$ can be computed stably as $z = Q_2(Q_2^\top y)$. With $m < n$, the QR factors are best obtained using a product of Householder transformations [21]:

$$Q^\top A^\top = H_m \dots H_3 H_2 H_1 A^\top = \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1^\top \\ Q_2^\top \end{bmatrix} A^\top. \tag{5.2}$$

Thus $Q = H_1 H_2 H_3 \dots H_m$, and the operators Q_1 and Q_2 are available from

$$Q_1 = Q \begin{bmatrix} I \\ 0 \end{bmatrix} \qquad \text{and} \qquad Q_2 = Q \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{5.3}$$

When A is sparse, the SuiteSparseQR software [11] permutes the columns of $A^\top$ in (5.2) to retain sparsity in H_k and R. The projection $z = Py = Q_2(Q_2^\top y)$ can then be computed efficiently.

One can avoid storage of Q_1 by noting that $Q_1 = A^\top R^{-1}$. The projection can be computed as $z = (I - Q_1 Q_1^\top)y = y - A^\top R^{-1} R^{-\top} Ay$, though with lower precision than $z = Q_2(Q_2^\top y)$.

5.2. Iterative computation of *z*. Computing QR factors is sometimes not practical because A contains one or more relatively dense columns. (In the numerical experiments of section 9, this occurred with only 2 out of 142 sparse constrained problems.) The multifrontal QR solver SuiteSparseQR [11] then has to handle dense factors, slowing computing times. For problems with thousands of constraints we regard column j as relatively dense if $\text{nnz}(A_{:j})/m > 0.1$. When one expects the QR factorization to be slow because of dense columns, an alternative is to solve the least-squares problem

$$\min_w \|A^\top w - y\| \tag{5.4}$$

and compute the residual $z = Py = y - A^\top w$. Suitable iterative solvers for (5.4) are CGLS [23], LSQR [28], and LSMR [17]. If $\tilde{A}$ is the same as A with any relatively dense columns deleted, the factor $\tilde{R}$ from a sparse QR factorization of $\tilde{A}^\top$ (again with suitable column permutation) should be a good right-preconditioner to accelerate the iterative solvers. If $\tilde{A}$ does not have full row rank, the zero or small diagonals of $\tilde{R}$ can be changed to 1 before $\tilde{R}$ is used as a preconditioner.

5.3. Implementation. Appendix B describes the implementation of the two preceding projections (Tables 1 and 2). We refer to these operations through the definition

$$z \equiv \text{compProj}(A, y, \texttt{P}) \equiv \begin{cases} \text{Householder QR} & \text{if } \texttt{P} = 1, \\ \text{Preconditioned LSQR} & \text{if } \texttt{P} = 2. \end{cases}$$

Note that the implementations do not require A to have full row rank.

5.4. Trust-region algorithm. To solve (1.1) we use the trust-region strategy, which is regarded as a robust minimization method [10]. At each iteration, the method measures progress using the ratio of actual over predicted reductions:

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{q(0) - q(s_k)},$$

Table 1

MATLAB commands to use SparseSuite functions for computing projections $z = Py$ using a Householder QR factorization.

```
% Options
opts.Q = 'Householder';
opts.permutation = 'vector';

% QR factorization using SPQR
[Q,~,~,info] = spqr(A',opts);
rankA = info.rank_A_estimate;

% Projection
ztmp = spqr_qmult(Q,y,0);
zrkA = zeros(rankA,1);
z = [zrkA;ztmp(rankA+1:end)];
z = spqr_qmult(Q,z,1);
```

Table 2

MATLAB commands for computing projections $z = Py$ using preconditioned LSQR (where $P = I - A^\top(AA^\top)^{-1}A$). If A has full row rank (`rankA` $= m$), LSQR should need only 1 iteration. Notes: SPQR uses all of $A^\top$ in the QR factorization $A^\top P_{msk} = QR$, where P_{msk} is a column permutation of $A^\top$ and R is upper trapezoidal. We store the permutation in the vector `maskA`. If $A^\top$ does not have full row rank, we use the first `rankA` columns of $A^\top P_{msk}$ (the command `A(maskA(1:rankA),:)'`). If A contains some relatively dense columns, we should partition $AP_{prt} = [\, A_S \; A_D \,]$ into sparse and dense columns, then use A_S in place of A in the call to `spqr`.

```
% Options
opts.econ = 0;
opts.Q = 'Householder';
opts.permutation = 'vector';
tol = 1e-15;
maxit = m;

% Preconditioner using a triangular
% factor from SPQR
[~,R,maskA,info] = spqr(A',opts);
rankA = info.rank_A_estimate;

% Projection
x = lsqr(A(maskA(1:rankA),:)',y,...
        tol,maxit,R(1:rankA,1:rankA));
z = y - A(maskA(1:rankA),:)'*x(1:rankA,1);
```

where s_k is an intermediate search direction, in the sense that s_k will ultimately be used as an update only if ρ_k is greater than a threshold. By accepting steps that fulfill the so-called sufficient decrease condition $\rho > c_1$ (suppressing the subscript k on ρ_k) for a constant $c_1 > 0$, the method successively moves towards a local minimizer (though there is no guarantee that a minimizer will be reached). The trust-region radius $\Delta > 0$ controls the norm of the search direction by means of the constraint $\|s\|_2 \le \Delta$. There are two possible cases for the solution of the trust-region subproblem: either the search direction is in the interior of the constraint ($\|s\| < \Delta$), or it is on the boundary ($\|s\| = \Delta$). Since the L-BFGS matrix B_k is positive definite, the solution of (1.2) is given by the unconstrained minimizer $s = s_E$ from (2.1) if $\|s_E\| \le \Delta$. Otherwise, if $\|s_E\| > 0$, then (1.2) is solved with the active norm constraint $\|s\| = \Delta$. Note that even if $\|s_E\| \le \Delta$, the condition $\rho > c_1$ might not hold. In this situation, or in any case when $\rho \le c_1$, the radius Δ is reduced and a new problem (1.2) (with smaller Δ) and constraint $\|s\| = \Delta$ is solved. The overall trust-region strategy for one iteration is given next, with radius $\Delta > 0$ and $c_1 > 0$ and iteration counter suppressed.

Trust-Region Strategy:

```
1.  Compute the unconstrained step $s \leftarrow s_E$ from (2.1) (using (4.3))
2.  While ($\|s\|_2 > \Delta$ or $\rho \le c_1$)
        2.1. Solve (1.2) with $\|s\| = \Delta$
        2.2. Reduce $\Delta$
    end
3.  Increase (or at least do not decrease) $\Delta$
4.  Update iterate $x \leftarrow x + s$
```

Practical aspects of an implementation include the setting of constants and starting the method. Detailed procedures are described in sections 6, 7, 8, and 9.

6. ℓ_2-norm trust-region constraint.

With an ℓ_2-norm trust-region constraint in (1.2), the search direction is given by

$$s_{L2} = \underset{\|s\|_2 \le \Delta_k}{\arg\min}\ q(s) \quad \text{subject to} \quad As = 0.$$

With $\sigma \ge 0$ denoting a scalar Lagrange multiplier, the search direction is a feasible solution to a shifted KKT system including the norm constraint:

$$\begin{bmatrix} B_k + \sigma I & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} s_{L2} \\ \lambda_{L2} \end{bmatrix} = \begin{bmatrix} -g_k \\ 0 \end{bmatrix}, \qquad \|s_{L2}\|_2 \le \Delta_k. \tag{6.1}$$

By computing the (1,1) block of the shifted inverse KKT matrix, we note that a necessary condition for the solution is $s_{L2}(\sigma) = -V_k(\sigma) g_k$, where

$$V_k(\sigma) = (B_k + \sigma I)^{-1} - (B_k + \sigma I)^{-1} A^\top (A (B_k + \sigma I)^{-1} A^\top)^{-1} A (B_k + \sigma I)^{-1}.$$

For the L-BFGS matrix, with $\tau_k = \tau_k(\sigma) = (1/\delta_k + \sigma)$ we have $(B_k + \sigma I)^{-1} = \tau_k^{-1} I + J_k W_k(\sigma) J_k^\top$, where the small $2l \times 2l$ matrix is

$$W_k(\sigma) = -\begin{bmatrix} \theta_k S_k^\top S_k & \theta_k L_k + \tau_k T_k \\ \theta_k L_k^\top + \tau_k T_k^\top & \tau_k(\tau_k D_k + Y_k^\top Y_k) \end{bmatrix}^{-1}$$

with $\theta_k = \tau_k(1 - \delta_k \tau_k)$. In terms of $C_k(\sigma) \equiv A J_k W_k(\sigma)$ and $G_k(\sigma) \equiv (A(B_k + \sigma I)^{-1} A^\top)^{-1}$, the compact representation of $V_k(\sigma)$ [6, Corollary 1] is

$$V_k(\sigma) = \frac{1}{\tau_k} I + \begin{bmatrix} A^\top & J_k \end{bmatrix} \begin{bmatrix} -\frac{1}{\tau_k^2} G_k(\sigma) & -\frac{1}{\tau_k} G_k(\sigma) C_k(\sigma) \\ -\frac{1}{\tau_k} C_k(\sigma)^\top G_k(\sigma) & W_k(\sigma) - C_k(\sigma)^\top G_k(\sigma) C_k(\sigma) \end{bmatrix} \begin{bmatrix} A \\ J_k^\top \end{bmatrix}. \tag{6.2}$$

Once the middle matrix in (6.2) is formed, the compact representation can be used to compute matrix-vector products efficiently. However, when m is large (many equality constraints), computing terms such as $G_k(\sigma)$ become expensive. Therefore, we describe a reduced representation similar to (4.1), based on the property that $P^\top V_k(\sigma) P = V_k(\sigma)$ and by storing S_k and Z_k. Lemma 2 summarizes the outcome.

LEMMA 2. *The RCR of $V_k(\sigma)$ in (6.2) for the L-BFGS matrix is given by*

$$V_k(\sigma) = \frac{1}{\tau_k} I + \begin{bmatrix} A^\top & S_k & Z_k \end{bmatrix} \begin{bmatrix} -\frac{1}{\tau_k}(AA^\top)^{-1} & \\ & N_k(\sigma) \end{bmatrix} \begin{bmatrix} A \\ S_k^\top \\ Z_k^\top \end{bmatrix}, \tag{6.3}$$

where $\tau_k = \tau_k(\sigma) = (1/\delta_k + \sigma)$, $\theta_k = \theta_k(\sigma) = \tau_k(\sigma)(1 - \delta_k \tau_k(\sigma))$, and

$$N_k(\sigma) = -\begin{bmatrix} \theta_k(\sigma) S_k^\top S_k & \theta_k(\sigma) L_k + \tau_k(\sigma) T_k \\ \theta_k(\sigma) L_k^\top + \tau_k(\sigma) T_k^\top & \tau_k(\sigma)(\tau_k(\sigma) D_k + Z_k^\top Z_k) \end{bmatrix}^{-1}.$$

Proof. To simplify notation, we suppress the explicit dependence on σ in this proof, so that $V_k \equiv V_k(\sigma)$, $C_k \equiv C_k(\sigma)$, and $W_k \equiv W_k(\sigma)$. Multiplying V_k in (6.2) from the left and right by $P^\top$ and P yields

$$V_k = \frac{1}{\tau_k} P + \begin{bmatrix} S_k & Z_k \end{bmatrix} (W_k - C_k^\top G_k C_k) \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top.$$

Observe that $C_k = AJ_kW_k = \begin{bmatrix} 0 & AY_k \end{bmatrix} W_k$ is block-rectangular and that $G_k = (A(\frac{1}{\tau_k}I + J_kW_kJ_k^\top)A^\top)^{-1}$ depends on W_k. Defining $F_k \equiv \tau_k(AA^\top)^{-1}$, we show that the Sherman–Morrison–Woodbury (SMW) inverse gives the simplification

$$\begin{aligned}
&W_k - C_k^\top G_k C_k \\
&= W_k - W_k \begin{bmatrix} 0 \\ Y_k^\top A^\top \end{bmatrix} G_k \begin{bmatrix} 0 & AY_k \end{bmatrix} W_k \\
&= W_k - W_k \begin{bmatrix} 0 \\ Y_k^\top A^\top \end{bmatrix} \left(I + \begin{bmatrix} 0 & F_kAY_k \end{bmatrix} W_k \begin{bmatrix} 0 \\ Y_k^\top A^\top \end{bmatrix}\right)^{-1} \begin{bmatrix} 0 & F_kAY_k \end{bmatrix} W_k \\
&= \left(W_k^{-1} + \begin{bmatrix} 0 \\ Y_k^\top A^\top \end{bmatrix} \begin{bmatrix} 0 & F_kAY_k \end{bmatrix}\right)^{-1},
\end{aligned}$$

where the third equality is obtained by applying the SMW formula in reverse. Since only the (2,2) block in the low-rank matrix of the third equality is nonzero, and since $F_k = \tau_k(AA^\top)^{-1}$, note that

$$(W_k^{-1})_{22} + Y_k^\top A^\top F_k AY_k = -(\tau_k(\tau_k D_k + Y_k^\top Y_k - Y_k^\top A^\top (AA^\top)^{-1} AY_k)),$$

which corresponds to the $(2,2)$ block $N_k(\sigma)$ in (6.3). Because all other blocks are unaffected, it holds that $W_k - C_k^\top G_k C_k = N_k(\sigma)$. Subsequently, by factoring $P = I - A^\top(AA^\top)^{-1}A$ we deduce the compact representation (6.3). ∎

Note that $S_k^\top Z_k = S_k^\top Y_k = L_k + D_k + \bar{T}_k$, with $T_k = D_k + \bar{T}_k$, means that the RCR for $V_k(\sigma)$ is fully specified by storing S_k and Z_k. An exception is the scalar δ_k, which may depend on the most recent y_k. Also when $\sigma = 0$, the representations (4.1) and (6.3) coincide. We apply $V_k(\sigma)$ to a vector g as

$$h = \begin{bmatrix} S_k^\top \\ Z_k^\top \end{bmatrix} g, \qquad V_k(\sigma)g = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k(\sigma)h + \frac{1}{\tau_k}Pg.$$

6.1. ℓ_2-norm search direction. To compute the ℓ_2 trust-region minimizer we first set $\sigma = 0$ and $s_{L2}(0) = -V_k(0)g_k$. If $\|s_{L2}(0)\|_2 \le \Delta_k$, the minimizer with the ℓ_2-norm is given by $s_{L2}(0)$. Otherwise ($\|s_{L2}(0)\|_2 > \Delta_k$) we define the so-called secular equation [10] as

$$\phi(\sigma) \equiv \frac{1}{\|s_{L2}(\sigma)\|_2} - \frac{1}{\Delta_k}.$$

To solve the secular equation we apply the 1D Newton iteration

$$\sigma_{j+1} = \sigma_j - \frac{\phi(\sigma_j)}{\phi'(\sigma_j)},$$

where $\phi'(\sigma_j) = -(s_{L2}(\sigma_j)^\top s_{L2}(\sigma_j)')/\|s_{L2}(\sigma_j)\|_2^3$ and $s_{L2}(\sigma_j)' = -V_k(\sigma_j)s_{L2}(\sigma_j)$ (with prime " $'$ " denoting the derivative). Note that $s_{L2}(\sigma_j)'$ can be derived from the shifted system (6.1) by differentiation with respect to σ. Applying the product rule in (6.1) and regarding the solutions as functions of σ, i.e., $s_{L2}' \equiv s_{L2}(\sigma)'$ and $\lambda_{L2}' \equiv \lambda_{L2}(\sigma)'$, one obtains the differentiated system

$$\begin{bmatrix} B_k + \sigma I & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} s_{L2}' \\ \lambda_{L2}' \end{bmatrix} = \begin{bmatrix} -s_{L2} \\ 0 \end{bmatrix}.$$

Since the system matrix is the same as in (6.1) (only the right-hand side differs), $s_{L2}(\sigma_j)'$ is fully determined by $V_k(\sigma_j)$ and $s_{L2}(\sigma_j)$. Starting from $\sigma_0 = 0$, we terminate the Newton iteration if $|\phi(\sigma_{j+1})| \le \varepsilon$ or an iteration limit is reached. The search direction is then computed as $s_{L2}(\sigma_{j+1}) = -V_k(\sigma_{j+1})g_k$.

Our approach with the ℓ_2-norm is summarized in Algorithm 6.1. This algorithm is based on storing and updating S_k, Z_k, and the small blocks of $N_k(\sigma)$ in (6.3). Suppose that s_0 and z_0 are obtained by an initialization procedure (for instance, Init. 1 from section 9). With $k = 0$, the initial matrices that define $V_k(\sigma)$ are given as

$$S_k = \begin{bmatrix} s_k \end{bmatrix}, \quad Z_k = \begin{bmatrix} z_k \end{bmatrix}, \tag{6.4}$$

$$D_k = \begin{bmatrix} s_k^\top z_k \end{bmatrix}, \quad T_k = \begin{bmatrix} s_k^\top z_k \end{bmatrix}, \quad Z_k^\top Z_k = \begin{bmatrix} z_k^\top z_k \end{bmatrix}, \quad L_k = \begin{bmatrix} 0 \end{bmatrix}. \tag{6.5}$$

Once the iteration starts, we update

$$S_{k+1} = \mathrm{colUpdate}(S_k, s_k), \quad Z_{k+1} = \mathrm{colUpdate}(Z_k, z_k), \tag{6.6}$$

$$\begin{aligned} D_{k+1} &= \mathrm{prodUpdate}(D_k, 0, 0, s_k, z_k), \\ T_{k+1} &= \mathrm{prodUpdate}(T_k, S_k, 0, s_k, z_k), \\ Z_{k+1}^\top Z_{k+1} &= \mathrm{prodUpdate}(Z_k^\top Z_k, Z_k, Z_k, z_k, z_k), \text{ and} \\ L_{k+1} &= \mathrm{prodUpdate}(L_k, 0, Z_k, s_k, 0). \end{aligned} \tag{6.7}$$

Note that we store and update matrices like $Z_k^\top Z_k \in \mathbb{R}^{l \times l}$ instead of recomputing them. Because of the limited-memory technique (typically $3 \le l \le 7$ [8]), such matrices are very small relative to large n. Subsequently, $N_k(\sigma) \in \mathbb{R}^{2l \times 2l}$, defined by the blocks in (6.7), remains very small compared to n.

7. Eigendecomposition of $\boldsymbol{V_k}$.

We describe how to exploit the structure of the RCR (4.1) to compute an implicit eigendecomposition of V_k and how to combine this with a shape-changing norm. The effect is that the trust-region subproblem solution is given by an analytic formula. Since the RCR is equivalent to representation (3.2), we can apply previous results. However, using representation (4.1) is computationally more efficient. First, note that $N_k \in \mathbb{R}^{2l \times 2l}$ is a small symmetric square matrix. Therefore, computing the nonzero eigenvalues and corresponding eigenvectors of the matrix $\begin{bmatrix} S_k & Z_k \end{bmatrix} N_k \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top = U_2 \Lambda_2 U_2^\top$ is inexpensive. In particular, we compute the thin QR factorization $\begin{bmatrix} S_k & Z_k \end{bmatrix} = \widehat{Q}_2 \widehat{R}_2$ and the small eigendecomposition $\widehat{R}_2 N_k \widehat{R}_2^\top = \widehat{P}_2 \Lambda_2 \widehat{P}_2^\top$. The small factorization is then

$$\begin{bmatrix} S_k & Z_k \end{bmatrix} N_k \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top = \widehat{Q}_2 (\widehat{R}_2 N_k \widehat{R}_2^\top) \widehat{Q}_2^\top = \widehat{Q}_2 (\widehat{P}_2 \Lambda_2 \widehat{P}_2^\top) \widehat{Q}_2^\top \equiv U_2 \Lambda_2 U_2^\top,$$

where the orthonormal matrix on the right-hand side is defined as $U_2 \equiv \widehat{Q}_2 \widehat{P}_2$. Since $A^\top (AA^\top)^{-1} A = Q_1 Q_1^\top$ from (5.1), we express V_k as

$$V_k = \delta_k I + \begin{bmatrix} Q_1 & U_2 \end{bmatrix} \begin{bmatrix} -\delta_k I_m & \\ & \Lambda_2 \end{bmatrix} \begin{bmatrix} Q_1^\top \\ U_2^\top \end{bmatrix},$$

where $Q_1 \in \mathbb{R}^{n \times m}$ and $U_2 \in \mathbb{R}^{n \times 2l}$ are orthonormal, while $\Lambda_2 \in \mathbb{R}^{2l \times 2l}$ is diagonal. Defining the orthogonal matrix $U \equiv \begin{bmatrix} Q_1 & U_2 & U_3 \end{bmatrix}$, where $U_3 \in \mathbb{R}^{n \times n-(m+2l)}$ represents the orthogonal complement of $\begin{bmatrix} Q_1 & U_2 \end{bmatrix}$, we obtain the implicit eigendecomposition of V_k as

$$V_k = \begin{bmatrix} Q_1 & U_2 & U_3 \end{bmatrix} \begin{bmatrix} 0_m & & \\ & \delta_k I_{2l} + \Lambda_2 & \\ & & \delta_k I_{n-(m+2l)} \end{bmatrix} \begin{bmatrix} Q_1^\top \\ U_2^\top \\ U_3^\top \end{bmatrix} \equiv U \Lambda U^\top. \tag{7.1}$$

Algorithm 6.1. LTRL2-SLEC (limited-memory trust-region 2-norm for sparse linear equality constraints).

```
Ensure: $0 < c_1, 0 < c_2, c_3, c_4, c_5, c_6 < 1 < c_7, 0 < \varepsilon_1, \varepsilon_2, 0 < i_{\max}, k = 0, 0 < l, \Delta_k = \|x_k\|_2$, $g_k = \nabla f(x_k)$, $\mathtt{P} \in [0,1]$, $g_k^P = \text{compProj}(A, g_k, \mathtt{P})$, $g_{k+1}^P, s_k, z_k, y_k$ (from initialization), $S_k, Z_k, D_k, T_k, L_k, Z_k^\top Z_k$ from (6.4) and (6.5), $\delta_k = s_k^\top z_k / y_k^\top y_k$, $\sigma = 0$, $\tau_k = (1/\delta_k + \sigma)$, $\theta_k = \tau_k(1 - \delta_k \tau_k)$, $N_k(\sigma)$ from (6.3), $k = k + 1$
1: while ($\varepsilon_1 \leq \|g_k^P\|_\infty$) do
2:   $h = -\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top g_k$
3:   $s_k = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k(0) h - \delta_k g_k^P$; $\rho_k = 0$ {Equality constrained step}
4:   if $\|s_k\|_2 \leq \Delta_k$ then
5:     $\rho_k = (f(x_k) - f(x_k + s_k))/(q(0) - q(s_k))$
6:   end if
7:   while $\rho_k \leq c_1$ do
8:     $\sigma = 0$, $i = 0$; $\tau_k = (1/\delta_k + \sigma)$, $\theta_k = \tau_k(1 - \delta_k \tau_k)$
9:     $h' = -\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top s_k$
10:    $s_k' = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k(\sigma) h' - \delta_k s_k$;
11:    while $\varepsilon_2 < |\phi(\sigma)|$ and $i < i_{\max}$ do
12:      $\sigma = \sigma - \phi(\sigma)/\phi'(\sigma)$
13:      $\tau_k = (1/\delta_k + \sigma)$, $\theta_k = \tau_k(1 - \delta_k \tau_k)$
14:      $h = -\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top g_k$; $s_k = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k(\sigma) h - \frac{1}{\tau_k} g_k^P$
15:      $h' = -\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top s_k$; $s_k' = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k(\sigma) h' - \frac{1}{\tau_k} s_k$;
16:      $i = i + 1$
17:    end while{Newton's method}
18:    $\rho_k = 0$
19:    if $0 < (f(x_k) - f(x_k + s_k))$ then
20:      $\rho_k = (f(x_k) - f(x_k + s_k))/(q(0) - q(s_k))$
21:    end if
22:    if $\rho_k \leq c_2$ then
23:      $\Delta_k = \min(c_3 \|s_k\|_2, c_4 \Delta_k)$
24:    end if
25:  end while
26:  $x_{k+1} = x_k + s_k$ {Accept step}
27:  if $c_5 \Delta_k \leq \|s_k\|_2$ and $c_6 \leq \rho_k$ then
28:    $\Delta_k = c_7 \Delta_k$
29:  end if
30:  $g_{k+1} = \nabla f(x_{k+1})$, $g_{k+1}^P = \text{compProj}(A, g_{k+1}, \mathtt{P})$, $z_k = g_{k+1}^P - g_k^P$, $y_k = g_{k+1} - g_k$, $S_{k+1}, Z_{k+1}, D_{k+1}, T_{k+1}, L_{k+1}, Z_{k+1}^\top Z_{k+1}$ from (6.6) and (6.7) $\delta_{k+1} = z_k^\top s_k / y_k^\top y_k$, $\sigma = 0$, $\tau_k = (1/\delta_k + \sigma)$, $\theta_k = \tau_k(1 - \delta_k \tau_k)$
31:  Update $N_k(\sigma)$ from (6.3), $k = k + 1$
32: end while
```

Note that we do not explicitly form the potentially expensive to compute orthonormal matrix U_3, as only scaled projections $\delta_k U_3 U_3^\top$ are needed. We therefore refer to factorization (7.1) as being implicit. In particular, from the identity $UU^\top = I$, we obtain that $U_3 U_3^\top = I - Q_1 Q_1^\top - U_2 U_2^\top = P - U_2 U_2^\top$. Note here and above that U_2 is a thin rectangular matrix with only $2l$ columns.

8. Shape-changing-norm trust-region constraint. To make use of the implicit eigensystem (7.1), we apply the so-called shape-changing infinity norm introduced in [7]:

$$\|s\|_U \equiv \max\left\{ \left\| \begin{bmatrix} Q_1 & U_2 \end{bmatrix}^\top s \right\|_\infty, \left\| U_3^\top s \right\|_2 \right\}.$$

With this norm, the trust-region subproblem has a computationally efficient solution that can be obtained from

$$s_{SC} = \underset{\|s\|_U \le \Delta_k}{\arg\min}\; q(s) \quad \text{subject to} \quad As = 0.$$

Since the RCR is equivalent to (3.2), we invoke [6, section 5.5] to obtain an direct formula for the search direction:

$$s_{SC} = U_2(v_2 - \beta U_2^\top g_k) + \beta P g_k,$$

where with $U_2^\top g_k = \widehat{P}_2^\top \widehat{R}_2^{-\top} \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top g_k \equiv u_k$, and $\mu_i = (\delta_k + (\Lambda_2)_{ii})^{-1}$,

$$(8.1) \qquad (v_2)_i = \begin{cases} \frac{-(u_k)_i}{\mu_i} & \text{if } \left| \frac{(u_k)_i}{\mu_i} \right| \le \Delta_k, \\ \frac{-\Delta_k (u_k)_i}{|(u_k)_i|} & \text{otherwise,} \end{cases}$$

$$(8.2) \qquad \beta = \begin{cases} -\delta_k & \text{if } \|\delta_k U_3^\top g_k\|_2 \le \Delta_k, \\ \frac{-\Delta_k}{\|U_3^\top g_k\|_2} & \text{otherwise,} \end{cases}$$

for $1 \le i \le 2l$. More details for the computation of s_{SC} are in Appendix C. Note that the norm $\|U_3^\top g_k\|_2$ can be computed without explicitly forming U_3, since $\|U_3^\top g_k\|_2^2 = g_k^\top (P - U_2 U_2^\top) g_k = \|P g_k\|_2^2 - \|U_2^\top g_k\|_2^2$. The trust-region algorithm using the RCR and the shape-changing norm is summarized in Algorithm 8.1 below. Like Algorithm 6.1, this algorithm is based on storing and updating S_k, Z_k, and the small blocks of N_k in (4.1). Therefore, the initializations (6.4)–(6.5) and updates (6.6)–(6.7) can be used. In addition, since in the thin QR factorization $\begin{bmatrix} S_k & Z_k \end{bmatrix} = \hat{Q}_2 \hat{R}_2$ the triangular $\hat{R}_2$ is computed from a Cholesky factorization of $\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top \begin{bmatrix} S_k & Z_k \end{bmatrix}$, we initialize the matrices

$$(8.3) \qquad S_k^\top S_k = \begin{bmatrix} s_k^\top s_k \end{bmatrix}, \quad S_k^\top Z_k = \begin{bmatrix} s_k^\top z_k \end{bmatrix}$$

with corresponding updates

$$(8.4) \qquad \begin{aligned} S_{k+1}^\top S_{k+1} &= \text{prodUpdate}(S_k^\top S_k, S_k, S_k, s_k, s_k) \text{ and} \\ S_{k+1}^\top Z_{k+1} &= \text{prodUpdate}(S_k^\top Z_k, S_k, Z_k, s_k, z_k). \end{aligned}$$

As before, with a small memory parameter l, these matrices are very small compared to large n, and computations with them are inexpensive.

9. Numerical experiments. The numerical experiments are carried out in MATLAB 2016a on a MacBook Pro @2.6 GHz Intel Core i7 with 32 GB of memory. For comparisons, we use the implementations of Algorithms 1 and 2 from [6], which we label TR1 and TR2. All codes are available in the public domain:

https://github.com/johannesbrust/LTR_LECx

For TR1, TR2 we use the modified stopping criterion $\|P g_k\|_\infty \le \epsilon$ in place of $\|P g_k\|_2 / \max(1, x_k) \le \epsilon$ in order to compare consistently across solvers. Unless otherwise specified, the default parameters of these two algorithms are used. We use the following names for our proposed algorithms:

Algorithm 8.1. LTRSC-SLEC (limited-memory trust-region shape-changing norm for sparse linear equality constraints).

```
Ensure: $0 < c_1$, $0 < c_2, c_3, c_4, c_5, c_6 < 1 < c_7$, $0 < \varepsilon_1$, $0 < l$, $k = 0$, $\Delta_k = \|x_k\|_2$, $g_k = \nabla f(x_k)$, $\mathtt{P} \in [0,1]$, $g^P_k = \text{compProj}(A, g_k, \mathtt{P})$, $g^P_{k+1}, s_k, z_k, y_k$ (from initialization),
    $S_k, Z_k, D_k, T_k, Z_k^\top Z_k, S_k^\top S_k, S_k^\top Z_k$ from (6.4), (6.5), and (8.3), $\delta_k = s_k^\top z_k / y_k^\top y_k$,
    $N_k$ from (4.1), $k = k+1$
1:  while $(\varepsilon_1 \le \|g^P_k\|_\infty)$ do
2:    $h = -\begin{bmatrix} S_k & Z_k \end{bmatrix}^\top g_k$
3:    $s_k = \begin{bmatrix} S_k & Z_k \end{bmatrix} N_k h - \delta_k g^P_k$; $\rho_k = 0$ ; {Equality constrained step}
4:    if $\|s_k\|_2 \le \Delta_k$ then
5:      $\rho_k = (f(x_k) - f(x_k + s_k))/(q(0) - q(s_k))$; $\|s_k\| = \|s_k\|_2$
6:    end if
7:    if $\rho_k \le c_1$ then
8:      $\hat{R}_2^\top \hat{R}_2 = \begin{bmatrix} S_k^\top S_k & S_k^\top Z_k \\ Z_k^\top S_k & Z_k^\top Z_k \end{bmatrix}$ {Cholesky factorization}
9:      $\hat{P}_2 \Lambda_2 \hat{P}_2^\top = \hat{R}_2 N_k \hat{R}_2^\top$ {Eigendecomposition}
10:     $u_k = \hat{P}_2^\top \hat{R}_2^{-\top} \begin{bmatrix} S_k & Z_k \end{bmatrix}^\top g_k$
11:     $\xi_k = (\|g^P_k\|_2^2 - \|u_k\|_2^2)^{\frac{1}{2}}$
12:     while $\rho_k \le c_1$ do
13:       Set $v_2$ from (8.1) using $u_k$, $\Lambda_2$
14:       Set $\beta$ from (8.2) using $\xi_k = \|U_3^\top g_k\|_2$
15:       $s_k = \begin{bmatrix} S_k & Z_k \end{bmatrix} \hat{R}_2^{-1} \hat{P}_2 (v_2 - \beta u_k) + \beta g^P_k$;     $\rho_k = 0$
16:       if $0 < (f(x_k) - f(x_k + s_k))$ then
17:         $\rho_k = (f(x_k) - f(x_k + s_k))/(q(0) - q(s_k))$
18:       end if
19:       if $\rho_k \le c_2$ then
20:         $\Delta_k = \min(c_3 \|s_k\|_U, c_4 \Delta_k)$
21:       end if
22:     end while
23:     $\|s_k\| = \|s_k\|_U$
24:   end if
25:   $x_{k+1} = x_k + s_k$ {Accept step}
26:   if $c_5 \Delta_k \le \|s_k\|$ and $c_6 \le \rho_k$ then
27:     $\Delta_k = c_7 \Delta_k$
28:   end if
29:   $g_{k+1} = \nabla f(x_{k+1})$, $g^P_{k+1} = \text{compProj}(A, g_{k+1}, \mathtt{P})$, $z_k = g^P_{k+1} - g^P_k$, $y_k = g_{k+1} - g_k$,
      $S_{k+1}, Z_{k+1}, D_{k+1}, T_{k+1}, Z_{k+1}^\top Z_{k+1}, S_{k+1}^\top S_{k+1}, S_{k+1}^\top Z_{k+1}$ from (6.6), (6.7) and
      (8.4); $\delta_{k+1} = z_k^\top s_k / y_k^\top y_k$
30:   Update $N_k$ from (4.1); $k = k+1$
31: end while
```

TR1H:	Algorithm 6.1 with representation (6.3) and Householder QR
TR1L:	Algorithm 6.1 with representation (6.3) and preconditioned LSQR
TR2H:	Algorithm 8.1 with representation (4.1) and Householder QR
TR2L:	Algorithm 8.1 with representation (4.1) and preconditioned LSQR

Note that TR1 and TR2 were developed for low-dimensional linear equality constraints. In addition, we include IPOPT [30] with an L-BFGS quasi-Newton matrix (we use a precompiled Mex file with IPOPT 3.12.12 that includes MUMPS and MA57 libraries). We note that a commercial state-of-the-art quasi-Newton trust-region solver that uses a projected conjugate gradient solver is implemented in the KNITRO-INTERIOR/CG [9, Algorithm 3.2]. For the freely available IPOPT we specify the L-BFGS option using the option `hessian_approximation='limited memory'` with `tol=1e-5`. (The parameter `tol` is used by IPOPT to ensure that the (scaled) projected gradient in the infinity norm and the constraint violation are below the specified threshold. The default value is `tol=1e-8`.) All other parameters in IPOPT are at their default values unless otherwise specified. The parameters in TR1{H,L} and TR2{H,L} are set to c_1 (as machine epsilon), $c_2 = 0.75$, $c_3 = 0.5$, $c_4 = 0.25$, $c_5 = 0.8$, $c_6 = 0.25$, $c_7 = 2$, and $i_{\max} = 10$. The limited-memory parameter of all compared TR solvers is set to $l = 5$ (IPOPT's default is $l = 6$). Because the proposed methods are applicable to problems with a large number of constraints, problems with large dimensions such as $m \geq 10^4$, $n \geq 10^5$ are included. Throughout the experiments, $A \in \mathbb{R}^{m\times n}$ with $m < n$.

To initialize the algorithm, we distinguish two main cases. If x_0 is not available, it is computed as the minimum-norm solution $x_0 = \text{argmin}_x \|x\|_2$ subject to $Ax = b$ (e.g., $x_0 = A^\top(AA^\top)^{-1}b$ when A is full rank.) If $\hat{x}_0$ is provided but is infeasible, the initial vector can be computed from $p_0 = \text{argmin}_p \|p\|_2$ subject to $Ap = b - A\hat{x}_0$ and $x_0 = \hat{x}_0 + p_0$. To compute the initial vectors $s_0 = x_1 - x_0$, $z_0 = Pg_1 - Pg_0$, and $y_0 = g_1 - g_0$ we determine an initial x_1 value also. Suppose that at $k = 0$, all of x_k, $g_k = \nabla f(x_k)$, and $g_k^P = Pg_k$ are known. An initialization for s_k, z_k, and y_k at $k = 0$ is the following:

```
Init. 1:
1.  Backtracking line-search: $x_{k+1} = x_k - \alpha g_k^P/\|g_k^P\|_2$ (cf. [27, Algorithm 3.1])
2.  $g_{k+1} = \nabla f(x_{k+1})$, $g_{k+1}^P = \text{compProj}(A, g_{k+1}, \texttt{P})$
3.  $s_k = x_{k+1} - x_k$,
    $z_k = g_{k+1}^P - g_k^P$,
    $y_k = g_{k+1} - g_k$
```

Once s_0, z_0, and y_0 have been initialized (with initial radius $\Delta_0 = \|s_0\|_2$), all other updates are done automatically within the trust-region strategy.

The outcomes from the subsequent Experiments I–III are summarized in Figures 1–3 as performance profiles (Dolan and Moré [15], extended in [25], and often used to compare the effectiveness of various solvers). Detailed information for each problem instance is in Tables 3–5. Relative performances are displayed in terms of iterations and computation times. The performance metric $\rho_s(\tau)$ on n_p test problems is given by

$$\rho_s(\tau) = \frac{\text{card}\,\{p : \pi_{p,s} \leq \tau\}}{n_p} \quad \text{and} \quad \pi_{p,s} = \frac{t_{p,s}}{\underset{1\leq i\leq S,\ i\neq s}{\min} t_{p,i}},$$

where $t_{p,s}$ is the "output" (i.e., iterations or time) of "solver" s on problem p, and S denotes the total number of solvers for a given comparison. This metric measures the proportion of how close a given solver is to the best result. Extended performance profiles are the same as the classical ones but include the part of the domain where $\tau \leq 1$. In the profiles we include a dashed vertical grey line to indicate $\tau = 1$. We note that although the iteration numbers are recorded differently for each solver, they correspond approximately to the number of KKT systems solved.

Overall, we observe that the number of iterations used by the respective solvers is relatively similar across different problems. However, the differences in computation times are large. In particular, the RCR implementations use the least time

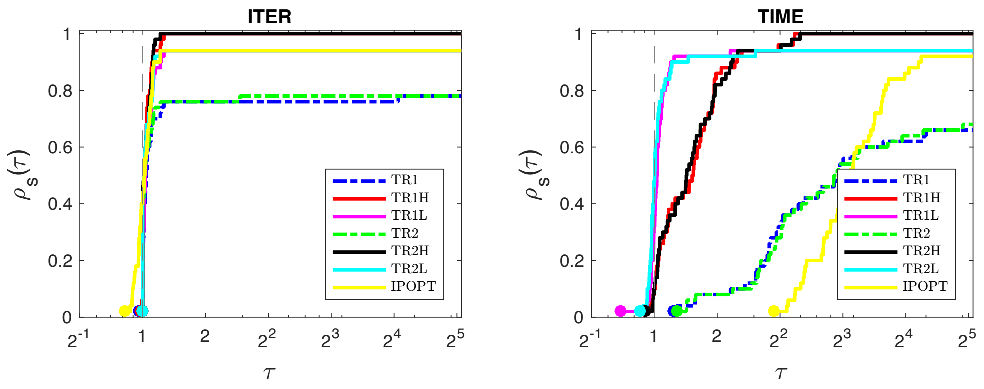

FIG. 1. *Comparison of the* 7 *solvers from Experiment* I *using performance profiles* [15] *on* 50 *test problems from* [12]. *TR2H and TR1H converge on all problem instances (100%). TR2L, TR1L, and IPOPT converge on* 47 *problems (94%). TR2 and TR1 are not applied to* 9 *large problems. In the right plot, TR2L and TR1L are the fastest (as seen from their curves being above others), while TR2H and TR1H are the most robust (as seen from their curves ultimately reaching the top of the plot). Overall, TR2{H,L} and TR1{H,L} are faster than the other solvers.*

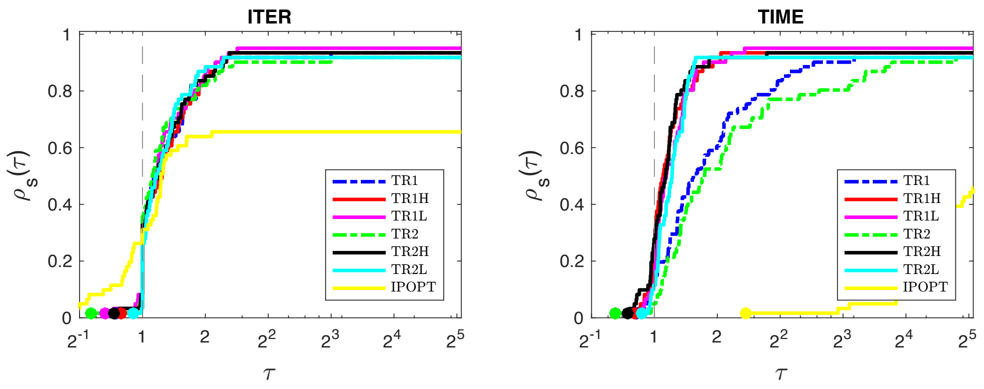

FIG. 2. *Comparison of the* 7 *solvers from Experiment* II *using performance profiles on* 62 *test problems from* [22]. *TR1L converged on* 58 *problems. All other solvers except IPOPT converged on* 57 *problems. In the left plot, the iteration numbers for TR1, TR1{H,L}, TR2, and TR2{H,L} are similar, as seen by the tight clustering of the lines. However, the computational times of TR1 and TR2 are markedly higher than those of TR1{H,L} and TR2{H,L}, as seen from the widening gap in the right plot.*

in almost all problem instances. This is possible because RCR enables an efficient decoupling of computations with the constraint matrix A and remaining small terms.

9.1. Experiment I.

This experiment uses problems with sparse and possibly low-rank $A \in \mathbb{R}^{m\times n}$. The objective is the Rosenbrock function

$$f(x) = \sum_{i=1}^{n/2} (x_{2i} - x_{2i-1})^2 + (1 - x_{2i-1})^2,$$

where n is an even integer. The matrices $A \in \mathbb{R}^{m\times n}$ are obtained from the SuiteSparse Matrix Collection [13]. Because TR1 and TR2 were not developed for problems with a large number of constraints, these solvers are only applied to problems for which

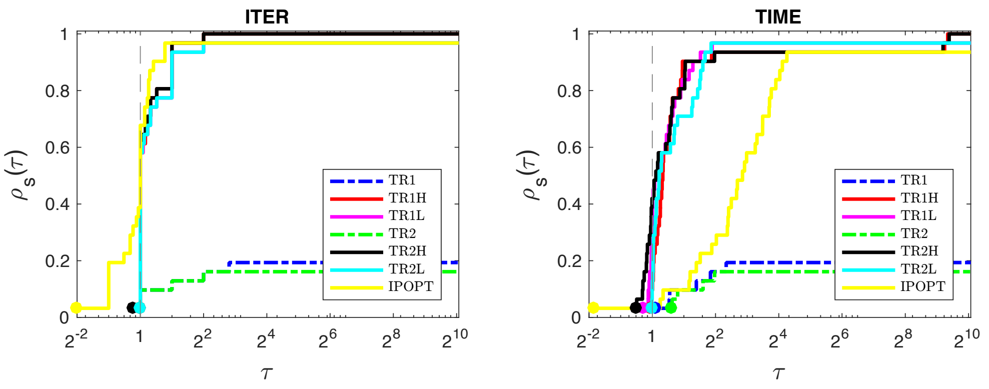

FIG. 3. *Comparison of the* 7 *solvers from Experiment* III *using performance profiles on* 31 *large linear equality constrained test problems from* [22]. *TR*1 *and TR*2 *are applied to* 6 *problems (they are not practical on the remaining problems because of their size). TR*2*H (also TR*1*H) converged on all* 31 *instances. TR*1*L, TR*2*L, and IPOPT converged on* 30 *problems. In the ITER plot the number of iterations is relatively similar across the solvers that converged. In the TIME plot there is a gap between TR*1*{H,L}, TR*2*{H,L}, and IPOPT. TR*2*L can have computational advantages but appears slightly less robust than TR*2*H, as seen from the final staircase in the TIME plot.*

$m \le 2500$. All other solvers were run on all test problems. Convergence of an algorithm is determined when two conditions are satisfied:

$$\|Pg_k\|_\infty < 10^{-5} \quad \text{and} \quad \|Ax_k - b\|_2 < 10^{-7}. \tag{9.1}$$

We summarize the outcomes in Figure 1 and Table 3.

In this experiment we observe that our proposed algorithms (any of TR1{H,L}, TR2{H,L}) perform well in terms of computing time. Both "H" versions of the proposed algorithms converged to the prescribed tolerances on all problems. On the other hand, the "L" versions are often the overall fastest, yet they did not converge on 3 problem instances (`beacxc`, `lp_cre_d`, `fit2d`).

After rerunning the 3 problems for which IPOPT did not converge, we note that IPOPT did converge to its own (scaled) tolerances on one of these problems (`beacxc`), yet the computed solution did not satisfy (9.1). On the other two problems (`lp_cre_d`, `fit2d`), IPOPT returned a message such as info.status=−2, which is caused by an abort when the "restoration phase" is called at an almost feasible point.

9.2. Experiment II. In a second experiment, we compare the 7 solvers on large problems from the CUTEst collection [22]. The dimension n is determined by the size of the corresponding CUTEst problem, while we set m to be about 25% of n, i.e., `m=ceil(0.25n)`. The matrices A are formed as `A=sprand(m,n,0.1)` with `rng(090317)`. Convergence is determined by each algorithm internally. For TR1, TR1H, TR1L, TR2, TR2H, TR2L the conditions $\|Pg_k\|_\infty < 1\times10^{-5}$ and $\|Ax_k - b\|_2 < 5\times10^{-8}$ are explicitly enforced, while for IPOPT we set `options_ipopt.ipopt.tol=1e-5`. We use the iteration limit of $100,000$ for all solvers. The limited-memory parameter is $l = 5$ for all TR solvers and $l = 6$ (default) for IPOPT. We summarize the outcomes in Figure 2 and Table 4.

9.3. Experiment III. In a third experiment we compare the 7 solvers on 31 linear equality constrained problems from CUTEst. Four of these problems (`AUG2D`, `AUG2DC`, `AUG3D`, `AUG3DC`) directly correspond to the problem formulation (1.1). The remaining problems have additional bound constraints, which are relaxed in this

TABLE 3

Experiment I *compares* 7 *solvers on problems from the SuiteSparse Matrix Collection* [13]. *Entries with* N/A* *denote problems to which* $TR1$ *and* $TR2$ *were not applied because they are too large.* NC† *means the solver did not converge to tolerances.* $TR2H$ *and* $TR1H$ *converged on all problem instances. Overall, the computational times of* $TR2\{H,L\}$ *and* $TR1\{H,L\}$ *were lower by a significant factor compared to the times of* $TR1$, $TR2$, *and IPOPT. The number of iterations for each solver is similar across all problems.*

Problem	m/n	rank(A)	TR2		TR2H		TR2L		TR1		TR1H		TR1L		IPOPT	
			It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec
beacxc	497/506	449/0.2	73	0.52	25	*0.044*	25	0.15	419	3.8	25	**0.041**	25	0.15	NC†	NC
lp_25fv47	821/1876	820/0.007	60	0.82	60	0.21	60	**0.14**	62	0.85	62	0.22	62	*0.14*	61	0.73
lp_agg2	516/758	516/0.01	40	0.21	40	*0.054*	40	**0.052**	42	0.21	42	0.056	42	0.055	41	0.22
lp_agg3	516/758	516/0.01	39	0.21	39	**0.051**	39	*0.051*	39	0.2	39	0.052	39	0.051	44	0.24
lp_bnl1	643/1586	642/0.005	70	0.57	70	0.14	70	*0.079*	67	0.6	67	0.14	67	**0.078**	62	0.59
lp_bnl2	2324/4486	2324/0.001	69	11	69	0.62	69	*0.28*	69	11	69	0.52	69	**0.27**	67	2.2
lp_cre_a	3516/7248	3428/0.0007	N/A*	N/A	83	0.65	83	**0.37**	N/A*	N/A	88	0.71	88	*0.38*	87	3.3
lp_cre_d	8926/73948	6476/0.0004	N/A*	N/A	556	1.2e+02	510	**24**	N/A*	N/A	503	1e+02	552	*25*	NC†	NC
lp_czprob	929/3562	929/0.003	17	0.27	17	0.059	17	*0.032*	17	0.25	17	0.049	17	**0.028**	18	0.34
lp_d6cube	415/6184	404/0.01	35	0.22	35	0.4	35	**0.17**	36	0.21	36	0.44	36	*0.18*	38	1.1
lp_degen3	1503/2604	1503/0.006	39	2.2	39	0.25	39	*0.25*	39	2.3	39	0.27	39	**0.24**	40	1.5
lp_dfl001	6071/12230	6071/0.0005	N/A*	N/A	226	**16**	231	19	N/A*	N/A	226	*16*	238	20	207	1.2e+02
lp_etamacro	400/816	400/0.008	78	0.27	78	0.12	78	**0.085**	86	0.29	86	0.13	86	*0.09*	68	0.44
lp_fffff800	524/1028	524/0.01	NC†	NC	61	**0.095**	NC†	NC	NC†	NC	59	*0.097*	NC†	NC	57	0.45
lp_finnis	497/1064	497/0.005	150	0.72	151	0.2	156	**0.13**	159	0.69	155	0.2	155	*0.14*	167	1.2
lp_fit2d	25/10524	25/0.5	266	1.3	266	*0.9*	258	**0.88**	247	1.4	261	0.91	279	0.99	NC†	NC
lp_ganges	1309/1706	1309/0.003	41	1.3	41	0.11	41	**0.067**	41	1.4	41	0.12	41	*0.073*	37	0.43
lp_gfrd_pnc	616/1160	616/0.003	NC†	NC	54	0.054	54	*0.043*	NC†	NC	54	0.052	54	**0.042**	48	0.36
lp_greenbea	2392/5598	2389/0.002	149	47	149	1.2	149	**0.63**	157	33	153	1.3	150	*0.72*	181	6.8
lp_greenbeb	2392/5598	2389/0.002	149	45	149	1.2	149	*0.65*	157	31	153	1.3	150	**0.65**	181	6.5
lp_grow22	440/946	440/0.02	79	0.24	79	0.079	79	*0.071*	79	0.24	79	0.08	79	**0.069**	65	0.36
lp_ken_07	2426/3602	2426/0.001	34	12	34	0.091	34	**0.067**	34	7.4	34	0.093	34	*0.07*	31	0.85
lp_maros	846/1966	846/0.006	74	0.87	74	*0.21*	NC†	NC	74	0.86	74	**0.21**	NC†	NC	71	0.92
lp_maros_r7	3136/9408	3136/0.005	N/A*	N/A	57	*2.1*	57	2.2	N/A*	N/A	57	**2.1**	57	2.3	51	25
lp_modszk1	687/1620	686/0.003	71	0.51	71	0.13	71	**0.07**	71	0.51	71	0.14	71	*0.071*	70	0.53
lp_osa_30	4350/104374	4350/0.001	N/A*	N/A	46	8.5	46	**1.9**	N/A*	N/A	45	8.8	45	*2*	43	30
lp_osa_60	10280/243246	10280/0.0006	N/A*	N/A	47	24	47	**5.9**	N/A*	N/A	44	23	44	*5.9*	42	1.1e+02
lp_pds_02	2953/7716	2953/0.0007	N/A*	N/A	25	0.25	25	*0.14*	N/A*	N/A	25	0.24	25	**0.099**	26	1.3
lp_pds_10	16558/49932	16558/0.0001	N/A*	N/A	61	13	61	*8.1*	N/A*	N/A	60	13	60	**7.9**	59	62
lp_perold	625/1506	625/0.007	58	0.39	58	0.16	58	**0.084**	58	0.38	58	0.16	58	*0.087*	57	0.59
lp_pilot	1441/4860	1441/0.006	105	13	105	1.3	105	**0.83**	109	6.2	109	1.3	109	*0.97*	117	5.8
lp_pilot87	2030/6680	2030/0.006	102	17	102	2.6	102	**1.9**	104	15	104	2.7	104	*1.9*	110	15
lp_pilot_we	722/2928	722/0.004	73	0.69	73	0.24	73	*0.11*	73	0.65	73	0.21	73	**0.11**	81	1.4
lp_pilotnov	975/2446	975/0.006	77	1.3	77	**0.35**	NC†	NC	77	1.3	77	*0.35*	NC†	NC	78	1.3
lp_qap12	3192/8856	3192/0.001	N/A*	N/A	27	*3.3*	27	3.7	N/A*	N/A	26	**3.1**	26	3.6	25	1.4e+02
lp_qap8	912/1632	912/0.005	20	0.42	20	0.15	20	*0.11*	22	0.31	22	0.15	22	**0.1**	21	1.9
lp_scfxm1	330/600	330/0.01	45	0.1	45	0.042	45	*0.036*	44	0.098	44	0.041	44	**0.036**	44	0.19
lp_scfxm2	660/1200	660/0.007	52	0.42	52	0.079	52	**0.056**	57	0.43	57	0.094	57	*0.06*	55	0.46
lp_scfxm3	990/1800	990/0.005	45	0.8	45	0.096	45	**0.061**	45	0.76	45	0.097	45	*0.062*	48	0.54
lp_scsd1	77/760	77/0.04	74	0.035	74	0.035	74	0.044	74	**0.033**	74	*0.034*	74	0.043	NC†	NC
lp_scsd6	147/1350	147/0.02	84	0.077	84	**0.054**	84	0.06	92	0.084	92	*0.06*	92	0.065	75	0.34
lp_scsd8	397/2750	397/0.008	66	0.21	66	0.07	66	**0.066**	65	0.21	65	0.069	65	*0.066*	66	0.54
lp_sctap1	300/660	300/0.009	107	0.25	107	0.088	107	*0.075*	102	0.24	102	0.081	102	**0.07**	100	0.45
lp_sctap2	1090/2500	1090/0.003	145	5.5	146	0.43	146	**0.18**	145	3.6	143	0.46	146	*0.19*	157	2.3
lp_sctap3	1480/3340	1480/0.002	204	27	205	0.75	201	*0.39*	199	13	197	0.74	202	**0.39**	220	4.3
lp_ship04l	402/2166	360/0.007	84	0.25	84	0.12	84	**0.077**	84	0.27	84	0.12	84	*0.08*	92	0.84
lp_ship04s	402/1506	360/0.007	74	0.17	74	0.063	74	**0.053**	74	0.16	74	0.065	74	*0.055*	71	0.48
lp_stair	356/614	356/0.02	47	0.11	47	0.047	47	*0.046*	47	0.11	47	0.047	47	**0.045**	47	0.23
lp_standata	359/1274	359/0.007	78	0.22	78	0.072	78	*0.058*	79	0.21	79	0.067	79	**0.057**	80	0.65
lp_standmps	467/1274	467/0.007	52	0.21	52	0.06	52	**0.042**	52	0.21	52	0.065	52	*0.043*	58	0.48

experiment. Problems 1–19 in Table 5 are convex and can immediately be attempted by the solvers (with bounds released). Problems 20–31 are not convex when the bounds are relaxed, but adding the term $\frac{\delta}{2}\|x\|_2^2$ with $\delta = 10$ to the objective functions produced finite solutions for these problems. As in the previous experiment, convergence is determined by each algorithm internally. For TR1, TR1H, TR1L, TR2, TR2H, TR2L the conditions $\|Pg_k\|_\infty < 1 \times 10^{-5}$ and $\|Ax_k - b\|_2 < 5 \times 10^{-8}$ are explicitly enforced, while for IPOPT we set `options_ipopt.ipopt.tol=1e-5`. We use the iteration limit of 100,000 for all solvers. The limited-memory parameter is $l = 5$ for all TR solvers and $l = 6$ (default) for IPOPT. Since TR1 and TR2 are not designed for large m, they are applied to problems with $m < 2500$, with the exception of 3 problems

Table 4

Experiment II *compares* 7 *solvers on* 61 *large problems from the CUTEst collection* [22]. NC† *means the solver did not converge to tolerances.* MX† *means the iteration limit was reached.* *TR1L converged on* 58 *problems, the largest number of problems among the solvers. TR2H was faster than TR2 on* 51 *problems, and TR2L was faster than TR2 on* 46 *problems (the differences are often significant). TR1H was faster than TR1 on* 49 *problems, and TR1L was faster than TR1 on* 41 *problems (often significantly). All of TR1{H,L} and TR2{H,L} were faster than IPOPT.*

Problem	m/n	TR2		TR2H		TR2L		TR1		TR1H		TR1L		IPOPT	
		It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec
ARWHEAD	1250/5000	343	1.7e+02	349	19	372	19	264	72	304	**16**	315	*16*	NC†	NC
BDQRTIC	1250/5000	181	50	174	**8.1**	187	9.9	174	31	186	8.9	160	*8.4*	78	1.2e+02
BOX	2500/10000	240	1.5e+03	280	63	281	79	218	2.1e+02	258	**54**	208	*58*	NC†	NC
BROYDN7D	1250/5000	355	20	370	*18*	367	18	355	20	370	**17**	381	19	432	6.5e+02
BRYBND	1250/5000	897	1.5e+02	883	**45**	1273	64	1396	1.2e+02	1177	*60*	1421	70	1027	1.7e+03
COSINE	2500/10000	NC†	NC	5028	1e+03	4527	1.2e+03	4755	2e+03	7318	1.6e+03	3292	*910*	NC†	NC
CRAGGLVY	1250/5000	373	63	371	**18**	369	*19*	400	45	390	20	397	21	205	3.4e+02
CURLY10	2500/10000	1563	7.2e+02	2498	5.3e+02	1496	*429*	1512	4.5e+02	1549	**347**	1759	4.9e+02	1775	3e+04
CURLY20	2500/10000	1951	9.5e+02	2015	**455**	1993	*552*	3149	9.5e+02	4110	8.7e+02	3836	1.1e+03	NC†	NC
CURLY30	2500/10000	4457	2.8e+03	4210	*952*	3669	1e+03	2744	**783**	6940	1.6e+03	6145	1.7e+03	NC†	NC
DIXMAANA	750/3000	10	0.53	10	**0.43**	10	0.51	10	0.5	10	0.47	10	*0.46*	13	8.3
DIXMAANB	750/3000	9	0.55	9	0.5	9	*0.5*	9	0.59	9	**0.5**	9	0.55	11	8.1
DIXMAANC	750/3000	12	0.73	12	0.67	12	0.72	12	*0.65*	12	**0.63**	12	0.69	14	10
DIXMAAND	750/3000	23	1.7	23	1.1	23	1.2	22	*0.93*	22	**0.83**	22	1	27	16
DIXMAANE	750/3000	35	1.1	35	1	35	1.1	35	**0.83**	35	*0.88*	35	1.1	41	18
DIXMAANF	750/3000	183	5.2	194	**3.9**	194	5.7	194	6.6	195	*4.9*	203	6.7	297	1.3e+02
DIXMAANG	750/3000	434	19	397	**8.3**	439	12	435	13	408	*9.8*	404	11	NC†	NC
DIXMAANH	750/3000	433	14	470	11	454	13	459	*11*	421	**9.3**	443	12	422	1.8e+02
DIXMAANI	750/3000	82	2	82	*1.8*	82	2.4	82	**1.6**	82	1.8	82	2.5	103	46
DIXMAANJ	750/3000	1054	41	1506	35	1023	*27*	1415	42	1490	34	944	**24**	NC†	NC
DIXMAANK	750/3000	2971	1e+02	3026	65	3082	71	2831	80	2870	**61**	2691	*62*	NC†	NC
DIXMAANL	750/3000	1461	**38**	3198	69	2609	60	2690	66	2728	*58*	2597	59	NC†	NC
DIXON3DQ	2500/10000	51	17	51	*12*	51	17	51	17	51	**12**	51	16	56	6.7e+02
DQDRTIC	1250/5000	13	1.7	7	0.85	7	0.77	13	1.5	7	**0.75**	7	*0.75*	7	13
DQRTIC	1250/5000	63	*4.6*	107	6.7	107	7	63	**4.5**	107	5.7	107	6.1	93	1.5e+02
EDENSCH	500/2000	32	*0.33*	32	0.4	32	0.38	32	**0.32**	32	0.39	32	0.36	34	5
EG2	250/1000	423	2.2	504	*1.3*	439	**1.3**	514	5.2	624	2	502	1.9	908	23
ENGVAL1	1250/5000	31	2.7	31	**1.8**	31	2	31	2.6	31	*1.9*	31	2	38	61
EXTROSNB	250/1000	148	*0.44*	148	0.45	148	0.49	145	0.53	145	0.46	145	**0.39**	129	3
FLETCHCR	250/1000	150	*0.4*	150	0.46	150	0.51	150	**0.37**	150	0.42	150	0.41	137	3.1
FMINSRF2	1407/5625	122	10	122	*9.3*	122	10	122	10	122	**7.8**	122	9.6	167	4e+02
FREUROTH	1250/5000	287	1e+02	247	**12**	235	*13*	274	37	255	13	234	13	202	3.2e+02
GENHUMPS	1250/5000	2215	1.2e+02	1762	99	1829	**93**	2215	1.3e+02	1762	98	1829	*95*	NC†	NC
LIARWHD	1250/5000	3854	1.6e+03	3998	4.4e+02	2726	*196*	2638	1.2e+03	2408	2.6e+02	1591	**128**	NC†	NC
MOREBV	1250/5000	151	23	151	22	151	20	151	19	151	*16*	151	**16**	NC†	NC
MSQRTALS	256/1024	MX†	MX	MX†	MX	MX†	MX	MX†	MX	78461	6.6e+02	99724	*620*	NC†	NC
MSQRTBLS	256/1024	MX†	MX	MX†	MX	MX†	MX	MX†	MX	MX†	MX	MX†	MX	NC†	NC
NCB20	1253/5010	345	47	348	18	349	18	314	33	317	**16**	307	*16*	252	3.9e+02
NONCVXU2	1250/5000	185	20	185	**9**	185	9.5	186	14	187	*9.2*	186	9.4	120	1.9e+02
NONCVXUN	1250/5000	282	33	283	**14**	282	*14*	360	31	354	17	370	19	199	3.1e+02
NONDIA	1250/5000	1612	6.9e+02	1600	88	1734	*88*	2764	7.3e+02	1407	**78**	1907	98	NC†	NC
NONDQUAR	1250/5000	897	4.3e+02	865	47	811	**42**	816	2.1e+02	876	47	857	*44*	332	8.1e+02
PENALTY1	250/1000	8	0.051	2	0.018	2	*0.017*	8	0.056	2	0.019	2	**0.016**	1	0.043
POWELLSG	1250/5000	88	6.1	88	**4.4**	88	4.6	88	5.6	88	*4.4*	88	4.6	99	1.5e+02
POWER	2500/10000	51	**17**	MX†	MX	MX†	MX	51	*17*	MX†	MX	MX†	MX	62	6.9e+02
QUARTC	1250/5000	70	*4.9*	104	5.3	104	5.4	70	**4.5**	104	5.1	104	5.6	89	1.4e+02
SCHMVETT	1250/5000	MX†	MX	70882	3.9e+03	MX†	MX	NC†	NC	MX†	MX	96572	5.1e+03	NC†	NC
SINQUAD	1250/5000	236	56	282	15	214	**11**	247	32	216	*12*	277	14	116	1.8e+02
SPARSQUR	2500/10000	35	13	43	*10*	43	14	35	13	43	**9.9**	43	14	31	3.5e+02
SPMSRTLS	1250/4999	2222	2.7e+02	1791	**95**	2377	1.2e+02	2792	2e+02	2475	1.3e+02	1834	*98*	NC†	NC
SROSENBR	1250/5000	5561	4.1e+02	8211	4.3e+02	4814	**235**	6400	4.3e+02	6747	3.6e+02	5280	*270*	NC†	NC
TOINTGSS	1250/5000	39	3.1	39	**2.2**	39	2.3	39	3	39	2.3	39	*2.3*	49	76
TQUARTIC	1250/5000	2069	8.7e+02	1155	**64**	1508	*78*	1494	3.7e+02	1867	1e+02	1871	98	NC†	NC
TRIDIA	1250/5000	147	9	82	**4.2**	82	4.3	147	9.1	82	*4.2*	82	4.4	66	1e+02
WOODS	1000/4000	1192	45	1157	38	1077	**27**	1236	44	1167	37	1132	*29*	971	1.3e+03
SPARSINE	1250/5000	1504	1.7e+02	1476	79	1464	**74**	2188	1.6e+02	1407	*74*	3999	2e+02	2294	5.6e+03
TESTQUAD	1250/5000	10988	**623**	14186	7.3e+02	13357	6.5e+02	10988	*643*	14186	7.3e+02	13357	6.6e+02	NC†	NC
JIMACK	888/3549	NC†	NC	NC†	NC	NC†	NC	NC†	NC	NC†	NC	NC†	NC	NC†	NC
NCB20B	1250/5000	57	4.1	56	3.2	56	3.2	57	4.2	56	**3.1**	56	*3.2*	47	73
EIGENALS	638/2550	202	*3.2*	204	3.7	203	4.1	202	**3**	204	3.6	203	4	161	43
EIGENBLS	638/2550	28	0.59	28	0.65	28	0.6	28	**0.51**	28	*0.52*	28	0.62	28	7.7

(BLOWEYA, BLOWEYB, BLOWEYC) that did not terminate within hours using TR1 and TR2. All other solvers are applied to all problems. The results are in Figure 3 and Table 5.

Table 5

Experiment III *compares* 7 *solvers on* 31 *linear equality constrained problems from the CUTEst collection* [22]. `NC`† *means the solver did not converge to tolerances.* `N/A` *means that* $TR1$ *and* $TR2$ *were not applied because the problem size rendered them not practical.* $TR2H$ *and* $TR1H$ *converged on all* 31 *problems.* $TR2L$, $TR1L$, *and* $IPOPT$ *converged on* 30 *problems (the exception is* `CVXQP2`*). The fastest and second fastest solvers for each problem are highlighted in bold and italic fonts, respectively. Overall,* $TR2H$ *was fastest on* 12 *problems (the best outcome on this experiment), while* $TR1L$ *was fastest on* 11 *problems (the second best outcome). Problems* `A0ESDNDL` *and* `A0ESINDL` *contain dense columns in* A, *and the sparse QR factorization takes additional time as seen from the entries of* $TR2H$ *and* $TR1H$. *However, preconditioned* $LSQR$ *can overcome this difficulty, as observed in the entries for* $TR2L$ *and* $TR1L$ *for these problem instances.*

Problem	m/n	TR2		TR2H		TR2L		TR1		TR1H		TR1L		IPOPT	
		It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec	It	Sec
AUG2D	10000/20200	N/A*	N/A	7	0.26	7	*0.15*	N/A*	N/A	7	0.24	7	**0.13**	12	1.4
AUG2DC	10000/20200	N/A*	N/A	2	0.11	2	*0.067*	N/A*	N/A	2	0.1	2	**0.067**	1	0.15
AUG2DCQP	10000/20200	N/A*	N/A	2	0.11	2	*0.072*	N/A*	N/A	2	0.11	2	**0.07**	1	0.16
AUG2DQP	10000/20200	N/A*	N/A	7	0.23	7	**0.13**	N/A*	N/A	7	0.24	7	*0.13*	12	1.4
AUG3D	8000/27543	N/A*	N/A	10	0.68	10	*0.52*	N/A*	N/A	10	0.6	10	**0.51**	11	2.6
AUG3DC	8000/27543	N/A*	N/A	2	0.3	2	*0.28*	N/A*	N/A	2	0.3	2	**0.26**	1	0.31
AUG3DCQP	8000/27543	N/A*	N/A	2	0.3	2	*0.27*	N/A*	N/A	2	0.33	2	**0.26**	1	0.33
AUG3DQP	8000/27543	N/A*	N/A	10	0.74	10	*0.55*	N/A*	N/A	10	0.64	10	**0.5**	11	2.6
CVXQP1	5000/10000	N/A*	N/A	827	7.8	805	**3.8**	N/A*	N/A	827	7.3	805	*3.8*	740	51
CVXQP2	2500/10000	N/A*	N/A	39596	1.5e+02	NC†	NC	N/A*	N/A	47572	1.8e+02	NC†	NC	NC†	NC
CVXQP3	7500/10000	N/A*	N/A	169	2.8	169	*1.4*	N/A*	N/A	169	2.4	169	**1.4**	118	8.9
STCQP1	4095/8193	N/A*	N/A	88	**0.15**	88	0.42	N/A*	N/A	88	*0.18*	88	0.36	75	6.8e+02
STCQP2	4095/8193	N/A*	N/A	142	**0.25**	142	0.8	N/A*	N/A	144	*0.28*	144	0.72	136	4.8
DTOC1L	3996/5998	N/A*	N/A	13	**0.073**	13	0.13	N/A*	N/A	13	*0.075*	13	0.14	16	0.41
DTOC3	2998/4499	N/A*	N/A	5	**0.025**	5	0.059	N/A*	N/A	5	*0.03*	5	0.033	4	0.09
PORTSQP	1/100000	2	0.09	2	0.064	2	0.067	2	*0.062*	2	**0.059**	2	0.062	1	0.42
HUES-MOD	2/5000	1	0.0028	1	**0.0018**	1	0.0027	1	0.0026	1	*0.0018*	1	0.0026	1	0.024
HUESTIS	2/5000	2	0.0073	2	**0.0042**	2	0.011	2	0.0061	2	*0.0047*	2	0.0094	2	0.072
AOESDNDL	15002/45006	N/A*	N/A	5	69	5	*0.13*	N/A*	N/A	5	71	5	**0.12**	6	1.8
AOESINDL	15002/45006	N/A*	N/A	5	73	5	*0.12*	N/A*	N/A	5	70	5	**0.11**	6	1.8
PORTSNQP	2/100000	NC†	NC	2	**0.092**	2	0.11	14	0.47	2	*0.095*	2	0.1	2	0.88
BLOWEYA	2002/4002	N/A*	N/A	2	**0.011**	2	0.031	N/A*	N/A	2	*0.015*	2	0.021	2	0.082
BLOWEYB	2002/4002	N/A*	N/A	2	**0.015**	2	0.019	N/A*	N/A	2	*0.016*	2	0.019	2	0.082
BLOWEYC	2002/4002	N/A*	N/A	2	**0.015**	2	0.017	N/A*	N/A	2	*0.015*	2	0.021	2	0.15
CONT5-QP	40200/40601	N/A*	N/A	2	*0.51*	2	0.79	N/A*	N/A	2	**0.49**	2	0.8	2	1.3
DTOC1L	3996/5998	N/A*	N/A	5	**0.03**	5	0.09	N/A*	N/A	5	*0.043*	5	0.045	4	0.12
FERRISDC	210/2200	2	0.084	2	0.083	2	0.077	2	*0.076*	2	0.078	2	0.079	0	**0.021**
GOULDQP2	9999/19999	N/A*	N/A	2	0.038	2	**0.025**	N/A*	N/A	2	0.038	2	*0.026*	2	0.2
GOULDQP3	9999/19999	N/A*	N/A	6	0.076	6	*0.054*	N/A*	N/A	6	0.077	6	**0.053**	7	0.69
LINCONT	419/1257	5	0.058	5	*0.02*	5	0.031	5	0.05	5	**0.019**	5	0.03	5	0.055
SOSQP2	2501/5000	N/A*	N/A	3	**0.017**	3	0.04	N/A*	N/A	3	0.022	3	*0.019*	4	0.11

10. Conclusion.

10. Conclusion. For subproblem (1.2), this article develops the RCR of the (1,1) block in the inverse KKT matrix, when the objective Hessian is approximated by a compact quasi-Newton matrix. The representation is based on the fact that part of the solution to the KKT system is unaffected when it is projected onto the nullspace of the constraints. An advantage of the RCR is that it enables a decoupling of solves with the constraint matrix and remaining small terms. Moreover, a projected gradient can be used in two places: once as part of the matrix update and second as part of the new step. By effectively handling orthogonal projections, in combination with limited-memory techniques, we can compute search directions efficiently. We apply the orthogonal projections with a sparse QR factorization or a preconditioned LSQR iteration, including large and potentially rank-deficient constraints. The RCRs are implemented in two trust-region algorithms, one of which exploits the underlying matrix structures in order to compute the search direction by an analytic formula. The other is based on an ℓ_2-norm and uses the RCR within a 1D Newton iteration to determine the optimal scalar shift. In numerical experiments on large problems, our implementations of the RCR yield often significant improvements in the computation time as a result of the advantageous structure of the proposed matrices.

Applications of problem (1.1) often include bounds $\ell \le x \le u$. When second derivatives of the objective function are available, the problem is best handled by an interior method. Otherwise, a barrier function could be added to the objective, and the methods here may sometimes be effective on a sequence of large equality constrained subproblems.

Appendix A. Here we describe a simplified expression for the matrix $C_k^\top G_k C_k$ from section 4.2. Recall that the L-BFGS inverse $B_k^{-1} = \delta_k I + J_k W_k J_k^\top$ is defined by

$$J_k = \begin{bmatrix} S_k & Y_k \end{bmatrix}, \quad W_k = \begin{bmatrix} T_k^{-\top}(D_k + \delta_k Y_k^\top Y_k)T_k^{-1} & -\delta_k T_k^{-\top} \\ -\delta_k T_k^{-1} & 0_{l\times l} \end{bmatrix}.$$

First, note that

$$C_k \equiv A J_k W_k = \begin{bmatrix} 0 & AY_k \end{bmatrix} W_k = \begin{bmatrix} -\delta_k A Y_k T_k^{-1} & 0 \end{bmatrix}.$$

Second, it holds that

$$G_k^{-1} \equiv A B_k^{-1} A^\top = \delta_k A A^\top + A J_k W_k J_k^\top A^\top = \delta_k A A^\top + C_k \begin{bmatrix} 0 \\ (AY_k)^\top \end{bmatrix},$$

so that $G_k^{-1} = \delta_k A A^\top$, because the last term in the above expression for G_k^{-1} vanishes. Multiplying $C_k^\top$, G_k, and C_k we see that

$$C_k^\top G_k C_k = \begin{bmatrix} \delta_k T_k^{-\top} Y_k^\top A^\top (AA^\top)^{-1} A Y_k T_k^{-1} & 0_{l\times l} \\ 0_{l\times l} & 0_{l\times l} \end{bmatrix}.$$

Appendix B. This appendix describes how we apply the functions from the SuiteSparse library [12] in our implementations. We use SuiteSparse version 5.8.1 from https://github.com/DrTimothyAldenDavis/SuiteSparse/releases.

B.1. Householder QR projection. The MATLAB commands to compute the projection Pg_k using a Householder QR factorization are listed in Table 1.

B.2. Preconditioned LSQR projection. The MATLAB commands to compute the projection Pg_k using preconditioned LSQR [28] are listed in Table 2.

Appendix C. This appendix overviews the subproblem solution with the shape-changing norm. Note that $U = \begin{bmatrix} Q_1 & U_2 & U_3 \end{bmatrix} \in \mathbb{R}^{n\times n}$ (from section 7) represents an orthogonal matrix and that the quadratic function is

$$q(s) = s^\top g_k + \frac{1}{2} s^\top B_k s = s^\top U U^\top g_k + \frac{1}{2} s^\top U U^\top B_k U U^\top s.$$

We introduce the change of variables $v^\top = \begin{bmatrix} v_1^\top & v_2^\top & v_3^\top \end{bmatrix} \equiv s^\top U$. Moreover, it holds that

$$U^\top B_k U = \begin{bmatrix} Q_1^\top B_k Q_1 & Q_1^\top B_k U_2 & Q_1^\top B_k U_3 \\ U_2^\top B_k Q_1 & (\delta_k I + \Lambda_2)^{-1} & \\ U_3^\top B_k Q_1 & & \delta_k^{-1} I \end{bmatrix}$$

(cf. [6, Lemma 2]) and that

$$AUU^\top s = AUv = \begin{bmatrix} R & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = Rv_1.$$

With the constraint $As = 0 = AUv$, this implies $v_1 = 0$ (for R nonsingular). Therefore, the trust-region subproblem defined by the shape-changing norm decouples into a problem with v_2 and v_3 only (once $v_1 = 0$ is fixed):

$$\underset{\substack{\|s\|_U \le \Delta_k \\ As = 0}}{\text{minimize}}\ q(s) = \left\{ \underset{\|v_2\|_\infty \le \Delta_k}{\text{minimize}}\ v_2^\top U_2^\top g_k + \frac{1}{2} v_2^\top (\delta_k I + \Lambda_2)^{-1} v_2 \right.$$
$$\left. +\ \underset{\|v_3\|_2 \le \Delta_k}{\text{minimize}}\ v_3^\top U_3^\top g_k + \frac{\|v_3\|_2^2}{2\delta_k} \right\}.$$

This reformulated subproblem can be solved analytically, and the componentwise solution of v_2 is in (8.1). The analytic solution of v_3 is $v_3 = \beta U_3^\top g_k$ with β from (8.2). Subsequently, s is obtained by transforming variables as $s = Uv = U_2 v_2 + U_3 v_3$. The orthonormal matrix U_2 is computed as $U_2 = \begin{bmatrix} S_k & Z_k \end{bmatrix} \hat{R}_2^{-1} \hat{P}_2$, and since $U_3 U_3^\top = P - U_2 U_2^\top$, the optimal step with the shape-changing norm is as in (8):

$$s_{SC} = U_2(v_2 - \beta U_2^\top g_k) + \beta P g_k.$$

With $u_k \equiv U_2^\top g_k$, the step is then computed as in Algorithm 8.1 (line 15):

$$s_{SC} = \begin{bmatrix} S_k & Z_k \end{bmatrix} \hat{R}_2^{-1} \hat{P}_2 (v_2 - \beta u_k) + \beta P g_k.$$

Appendix D.

D.1. Detailed table for Experiment I.

In this experiment the degree of difficulty in solving a problem depends largely on handling A because the structure of the objective function is the same for all instances. We observe that our proposed algorithms (any of TR1{H,L}, TR2{H,L}) always use less computation time (often significantly), except for two problem instances. On problem `lp_d6cube`, TR2 used less time than TR2H, as did TR1 over TR1H. However, the "L" versions were fastest overall on this problem. On problem `lp_scsd1`, TR1 used the least time. In these two problems the number of constraints is not large, and one can expect that TR1, TR2 do comparatively well. However, for all other 48 problems the new methods used the least time. We observe that both "H" versions converged to the prescribed tolerances on all problems. On the other hand, the "L" versions are often the overall fastest, yet they did not converge on 3 problem instances (`beacxc`, `lp_cre_d`, `fit2d`).

D.2. Detailed table for Experiment II.

In Experiment II, the objective functions for each problem are defined by a large CUTEst problem, whereas the corresponding A matrices are not meant to be overly challenging. We observe that the proposed algorithms (the ones including "{H,L}") improve the computation times on the majority of problems. For the 10 instances in which TR2 used less time than TR2H, the differences are relatively small. An exception is `DIXMAANL`, where the difference amounts to 31s. However, for the other 51 problems, TR2H resulted in often significant improvements in computation time. For instance, in `LIARWHD` this difference amounts to 1182s (more than 19 minutes). These observations carry over when comparing TR1 with TR1H. The "L" versions exhibit similar outcomes as the "H" ones, with occasional increases in computation times. Overall, TR1L converged to the specified tolerances on the largest number of problems. The problems reported as "`NC`" in IPOPT's column correspond to status flags other than "0, 1, 2" ≡ "solved, solved to acceptable level, infeasible problem detected."

D.3. Detailed table for Experiment III. In Experiment III, TR2H and TR1H converged on all 31 problems, while all other solvers (besides TR1 and TR2) converged on all problems except one: `CVXQP2`. TR2H was the fastest on 10 problems (the best outcome among the solvers), while TR1L was the fastest on 9 problems (the second best outcome). Problems `AOESDNDL` and `AOESINDL` appear noteworthy: they contain dense columns (satisfying the condition $\mathrm{nnz}(A_{:,j})/m > 0.1$). Sparse QR factorization is expensive because of fill-in. However, the iterative method LSQR (with the preconditioning technique from section 5.2) can overcome these difficulties.

Acknowledgments. We would like to acknowledge the valuable discussions initiated by Ariadna Cairo Baza and spurred by the 9th ICIAM conference at the Universidad de Valencia. We thank two referees for their extremely detailed and helpful comments.

REFERENCES

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2011), pp. 1–122, https://doi.org/10.1561/2200000016.

[2] C. G. Broyden, *The convergence of a class of double-rank minimization algorithms* 1. *General considerations*, IMA J. Appl. Math., 6 (1970), pp. 76–90, https://doi.org/10.1093/imamat/6.1.76.

[3] J. Brust, O. Burdakov, J. Erway, and R. Marcia, *A dense initialization for limited-memory quasi-Newton methods*, Comput. Optim. Appl., 74 (2019), pp. 121–142.

[4] J. J. Brust, *Large-Scale Quasi-Newton Trust-Region Methods: High-Accuracy Solvers, Dense Initializations, and Extensions*, PhD thesis, University of California, Merced, 2018, https://escholarship.org/uc/item/2bv922qk.

[5] J. J. Brust, J. B. Erway, and R. F. Marcia, *On solving L-SR*1 *trust-region subproblems*, Comput. Optim. Appl., 66 (2017), pp. 245–266.

[6] J. J. Brust, R. F. Marcia, and C. G. Petra, *Large-scale quasi-Newton trust-region methods with low-dimensional linear equality constraints*, Comput. Optim. Appl., 74 (2019), pp. 669–701, https://doi.org/10.1007/s10589-019-00127-4.

[7] O. Burdakov, L. Gong, Y.-X. Yuan, and S. Zikrin, *On efficiently combining limited memory and trust-region techniques*, Math. Program. Comput., 9 (2016), pp. 101–134.

[8] 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), pp. 129–156.

[9] R. H. Byrd, J. Nocedal, and R. A. Waltz, *Knitro: An Integrated Package for Nonlinear Optimization*, Springer US, Boston, MA, 2006, pp. 35–59, https://doi.org/10.1007/0-387-30065-1_4.

[10] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, SIAM, Philadelphia, PA, 2000.

[11] T. A. Davis, *Algorithm* 915, *SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization*, ACM Trans. Math. Software, 38 (2011), pp. 8:1–22.

[12] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), p. 25.

[13] T. A. Davis, Y. Hu, and S. Kolodziej, *SuiteSparse Matrix Collection*, https://sparse.tamu.edu/, 2015–present.

[14] O. DeGuchy, J. B. Erway, and R. F. Marcia, *Compact representation of the full Broyden class of quasi-Newton updates*, Numer. Linear Algebra Appl., 25 (2018), e2186.

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

[16] R. Fletcher, *A new approach to variable metric algorithms*, Comput. J., 13 (1970), pp. 317–322, https://doi.org/10.1093/comjnl/13.3.317.

[17] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971, https://doi.org/10.1137/10079687X.

[18] A. Fu, J. Zhang, and S. Boyd, *Anderson accelerated Douglas–Rachford splitting*, SIAM J. Sci. Comput., 42 (2020), pp. A3560–A3583, https://doi.org/10.1137/19M1290097.

[19] P. E. Gill and W. Murray, *Numerical Methods for Constrained Optimization*, Academic Press, London, 1974.

[20] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26, https://doi.org/10.1090/S0025-5718-1970-0258249-6.

[21] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins Stud. Math. Sci., Johns Hopkins University Press, Baltimore, MD, 2013.

[22] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEr and SifDec: A constrained and unconstrained testing environment, revisited*, ACM Trans. Math. Software, 29 (2003), pp. 373–394.

[23] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bureau. Standards, 49 (1952), pp. 409–436.

[24] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Program., 45 (1989), pp. 503–528.

[25] A. Mahajan, S. Leyffer, and C. Kirches, *Solving Mixed-Integer Nonlinear Programs by QP Diving*, Technical Report ANL/MCS-P2071-0312, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, 2012.

[26] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35 (1980), pp. 773–782.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer-Verlag, New York, 2006.

[28] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982a), pp. 43–71, https://doi.org/10.1145/355984.355989.

[29] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–656, https://doi.org/10.1090/S0025-5718-1970-0274029-X.

[30] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program., 106 (2006), pp. 25–57.

[31] H. Zhang and W. W. Hager, *A nonmonotone line search technique and its application to unconstrained optimization*, SIAM J. Optim., 14 (2004), pp. 1043–1056, https://doi.org/10.1137/S1052623403428208.

[32] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, *Algorithm* 778: *L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization*, ACM Trans. Math. Software, 23 (1997), pp. 550–560, https://doi.org/10.1145/279232.279236.