ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Compact Representations of BFGS Matrices

J. J. Brust, S. Leyffer, and C. G. Petra

Mathematics and Computer Science Division

Preprint ANL/MCS-P9279-0120

January 2020

[1]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. through the Project "Multifaceted Mathematics for Complex Energy Systems." This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

COMPACT REPRESENTATIONS OF STRUCTURED BFGS MATRICES*

JOHANNES J. BRUST†, SVEN LEYFFER†, AND COSMIN G. PETRA‡

Abstract. For general large-scale optimization problems compact representations exist in which recursive quasi-Newton update formulas are represented as compact matrix factorizations. For problems in which the objective function contains additional structure, so-called structured quasi-Newton methods exploit available second-derivative information and approximate unavailable second derivatives. This article develops the compact representations of two structured Broyden-Fletcher-Goldfarb-Shanno update formulas. The compact representations enable efficient limited memory and initialization strategies. Two limited memory line search algorithms are described and tested on a collection of problems.

Key words. Quasi-Newton method, limited memory method, large-scale optimization, compact representation, BFGS method

AMS subject classifications. 65K05 65F30 90C53 90C06

1. Introduction.

The unconstrained minimization problem is

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x}), \tag{1.1}$$

where $f:\mathbb{R}^n\to\mathbb{R}$ is assumed to be twice continuously differentiable. If the Hessian matrix $\nabla^2 f(\mathbf{x})\in\mathbb{R}^{n\times n}$ is unavailable, because it is unknown or difficult to compute, then quasi-Newton methods are effective methods, which approximate properties of the Hessian at each iteration, $\nabla^2 f(\mathbf{x}_{k+1})\approx\mathbf{B}_{k+1}$ [7]. Arguably, the most popular quasi-Newton matrix is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) matrix [3, 11, 13, 19], because of its convincing results on many problems. Given $\mathbf{s}_k\equiv\mathbf{x}_{k+1}-\mathbf{x}_k$ and $\mathbf{y}_k\equiv\nabla f(\mathbf{x}_{k+1})-\nabla f(\mathbf{x}_k)$ the BFGS recursive update formula is

$$\mathbf{B}_{k+1}=\mathbf{B}_k-\frac{1}{\mathbf{s}_k^T\mathbf{B}_k\mathbf{s}_k}\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^T\mathbf{B}_k+\frac{1}{\mathbf{s}_k^T\mathbf{y}_k}\mathbf{y}_k\mathbf{y}_k^T. \tag{1.2}$$

For a symmetric positive definite initialization $\mathbf{B}_0\in\mathbb{R}^{n\times n}$ (1.2) generates symmetric positive definite matrices as long as $\mathbf{s}_k^T\mathbf{y}_k>0$ for all $k\geq 0$ (see [11, Section 2]).

1.1. BFGS Compact Representation.

Byrd et al. [5] propose the compact representation of the recursive formula (1.2). The compact representation has been successfully used for large-scale unconstrained and constrained optimization [24]. Let the sequence of pairs $\{\mathbf{s}_i,\mathbf{y}_i\}_{i=0}^{k-1}$ be given, and let these vectors be collected in the matrices $\mathbf{S}_k=[\mathbf{s}_0,\ \cdots,\ \mathbf{s}_{k-1}]\in\mathbb{R}^{n\times k}$ and $\mathbf{Y}_k=[\mathbf{y}_0,\ \cdots,\ \mathbf{y}_{k-1}]\in\mathbb{R}^{n\times k}$. Moreover, let $\mathbf{S}_k^T\mathbf{Y}_k=\mathbf{L}_k+\mathbf{R}_k$, where $\mathbf{L}_k\in\mathbb{R}^{k\times k}$ is the strictly lower triangular matrix, $\mathbf{R}_k\in\mathbb{R}^{k\times k}$ is the upper triangular matrix (including the diagonal), and $\mathbf{D}_k=\text{diag}(\mathbf{S}_k^T\mathbf{Y}_k)\in\mathbb{R}^{k\times k}$ is the diagonal part of $\mathbf{S}_k^T\mathbf{Y}_k$. The compact representation of the BFGS formula (1.2) is [5, Theorem 2.3]:

$$\mathbf{B}_k=\mathbf{B}_0-\begin{bmatrix}\mathbf{B}_0\mathbf{S}_k & \mathbf{Y}_k\end{bmatrix}\begin{bmatrix}\mathbf{S}_k^T\mathbf{B}_0\mathbf{S}_k & \mathbf{L}_k\\ \mathbf{L}_k^T & -\mathbf{D}_k\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{S}_k^T\mathbf{B}_0\\ \mathbf{Y}_k^T\end{bmatrix}. \tag{1.3}$$

*Submitted to the editors DATE.

Funding: This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) under Contract DE-AC02-06CH11347.

†Argonne National Laboratory, Lemont, IL (jbrust@anl.gov, leyffer@mcs.anl.gov).

‡Lawrence Livermore National Laboratory, Livermore, CA (petra1@llnl.gov).

For large optimization problems limited memory versions of the compact representation in (1.3) are used. The limited memory versions typically store only the last $m > 0$ pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$ when $k \geq m$. In limited memory BFGS (L-BFGS) the dimensions of $\mathbf{S}_k$ and $\mathbf{Y}_k$ are consequently $n \times m$. Usually the memory parameter is much smaller than the problem size, namely, $m \ll n$. A typical range for this parameter is $5 \leq m \leq 50$ (see Boggs and Byrd in [2]). Moreover, in line search L-BFGS methods the initialization is frequently chosen as $\mathbf{B}_0 = \hat{\sigma}_k \mathbf{I}_n$, where $\hat{\sigma}_k = \mathbf{y}_{k-1}^T \mathbf{y}_{k-1} / \mathbf{s}_{k-1}^T \mathbf{y}_{k-1}$. Such an initialization enables efficient computations with the formula in (1.3), and adds extra information through the parameter $\hat{\sigma}_k$, which depends on the iteration k.

1.2. Structured Problems. When additional information about the structure of the objective function is known, it is desirable to include this information in a quasi-Newton update. Initial research efforts on structured quasi-Newton methods were in the context of nonlinear least squares problems. These include the work of Gill and Murray [12], Dennis et al. [8, 9], and Yabe and Takahashi [23]. Recently, Petra et al. [18] formulated the general structured minimization problem as

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x}), \qquad f(\mathbf{x}) = \widehat{k}(\mathbf{x}) + \widehat{u}(\mathbf{x}), \tag{1.4}$$

where $\widehat{k} : \mathbb{R}^n \to \mathbb{R}$ has known gradients and known Hessians and $\widehat{u} : \mathbb{R}^n \to \mathbb{R}$ has known gradients but unknown Hessians. For instance, objective functions composed of a general nonlinear function plus a regularizer or penalty term are described with (1.4). Thus, applications such as regularized logistic regressions [21] or optimal control problems contain structure that may be exploited, when we assume that the Hessian of the regularizer is know. Even though approximating the Hessian of the objective function in (1.4) by formula (1.2) or (1.3) is possible, this would not exploit the known parts of the Hessian. Therefore in [18] structured BFGS (S-BFGS) updates are derived, which combine known Hessian information with BFGS approximations for the unknown Hessian components. At each iteration the Hessian of the objective is approximated as $\nabla^2 f(\mathbf{x}_{k+1}) \approx \nabla^2 \widehat{k}(\mathbf{x}_{k+1}) + \mathbf{A}_{k+1}$, where $\mathbf{A}_{k+1}$ approximates the unknown Hessian, that is, $\mathbf{A}_{k+1} \approx \nabla^2 \widehat{u}(\mathbf{x}_{k+1})$. Given the known Hessian $\nabla^2 \widehat{k}(\mathbf{x}_{k+1}) \equiv \mathbf{K}_{k+1}$ and the gradients of $\widehat{u}$, let $\mathbf{u}_k \equiv \mathbf{K}_{k+1}\mathbf{s}_k + (\nabla \widehat{u}(\mathbf{x}_{k+1}) - \nabla \widehat{u}(\mathbf{x}_k))$. One of two structured approximations from [18] is the structured BFGS-Minus (S-BFGS-M) update

$$\mathbf{A}_{k+1}^{\mathrm{M}} = \mathbf{B}_k^{\mathrm{M}} - \mathbf{K}_{k+1} - \frac{1}{\mathbf{s}_k^T \mathbf{B}_k^{\mathrm{M}} \mathbf{s}_k} \mathbf{B}_k^{\mathrm{M}} \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k^{\mathrm{M}} + \frac{1}{\mathbf{s}_k^T \mathbf{u}_k} \mathbf{u}_k \mathbf{u}_k^T, \tag{1.5}$$

where $\mathbf{B}_k^{\mathrm{M}} = \mathbf{A}_k^{\mathrm{M}} + \mathbf{K}_k$. By adding $\mathbf{K}_{k+1}$ to both sides, the update from (1.5) implies a formula for $\mathbf{B}_{k+1}^{\mathrm{M}}$ that resembles (1.2), in which $\mathbf{B}_{k+1}$, $\mathbf{B}_k$, and $\mathbf{y}_k$ are replaced by $\mathbf{B}_{k+1}^{\mathrm{M}}$, $\mathbf{B}_k^{\mathrm{M}}$, and $\mathbf{u}_k$, respectively. Consequently, $\mathbf{B}_{k+1}^{\mathrm{M}}$ is symmetric positive definite given a symmetric positive definite initialization $\mathbf{B}_0^{\mathrm{M}}$ as long as $\mathbf{s}_k^T \mathbf{u}_k > 0$ for $k \geq 0$. A second formula is the structured BFGS-Plus (S-BFGS-P) update

$$\mathbf{A}_{k+1}^{\mathrm{P}} = \mathbf{A}_k^{\mathrm{P}} - \frac{1}{\mathbf{s}_k^T \widehat{\mathbf{B}}_k^{\mathrm{P}} \mathbf{s}_k} \widehat{\mathbf{B}}_k^{\mathrm{P}} \mathbf{s}_k \mathbf{s}_k^T \widehat{\mathbf{B}}_k^{\mathrm{P}} + \frac{1}{\mathbf{s}_k^T \mathbf{u}_k} \mathbf{u}_k \mathbf{u}_k^T, \tag{1.6}$$

where $\widehat{\mathbf{B}}_k^{\mathrm{P}} = \mathbf{A}_k^{\mathrm{P}} + \mathbf{K}_{k+1}$. Both of the updates in (1.5) and (1.6) were implemented in a line search algorithm and compared with the unstructured BFGS formula (1.2) in [18]. The structured updates obtained better results in terms of iteration count

and function evaluations than did the unstructured counterparts. Unlike the BFGS formula from (1.2), which recursively defines $\mathbf{B}_{k+1}$ as a rank-2 update to $\mathbf{B}_k$, the formulas for $\mathbf{A}^{\mathrm{M}}_{k+1}$ and $\mathbf{A}^{\mathrm{P}}_{k+1}$ in (1.5) and (1.6) additionally depend on the known Hessians $\mathbf{K}_{k+1}$ and $\mathbf{K}_k$. For this reason the compact representations of $\mathbf{A}^{\mathrm{M}}_{k+1}$ and $\mathbf{A}^{\mathrm{P}}_{k+1}$ are different from the one for $\mathbf{B}_{k+1}$ in (1.3) and have not yet been proposed.

1.3. Article Contributions. In this article we develop the compact representations of the structured BFGS updates $\mathbf{A}^{\mathrm{M}}_{k+1}$ and $\mathbf{A}^{\mathrm{P}}_{k+1}$ from (1.5) and (1.6). We propose the limited memory versions of the compact structured BFGS (L-S-BFGS) matrices and provide line search algorithms that implement them. We exploit the compact representations in order to compute search directions by means of the Sherman-Morrison-Woodbury formula and implement effective initialization strategies. Numerical experiments of the proposed limited memory structured BFGS methods on various problems are presented.

2. Structured BFGS Compact Representations. To develop the compact representations of the structured BFGS formulas, we define

$$\mathbf{U}_k = [\,\mathbf{u}_0, \quad \cdots, \quad \mathbf{u}_{k-1}\,], \quad \mathbf{S}_k^T\mathbf{U}_k = \mathbf{L}_k^{\mathrm{U}} + \mathbf{R}_k^{\mathrm{U}}, \quad \mathrm{diag}(\mathbf{S}_k^T\mathbf{U}_k) = \mathbf{D}_k^{\mathrm{U}}, \tag{2.1}$$

where $\mathbf{U}_k \in \mathbb{R}^{n\times k}$ collects all $\mathbf{u}_k$ for $k \geq 0$ and where $\mathbf{L}_k^{\mathrm{U}} \in \mathbb{R}^{k\times k}$ is a strictly lower triangular matrix, $\mathbf{R}_k^{\mathrm{U}} \in \mathbb{R}^{k\times k}$ is an upper triangular matrix (including the diagonal), and $\mathbf{D}_k^{\mathrm{U}} \in \mathbb{R}^{k\times k}$ is the diagonal part of $\mathbf{S}_k^T\mathbf{U}_k$.

2.1. Compact Representation of $\mathbf{A}_k^{\mathrm{M}}$. Theorem 2.1 contains the compact representation of $\mathbf{A}_k^{\mathrm{M}}$.

THEOREM 2.1. *The compact representation of $\mathbf{A}_k^M$ in the update formula (1.5) is*

$$\mathbf{A}_k^M = \mathbf{B}_0^M - \mathbf{K}_k - \begin{bmatrix}\mathbf{B}_0^M\mathbf{S}_k & \mathbf{U}_k\end{bmatrix}\begin{bmatrix}\mathbf{S}_k^T\mathbf{B}_0^M\mathbf{S}_k & \mathbf{L}_k^U \\ (\mathbf{L}_k^U)^T & -\mathbf{D}_k^U\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{S}_k^T(\mathbf{B}_0^M)^T \\ \mathbf{U}_k^T\end{bmatrix}, \tag{2.2}$$

where $\mathbf{S}_k$ is as defined in (1.3), $\mathbf{U}_k$, $\mathbf{L}_k^U$, and $\mathbf{D}_k^U$ are defined in (2.1), and

$$\mathbf{B}_0^M = \mathbf{A}_0^M + \mathbf{K}_0.$$

Proof. Observe that by adding $\mathbf{K}_{k+1}$ to both sides of (1.5) the update formula of $\mathbf{B}^{\mathrm{M}}_{k+1}$ becomes

$$\mathbf{B}^{\mathrm{M}}_{k+1} = \mathbf{B}^{\mathrm{M}}_k - \frac{1}{\mathbf{s}_k^T\mathbf{B}^{\mathrm{M}}_k\mathbf{s}_k}\mathbf{B}^{\mathrm{M}}_k\mathbf{s}_k\mathbf{s}_k^T\mathbf{B}^{\mathrm{M}}_k + \frac{1}{\mathbf{s}_k^T\mathbf{u}_k}\mathbf{u}_k\mathbf{u}_k^T.$$

This expression is the same as (1.2) when $\mathbf{B}^{\mathrm{M}}_{k+1}$ is relabeled as $\mathbf{B}_{k+1}$, $\mathbf{B}^{\mathrm{M}}_k$ is relabeled as $\mathbf{B}_k$, and $\mathbf{u}_k$ is relabeled as $\mathbf{y}_k$. The compact representation of (1.2) is given by (1.3), and therefore the compact representation of $\mathbf{B}^{\mathrm{M}}_k$ is given by (1.3) with $\mathbf{Y}_k$ replaced by $\mathbf{U}_k$ and $\mathbf{B}_0$ replaced by $\mathbf{B}^{\mathrm{M}}_0$. Then (2.2) is obtained by subtracting $\mathbf{K}_k$ from the compact representation of $\mathbf{B}^{\mathrm{M}}_k$, and noting that $\mathbf{B}^{\mathrm{M}}_0 = \mathbf{A}^{\mathrm{M}}_0 + \mathbf{K}_0$. Since $\mathbf{B}^{\mathrm{M}}_k$ is symmetric positive definite as long as $\mathbf{B}^{\mathrm{M}}_0$ is symmetric positive definite and $\mathbf{s}_k^T\mathbf{u}_k > 0$ for $k \geq 0$, the inverse in the right-hand side of (2.2) is nonsingular as long as $\mathbf{B}^{\mathrm{M}}_0$ is symmetric positive definite and $\mathbf{s}_k^T\mathbf{u}_k > 0$ for $k \geq 0$. ∎

Corollary 2.2 describes the compact representation of the inverse $\mathbf{H}^{\mathrm{M}}_k = (\mathbf{K}_k + \mathbf{A}^{\mathrm{M}}_k)^{-1}$, which is used to compute search directions in a line search algorithm (e.g., $\mathbf{p}^{\mathrm{M}}_k = -\mathbf{H}^{\mathrm{M}}_k\nabla f(\mathbf{x}_k)$).

COROLLARY 2.2. *The inverse* $\mathbf{H}_k^M = \left(\mathbf{K}_k + \mathbf{A}_k^M\right)^{-1}$, *with the compact representation of* $\mathbf{A}_k^M$ *from* (2.2), *is given as*

(2.3)
$$\mathbf{H}_k^M = \mathbf{H}_0^M - \begin{bmatrix}\mathbf{S}_k & \mathbf{H}_0^M\mathbf{U}_k\end{bmatrix}\begin{bmatrix}(\mathbf{T}_k^U)^T\left(\mathbf{D}_k^U + \mathbf{U}_k^T\mathbf{H}_0^M\mathbf{U}_k\right)\mathbf{T}_k^U & -(\mathbf{T}_k^U)^T \\ -\mathbf{T}_k^U & \mathbf{0}_{k\times k}\end{bmatrix}\begin{bmatrix}\mathbf{S}_k^T \\ \mathbf{U}_k^T(\mathbf{H}_0^M)^T\end{bmatrix},$$

where

$$\mathbf{H}_0^M = (\mathbf{B}_0^M)^{-1} = (\mathbf{A}_0^M + \mathbf{K}_0)^{-1}$$

and where $\mathbf{T}_k^U = (\mathbf{R}_k^U)^{-1}$ *with* $\mathbf{S}_k$, $\mathbf{U}_k$, $\mathbf{D}_k^U$, *and* $\mathbf{R}_k^U$ *defined in Theorem* 2.1 *and* (2.1).

Proof. Define

$$\boldsymbol{\Xi}_k \equiv \begin{bmatrix}\mathbf{B}_0^{\mathrm{M}}\mathbf{S}_k & \mathbf{U}_k\end{bmatrix}, \qquad \mathbf{M}_k \equiv \begin{bmatrix}\mathbf{S}_k^T\mathbf{B}_0^{\mathrm{M}}\mathbf{S}_k & \mathbf{L}_k^{\mathrm{U}} \\ (\mathbf{L}_k^{\mathrm{U}})^T & -\mathbf{D}_k^{\mathrm{U}}\end{bmatrix},$$

in the compact representation of $\mathbf{A}_k^{\mathrm{M}}$ in (2.2). Let $\mathbf{H}_0^{\mathrm{M}} = (\mathbf{B}_0^{\mathrm{M}})^{-1}$ then the expression of $\mathbf{H}_k^{\mathrm{M}}$ is obtained by the Sherman-Morrison-Woodbury identity:

$$\begin{aligned}
\mathbf{H}_k^{\mathrm{M}} &= \left(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{M}}\right)^{-1} \\
&= \left(\mathbf{B}_0^{\mathrm{M}} - \boldsymbol{\Xi}_k\mathbf{M}_k^{-1}\boldsymbol{\Xi}_k^T\right)^{-1} \\
&= (\mathbf{B}_0^{\mathrm{M}})^{-1} + (\mathbf{B}_0^{\mathrm{M}})^{-1}\boldsymbol{\Xi}_k\left[\mathbf{M}_k - \boldsymbol{\Xi}_k^T(\mathbf{B}_0^{\mathrm{M}})^{-1}\boldsymbol{\Xi}_k\right]^{-1}\boldsymbol{\Xi}_k^T(\mathbf{B}_0^{\mathrm{M}})^{-1} \\
&= \mathbf{H}_0^{\mathrm{M}} - \mathbf{H}_0^{\mathrm{M}}\boldsymbol{\Xi}_k\begin{bmatrix}\mathbf{0}_{k\times k} & \mathbf{R}_k^{\mathrm{U}} \\ (\mathbf{R}_k^{\mathrm{U}})^T & \mathbf{D}_k^{\mathrm{U}} + \mathbf{U}_k^T\mathbf{H}_0^{\mathrm{M}}\mathbf{U}_k\end{bmatrix}^{-1}\boldsymbol{\Xi}_k^T\mathbf{H}_0^{\mathrm{M}} \\
&= \mathbf{H}_0^{\mathrm{M}} - \mathbf{H}_0^{\mathrm{M}}\boldsymbol{\Xi}_k\begin{bmatrix}(\mathbf{R}_k^{\mathrm{U}})^{-T}\left(\mathbf{D}_k^{\mathrm{U}} + \mathbf{U}_k^T\mathbf{H}_0^{\mathrm{M}}\mathbf{U}_k\right)(\mathbf{R}_k^{\mathrm{U}})^{-1} & -(\mathbf{R}_k^{\mathrm{U}})^{-T} \\ -(\mathbf{R}_k^{\mathrm{U}})^{-1} & \mathbf{0}_{k\times k}\end{bmatrix}\boldsymbol{\Xi}_k^T\mathbf{H}_0^{\mathrm{M}}
\end{aligned}$$

where the third equality is obtained from applying the Sherman-Morrison-Woodbury inverse, the fourth equality uses the identity $\mathbf{S}_k^T\mathbf{U}_k - \mathbf{L}_k^{\mathrm{U}} = \mathbf{R}_k^{\mathrm{U}}$, and the fifth equality is obtained by explicitly computing the inverse of the block matrix. Using $(\mathbf{R}_k^{\mathrm{U}})^{-1} = \mathbf{T}_k^{\mathrm{U}}$ and $(\mathbf{B}_0^{\mathrm{M}})^{-1}\boldsymbol{\Xi}_k = (\mathbf{B}_0^{\mathrm{M}})^{-1}[\mathbf{B}_0^{\mathrm{M}}\mathbf{S}_k \quad \mathbf{U}_k]$ yields the expression in (2.3). ∎

2.2. Compact Representation of $\mathbf{A}_k^{\mathrm{P}}$. To develop the compact representation of the structured BFGS matrix $\mathbf{A}_k^{\mathrm{P}}$, we define $\mathbf{v}_k \equiv \mathbf{K}_{k+1}\mathbf{s}_k$ in addition to the expressions in (2.1) and

$$\mathbf{V}_k = \begin{bmatrix}\mathbf{v}_0, & \ldots, & \mathbf{v}_{k-1}\end{bmatrix}, \quad \mathbf{S}_k^T\mathbf{V}_k = \mathbf{L}_k^{\mathrm{V}} + \mathbf{R}_k^{\mathrm{V}}, \quad \mathrm{diag}(\mathbf{S}_k^T\mathbf{V}_k) = \mathbf{D}_k^{\mathrm{V}}, \tag{2.4}$$

where $\mathbf{V}_k \in \mathbb{R}^{n\times k}$ collects all $\mathbf{v}_k$ for $k \geq 0$ and where $\mathbf{L}_k^{\mathrm{V}} \in \mathbb{R}^{k\times k}$ is the strictly lower triangular matrix, $\mathbf{R}_k^{\mathrm{V}} \in \mathbb{R}^{k\times k}$ is the upper triangular matrix (including the diagonal), and $\mathbf{D}_k^{\mathrm{V}} \in \mathbb{R}^{k\times k}$ is the diagonal part of $\mathbf{S}_k^T\mathbf{V}_k$. Theorem 2.3 contains the compact representation of $\mathbf{A}_k^{\mathrm{P}}$.

THEOREM 2.3. *The compact representation of* $\mathbf{A}_k^P$ *in the update formula* (1.6) *is*

$$\mathbf{A}_k^P = \mathbf{A}_0^P - \begin{bmatrix}\mathbf{Q}_k & \mathbf{U}_k\end{bmatrix}\begin{bmatrix}\mathbf{D}_k^V + \mathbf{L}_k^V + (\mathbf{L}_k^V)^T + \mathbf{S}_k^T\mathbf{A}_0^P\mathbf{S}_k & \mathbf{L}_k^U \\ (\mathbf{L}_k^U)^T & -\mathbf{D}_k^U\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{Q}_k^T \\ \mathbf{U}_k^T\end{bmatrix}, \tag{2.5}$$

where

$$\mathbf{Q}_k \equiv \mathbf{V}_k + \mathbf{A}_0^P\mathbf{S}_k$$

and where $\mathbf{S}_k, \mathbf{U}_k, \mathbf{D}_k^U$, *and* $\mathbf{L}_k^U$ *are defined in Theorem 2.3 and* $\mathbf{V}_k$, $\mathbf{L}_k^V$, *and* $\mathbf{D}_k^V$ *are defined in (2.4).*

Proof. The proof of (2.5) is by induction. For $k = 1$ in (2.5) it holds that

$$\begin{aligned}\mathbf{A}_1^{\mathrm{P}} &= \mathbf{A}_0^{\mathrm{P}} - \begin{bmatrix}\mathbf{v}_0 + \mathbf{A}_0^{\mathrm{P}}\mathbf{s}_0 & \mathbf{u}_0\end{bmatrix}\begin{bmatrix}\mathbf{s}_0^T\mathbf{v}_0 + \mathbf{s}_0^T\mathbf{A}_0^{\mathrm{P}}\mathbf{s}_0 & \\ & -\mathbf{s}_0^T\mathbf{u}_0\end{bmatrix}^{-1}\begin{bmatrix}(\mathbf{v}_0 + \mathbf{A}_0^{\mathrm{P}}\mathbf{s}_0)^T \\ \mathbf{u}_0^T\end{bmatrix} \\ &= \mathbf{A}_0^{\mathrm{P}} - \frac{1}{\mathbf{s}_0^T(\mathbf{K}_1 + \mathbf{A}_0^{\mathrm{P}})\mathbf{s}_0}(\mathbf{K}_1 + \mathbf{A}_0^{\mathrm{P}})\mathbf{s}_0\mathbf{s}_0^T(\mathbf{K}_1 + \mathbf{A}_0^{\mathrm{P}})^T + \frac{1}{\mathbf{s}_0^T\mathbf{u}_0}\mathbf{u}_0\mathbf{u}_0^T.\end{aligned}$$

This expression is the same as $\mathbf{A}_1^{\mathrm{P}}$ in (1.6), and thus the compact representation holds for $k = 1$. Next assume that (2.5) is valid for $k \geq 1$, and in particular let it be represented as

$$\mathbf{A}_k^{\mathrm{P}} = \mathbf{A}_0^{\mathrm{P}} - \begin{bmatrix}\mathbf{Q}_k & \mathbf{U}_k\end{bmatrix}\begin{bmatrix}(\mathbf{M}_k)_{11} & (\mathbf{M}_k)_{12} \\ (\mathbf{M}_k)_{12}^T & (\mathbf{M}_k)_{22}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{Q}_k^T \\ \mathbf{U}_k^T\end{bmatrix}, \tag{2.6}$$

where

$$(\mathbf{M}_k)_{11} = \mathbf{D}_k^{\mathrm{V}} + \mathbf{L}_k^{\mathrm{V}} + (\mathbf{L}_k^{\mathrm{V}})^T + \mathbf{S}_k^T\mathbf{A}_0^{\mathrm{P}}\mathbf{S}_k, \quad (\mathbf{M}_k)_{12} = \mathbf{L}_k^{\mathrm{U}}, \quad (\mathbf{M}_k)_{22} = -\mathbf{D}_k^{\mathrm{U}}.$$

We verify the validity of (2.6) by substituting it in the update formula (1.6), and then seek the representation

$$\mathbf{A}_{k+1}^{\mathrm{P}} = \mathbf{A}_0^{\mathrm{P}} - \begin{bmatrix}\mathbf{Q}_{k+1} & \mathbf{U}_{k+1}\end{bmatrix}\begin{bmatrix}(\mathbf{M}_{k+1})_{11} & (\mathbf{M}_{k+1})_{12} \\ (\mathbf{M}_{k+1})_{12}^T & (\mathbf{M}_{k+1})_{22}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{Q}_{k+1}^T \\ \mathbf{U}_{k+1}^T\end{bmatrix}.$$

First let

$$\mathbf{q}_k = \mathbf{v}_k + \mathbf{A}_0^{\mathrm{P}}\mathbf{s}_k, \quad \mathbf{w}_k = \mathbf{Q}_k^T\mathbf{s}_k, \quad \mathbf{r}_k = \mathbf{U}_k^T\mathbf{s}_k, \quad \boldsymbol{\xi}_k = \begin{bmatrix}\mathbf{w}_k \\ \mathbf{r}_k\end{bmatrix},$$

and note that in (1.6) it holds that

$$\begin{aligned}(\mathbf{A}_k^{\mathrm{P}} + \mathbf{K}_{k+1})\mathbf{s}_k &= \mathbf{A}_k^{\mathrm{P}}\mathbf{s}_k + \mathbf{v}_k \\ &= \mathbf{A}_0^{\mathrm{P}}\mathbf{s}_k - \begin{bmatrix}\mathbf{Q}_k & \mathbf{U}_k\end{bmatrix}[\mathbf{M}_k]^{-1}\begin{bmatrix}\mathbf{Q}_k^T\mathbf{s}_k \\ \mathbf{U}_k^T\mathbf{s}_k\end{bmatrix} + \mathbf{v}_k \\ &\equiv \mathbf{q}_k - \begin{bmatrix}\mathbf{Q}_k & \mathbf{U}_k\end{bmatrix}[\mathbf{M}_k]^{-1}\begin{bmatrix}\mathbf{w}_k \\ \mathbf{r}_k\end{bmatrix} \\ &\equiv \mathbf{q}_k - \begin{bmatrix}\mathbf{Q}_k & \mathbf{U}_k\end{bmatrix}[\mathbf{M}_k]^{-1}\boldsymbol{\xi}_k.\end{aligned}$$

Next we define $\sigma_k^{\mathrm{P}} = 1/\mathbf{s}_k^T(\mathbf{A}_k^{\mathrm{P}} + \mathbf{K}_{k+1})\mathbf{s}_k$ and obtain the following representation of

$\mathbf{A}^{\mathrm{P}}_{k+1}$:

$$\begin{aligned}
\mathbf{A}^{\mathrm{P}}_{k+1} &= \mathbf{A}^{\mathrm{P}}_{k} - \sigma^{\mathrm{P}}_{k}(\mathbf{A}^{\mathrm{P}}_{k}\mathbf{s}_k + \mathbf{v}_k)(\mathbf{A}^{\mathrm{P}}_{k}\mathbf{s}_k + \mathbf{v}_k)^T + \frac{1}{\mathbf{s}_k^T\mathbf{u}_k}\mathbf{u}_k\mathbf{u}_k^T \\
&= \mathbf{A}^{\mathrm{P}}_{0} - \sigma^{\mathrm{P}}_{k}\begin{bmatrix} \mathbf{Q}_k & \mathbf{U}_k & \mathbf{q}_k \end{bmatrix}\begin{bmatrix} \frac{\mathbf{M}_k^{-1}}{\sigma^{\mathrm{P}}_{k}} + \mathbf{M}_k^{-1}\boldsymbol{\xi}_k\boldsymbol{\xi}_k^T\mathbf{M}_k^{-1} & -\mathbf{M}_k^{-1}\boldsymbol{\xi}_k \\ -\boldsymbol{\xi}_k^T\mathbf{M}_k^{-1} & 1 \end{bmatrix}\begin{bmatrix} \mathbf{Q}_k^T \\ \mathbf{U}_k^T \\ \mathbf{q}_k^T \end{bmatrix} \\
&\quad + \frac{1}{\mathbf{s}_k^T\mathbf{u}_k}\mathbf{u}_k\mathbf{u}_k^T \\
&= \mathbf{A}^{\mathrm{P}}_{0} - \begin{bmatrix} \mathbf{Q}_k & \mathbf{U}_k & \mathbf{q}_k \end{bmatrix}\begin{bmatrix} \mathbf{M}_k & \begin{bmatrix} \mathbf{w}_k \\ \mathbf{r}_k \end{bmatrix} \\ \begin{bmatrix} \mathbf{w}_k^T & \mathbf{r}_k^T \end{bmatrix} & \mathbf{s}_k^T\mathbf{q}_k \end{bmatrix}^{-1}\begin{bmatrix} \mathbf{Q}_k^T \\ \mathbf{U}_k^T \\ \mathbf{q}_k^T \end{bmatrix} + \frac{1}{\mathbf{s}_k^T\mathbf{u}_k}\mathbf{u}_k\mathbf{u}_k^T.
\end{aligned}$$

Using the permutation matrix $\mathbf{P} = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_k & \mathbf{e}_{2k+1} & \cdots & \mathbf{e}_{2k} \end{bmatrix}$, we represent $\mathbf{A}^{\mathrm{P}}_{k+1}$ as

$$\begin{aligned}
\mathbf{A}^{\mathrm{P}}_{k+1} &= \mathbf{A}^{\mathrm{P}}_{0} - \begin{bmatrix} \mathbf{Q}_k & \mathbf{U}_k & \mathbf{q}_k \end{bmatrix}\mathbf{P}\mathbf{P}^T\begin{bmatrix} \mathbf{M}_k & \begin{bmatrix} \mathbf{w}_k \\ \mathbf{r}_k \end{bmatrix} \\ \begin{bmatrix} \mathbf{w}_k^T & \mathbf{r}_k^T \end{bmatrix} & \mathbf{s}_k^T\mathbf{q}_k \end{bmatrix}^{-1}\mathbf{P}\mathbf{P}^T\begin{bmatrix} \mathbf{Q}_k^T \\ \mathbf{U}_k^T \\ \mathbf{q}_k^T \end{bmatrix} \\
&\quad + \frac{1}{\mathbf{s}_k^T\mathbf{u}_k}\mathbf{u}_k\mathbf{u}_k^T \\
&= \mathbf{A}^{\mathrm{P}}_{0} - \begin{bmatrix} \mathbf{Q}_k & \mathbf{q}_k & \mathbf{U}_k & \mathbf{u}_k \end{bmatrix}\begin{bmatrix} (\mathbf{M}_k)_{11} & \mathbf{w}_k & (\mathbf{M}_k)_{12} & \mathbf{0} \\ \mathbf{w}_k^T & \mathbf{s}_k^T\mathbf{q}_k & \mathbf{r}_k^T & 0 \\ (\mathbf{M}_k)_{12}^T & \mathbf{r}_k & (\mathbf{M}_k)_{22} & \mathbf{0} \\ \mathbf{0} & 0 & \mathbf{0} & -\mathbf{s}_k^T\mathbf{u}_k \end{bmatrix}^{-1}\begin{bmatrix} \mathbf{Q}_k^T \\ \mathbf{q}_k^T \\ \mathbf{U}_k^T \\ \mathbf{u}_k^T \end{bmatrix}.
\end{aligned}$$

Now we verify that the identities hold:

$$\begin{aligned}
\mathbf{Q}_{k+1} &= \begin{bmatrix} \mathbf{Q}_k & \mathbf{q}_k \end{bmatrix} = \begin{bmatrix} \mathbf{V}_k + \mathbf{A}^{\mathrm{P}}_{0}\mathbf{S}_k & \mathbf{v}_k + \mathbf{A}^{\mathrm{P}}_{0}\mathbf{s}_k \end{bmatrix} = \mathbf{V}_{k+1} + \mathbf{A}^{\mathrm{P}}_{0}\mathbf{S}_{k+1}, \\
\mathbf{U}_{k+1} &= \begin{bmatrix} \mathbf{U}_k & \mathbf{u}_k \end{bmatrix}, \\
(\mathbf{M}_{k+1})_{11} &= \begin{bmatrix} (\mathbf{M}_k)_{11} & \mathbf{w}_k \\ \mathbf{w}_k^T & \mathbf{s}_k^T\mathbf{q}_k \end{bmatrix} = \mathbf{D}^{\mathrm{V}}_{k+1} + \mathbf{L}^{\mathrm{V}}_{k+1} + (\mathbf{L}^{\mathrm{V}}_{k+1})^T + \mathbf{S}_{k+1}^T\mathbf{A}^{\mathrm{P}}_{0}\mathbf{S}_{k+1}, \\
(\mathbf{M}_{k+1})_{12} &= \begin{bmatrix} (\mathbf{M}_k)_{12} & \mathbf{0} \\ \mathbf{r}_k^T & 0 \end{bmatrix} = \mathbf{L}_{k+1}, \\
(\mathbf{M}_{k+1})_{22} &= \begin{bmatrix} (\mathbf{M}_k)_{22} & \mathbf{0} \\ \mathbf{0} & -\mathbf{s}_k^T\mathbf{u}_k \end{bmatrix} = -\mathbf{D}_{k+1}.
\end{aligned}$$

Therefore we conclude that $\mathbf{A}^{\mathrm{P}}_{k+1}$ is of the form (2.5) with $k+1$ replacing the indices k. ∎

2.3. Limited Memory Compact Structured BFGS. The limited memory representations of (2.2) and (2.5) are obtained by storing only the last $m \geq 1$ columns of $\mathbf{S}_k, \mathbf{U}_k$ and $\mathbf{V}_k$. By setting $m \ll n$ limited memory strategies enable computational efficiencies and lower storage requirements, see e.g., [17]. Updating $\mathbf{S}_k, \mathbf{U}_k$ and $\mathbf{V}_k$ 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, $\underline{\mathbf{S}}_k$ represents $\mathbf{S}_k$ without its first column. With this notation, a column update of a matrix, say

$\mathbf{S}_k$, by a vector $\mathbf{s}_k$ is defined as follows.

$$\text{colUpdate}\,(\mathbf{S}_k, \mathbf{s}_k) \equiv \begin{cases} [\,\mathbf{S}_k\ \mathbf{s}_k\,] & \text{if } k < m \\ [\,\underline{\mathbf{S}}_k\ \mathbf{s}_k\,] & \text{if } k \geq m. \end{cases}$$

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 to, for instance, obtain $\mathbf{S}_{k+1}$ from $\mathbf{S}_k$ and $\mathbf{s}_k$, i.e., $\mathbf{S}_{k+1} = \text{colUpdate}(\mathbf{S}_k, \mathbf{s}_k)$. Next, let an overline above a matrix represent the matrix with its first row removed. That is, $\overline{\mathbf{S}_k^T\mathbf{U}}_k$ represents $\mathbf{S}_k^T\mathbf{U}_k$ without its first row. With this notation, a product update of, say $\mathbf{S}_k^T\mathbf{U}_k$, by matrices $\mathbf{S}_k$, $\mathbf{U}_k$ and vectors $\mathbf{s}_k$, $\mathbf{u}_k$ is defined as:

$$\text{prodUpdate}\left(\mathbf{S}_k^T\mathbf{U}_k, \mathbf{S}_k, \mathbf{U}_k, \mathbf{s}_k, \mathbf{u}_k\right) \equiv \begin{cases} \begin{bmatrix} \mathbf{S}_k^T\mathbf{U}_k & \mathbf{S}_k^T\mathbf{u}_k \\ \mathbf{s}_k^T\mathbf{U}_k & \mathbf{s}_k^T\mathbf{u}_k \end{bmatrix} & \text{if } k < m \\ \begin{bmatrix} \left(\overline{\mathbf{S}_k^T\mathbf{U}_k}\right) & \underline{\mathbf{S}}_k^T\mathbf{u}_k \\ \mathbf{s}_k^T\underline{\mathbf{U}}_k & \mathbf{s}_k^T\mathbf{u}_k \end{bmatrix} & \text{if } k \geq m. \end{cases}$$

This product update is used to compute matrix products, such as, $\mathbf{S}_{k+1}^T\mathbf{U}_{k+1}$, with $\mathcal{O}(2mn)$ multiplications, instead of $\mathcal{O}(m^2n)$ when the product $\mathbf{S}_k^T\mathbf{U}_k$ had previously been stored. Moreover, we let "diag($\mathbf{S}_k^T\mathbf{U}_k$)" extract the diagonal elements of a matrix, say $\mathbf{S}_k^T\mathbf{U}_k$, while "tril($\mathbf{S}_k^T\mathbf{U}_k, -1$)" are the strictly lower triangular elements (elements below, excluding the main diagonal) and "triu($\mathbf{S}_k^T\mathbf{U}_k, 0$)" the upper triangular elements (elements above, including the main diagonal). Section 3 discusses computational and memory aspects in greater detail.

3. Algorithms. This section describes two line search algorithms with limited memory structured BFGS matrices. The compact representations enable efficient reinitialization strategies and search directions, and we discuss these two components first, before presenting the overall algorithms.

3.1. Initializations. For the limited memory BFGS matrix based on (1.3) one commonly uses the initializations $\mathbf{B}_0^{(k)} = \widehat{\sigma}_k\mathbf{I}_n$, where $\widehat{\sigma}_k = \mathbf{y}_{k-1}^T\mathbf{y}_{k-1}/\mathbf{s}_{k-1}^T\mathbf{y}_{k-1}$ (c.f. [5]). Choosing the initialization as a multiple of the identity matrix enables fast computations with the matrix in (1.3). In particular, the inverse of this matrix may be computed efficiently by the Sherman-Morrison-Woodbury identity. Because at the outset it is not necessarily obvious which initializations to use for the limited memory structured-BFGS (L-S-BFGS) matrices based on (2.2) and (2.5), we investigate different approaches. We are motivated by the analysis in [1], which proposed formula $\widehat{\sigma}_k$. Additionally, in that work a second initialization $\widehat{\sigma}_k^{(2)} = \mathbf{s}_{k-1}^T\mathbf{y}_{k-1}/\mathbf{s}_{k-1}^T\mathbf{s}_{k-1}$ was proposed. Because in the S-BFGS methods the vectors $\widehat{\mathbf{u}}_k$ and $\mathbf{u}_k$ are used instead of $\mathbf{y}_k$ (unstructured BFGS), the initializations in this article are the below.

$$\sigma_{k+1} = \begin{cases} \frac{\mathbf{u}_k^T\mathbf{u}_k}{\mathbf{s}_k^T\mathbf{u}_k} & \text{Init. 1} \\ \frac{\widehat{\mathbf{u}}_k^T\widehat{\mathbf{u}}_k}{\mathbf{s}_k^T\widehat{\mathbf{u}}_k} & \text{Init. 2} \\ \frac{\mathbf{s}_k^T\mathbf{u}_k}{\mathbf{s}_k^T\mathbf{s}_k} & \text{Init. 3} \\ \frac{\mathbf{s}_k^T\widehat{\mathbf{u}}_k}{\mathbf{s}_k^T\mathbf{s}_k} & \text{Init. 4} \end{cases} \tag{3.1}$$

Note that Init. 1 and Init. 2 are extensions of $\widehat{\sigma}_k$ to structured methods. Instead of using $\mathbf{y}_k$ these initializations are defined by $\widehat{\mathbf{u}}_k$ and $\mathbf{u}_k$. Init. 3 and Init. 4

extend $\widehat{\sigma}_k^{(2)}$. Observe that the vectors $\widehat{\mathbf{u}}_k = \nabla\widehat{u}(\mathbf{x}_{k+1}) - \nabla\widehat{u}(\mathbf{x}_k)$ depend only on gradient information of $\widehat{u}(\mathbf{x})$. In contrast, $\mathbf{u}_k = \mathbf{K}_{k+1}\mathbf{s}_k + \widehat{\mathbf{u}}_k$ depends on known second-derivative information, too. Because the initial matrices $\mathbf{A}_0^{\mathrm{M}}$ and $\mathbf{A}_0^{\mathrm{P}}$ affect the compact representations from Theorems 2.1 and 2.3 differently, we accordingly adjust our initialization strategies for these two matrices. In particular, for L-S-BFGS-M the compact limited memory formula for $\mathbf{B}_k^{\mathrm{M}}$ simplifies if we take $\mathbf{B}_0^{\mathrm{M}}$ as a multiple of the identity matrix:

$$\mathbf{B}_0^{\mathrm{M}} = \mathbf{A}_0^{\mathrm{M}} + \mathbf{K}_0 \equiv \sigma_k \mathbf{I}. \tag{3.2}$$

The advantage of this choice is that it enables computational complexities similar to those of the L-BFGS formula from (1.3). However by setting this default initialization for $\mathbf{B}_0^{\mathrm{M}}$ the corresponding limited memory matrices $\mathbf{B}_k^{\mathrm{M}}$ are not equivalent anymore to the full-memory matrices $\mathbf{B}_k^{\mathrm{M}}$ defined by (1.5), even when $k < m$. In Section 3.4.1 computational techniques are discussed when $\mathbf{B}_0^{\mathrm{M}}$ is not taken as a multiple of the identity matrix. For L-S-BFGS-P we set $\mathbf{A}_0^{\mathrm{P}} = \sigma_k \mathbf{I}$. This initialization, as long as σ_k remains constant, implies that the limited memory compact representations from Theorem 2.1 and the update formulas from (1.6) produce the same matrices when $k < m$.

3.2. Search Directions. The search directions for line search algorithms, with the structured BFGS approximations, are computed as

$$\mathbf{p}_k = -(\mathbf{K}_k + \mathbf{A}_k)^{-1}\mathbf{g}_k, \tag{3.3}$$

where $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ and where $\mathbf{A}_k$ is either the limited memory version of $\mathbf{A}_k^{\mathrm{M}}$ from (2.2) or $\mathbf{A}_k^{\mathrm{P}}$ from (2.5). When $\mathbf{A}_k^{\mathrm{M}}$ is used, we apply the expression of the inverse from Corollary 2.2, in order to compute search directions. In particular, with the initialization strategy $\mathbf{B}_0^{\mathrm{M}} = \sigma_k \mathbf{I}$ from the preceding section, the search directions (3.3) are computed efficiently by

$$\mathbf{p}_k^{\mathrm{M}} = -\frac{\mathbf{g}_k}{\sigma_k} + [\mathbf{S}_k \quad \mathbf{U}_k] \begin{bmatrix} (\mathbf{T}_k^{\mathrm{U}})^T \left(\mathbf{D}_k^{\mathrm{U}} + 1/\sigma_k \mathbf{U}_k^T \mathbf{U}_k\right) \mathbf{T}_k^{\mathrm{U}} & -1/\sigma_k (\mathbf{T}_k^{\mathrm{U}})^T \\ -1/\sigma_k \mathbf{T}_k^{\mathrm{U}} & \mathbf{0}_{k\times k} \end{bmatrix} \left(\begin{bmatrix} \mathbf{S}_k^T \\ \mathbf{U}_k^T \end{bmatrix} \mathbf{g}_k \right), \tag{3.4}$$

where $\mathbf{T}_k^{\mathrm{U}}$ is defined in Corollary 2.2. This computation is done efficiently assuming that all matrices have been updated before, such as $\mathbf{U}_k^T\mathbf{U}_k$. Omitting terms of order m, the multiplication complexity for this search direction is $\mathcal{O}(n(4m+1)+3m^2)$. In particular, computing $\mathbf{p}_k^{\mathrm{M}}$ can be done by: two vector multiplies with the $n \times 2m$ matrix $[\,\mathbf{S}_k\,\mathbf{U}_k\,]$ (order $4nm$), the scaling $\frac{\mathbf{g}_k}{\sigma_k}$ (order n) and a matrix vector product with a structured $2m \times 2m$ matrix. Since $\mathbf{T}_k^{\mathrm{U}}$ represents a solve with an $m \times m$ upper triangular matrix the vector product with the middle $2m \times 2m$ matrix is done in order $3m^2$. When $\mathbf{A}_k^{\mathrm{P}}$ is used, search directions are computed by solves of the linear system $(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}})\mathbf{p}_k^{\mathrm{P}} = -\mathbf{g}_k$.

3.3. Algorithms. As in the work of Petra et al. [18], the compact representations of the structured BFGS formulas are implemented in a strong Wolfe line search algorithm based on [16]. For nonnegative constants $0 < c_1 \leq c_2$, the current iterate $\mathbf{x}_k$ and search direction $\mathbf{p}_k$, the strong Wolfe conditions define the step length parameter α by two inequalities

$$f(\mathbf{x}_k + \alpha\mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1\alpha(\mathbf{p}_k^T\nabla f(\mathbf{x}_k)),\ \left|\mathbf{p}_k^T\nabla f(\mathbf{x}_k + \alpha\mathbf{p}_k)\right| \leq c_2\left|\mathbf{p}_k^T\nabla f(\mathbf{x}_k)\right|. \tag{3.5}$$

Because the S-BFGS-M matrix from (2.2) is positive definite as long as $\mathbf{s}_k^T\mathbf{u}_k > 0$ for $k \geq 0$, the line searches in our algorithms include this condition. Moreover, when S-BFGS-M is used, new search directions are computed by using the inverse from Corollary 2.2. In contrast, because the S-BFGS-P matrix from (2.5) is not necessarily positive definite even if $\mathbf{s}_k^T\mathbf{u}_k > 0$ for $k \geq 0$ (see [18]), our implementation checks whether $\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}}$ is positive definite, before computing a new search direction. If this matrix is positive definite, then a new search direction is computed by solving the linear system $(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}})\mathbf{p}_k^{\mathrm{P}} = -\mathbf{g}_k$. Otherwise the search direction is computed by solving the system $(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}} + \delta\mathbf{I}_n)\mathbf{p}_k^{\mathrm{P}} = -\mathbf{g}_k$, where the scalar $\delta > 0$ ensures that $(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}} + \delta\mathbf{I}_n) \succ 0$ (Here δ is chosen as the the first $\delta = 10^j, j = 0, 1, \ldots$ that yields a positive definite matrix). The proposed limited memory line search algorithms are listed in Algorithms 3.1 and 3.2.

Algorithm 3.1 Limited Memory Structured-BFGS-Minus (L-S-BFGS-M)

1: Initialize: $k = 0$, $m > 0$, $\epsilon > 0$, $\sigma_k > 0$, $0 < c_1 \leq c_2$, $\mathbf{x}_k$, $\mathbf{g}_k = \nabla f(\mathbf{x}_k) = \nabla\hat{k}(\mathbf{x}_k) + \nabla\hat{u}(\mathbf{x}_k)$, $\mathbf{S}_k = 0, \mathbf{U}_k = 0, \mathbf{D}_k^{\mathrm{U}} = 0, (\mathbf{R}_k^{\mathrm{U}})^{-1} = 0$, $\mathbf{S}_k^T\mathbf{U}_k = 0$, $\mathbf{U}_k^T\mathbf{U}_k = 0$, $\mathbf{H}_0 = (1/\sigma_k)\mathbf{I}$, $\mathbf{\Theta}_k = [\mathbf{S}_k \quad \mathbf{H}_0\mathbf{U}_k]$
2: **while** $\|\mathbf{g}_k\|_\infty > \epsilon$ **do**
3: Compute:

$$\mathbf{p}_k = -\mathbf{H}_0\mathbf{g}_k + \mathbf{\Theta}_k\mathbf{M}_k(\mathbf{\Theta}_k^T\mathbf{g}_k),$$

where

$$\mathbf{M}_k = \begin{bmatrix} (\mathbf{R}_k^{\mathrm{U}})^{-T}\left(\mathbf{D}_k^{\mathrm{U}} + \mathbf{U}_k^T\mathbf{H}_0\mathbf{U}_k\right)(\mathbf{R}_k^{\mathrm{U}})^{-1} & -(\mathbf{R}_k^{\mathrm{U}})^{-T} \\ -(\mathbf{R}_k^{\mathrm{U}})^{-1} & \mathbf{0} \end{bmatrix}.$$

4: Strong Wolfe line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{p}_k,$$

where $\alpha > 0$, $\mathbf{x}_{k+1}$ satisfies strong Wolfe conditions (cf. [18] and (3.5)), $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{s}_k^T\mathbf{u}_k > 0$.
5: Updates: $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$, $\mathbf{u}_k = \nabla^2\hat{k}(\mathbf{x}_{k+1})\mathbf{s}_k + (\nabla\hat{u}(\mathbf{x}_{k+1}) - \nabla\hat{u}(\mathbf{x}_k))$
6: $\mathbf{S}_{k+1} = \mathrm{colUpdate}(\mathbf{S}_k, \mathbf{s}_k)$
7: $\mathbf{U}_{k+1} = \mathrm{colUpdate}(\mathbf{U}_k, \mathbf{u}_k)$
8: $\mathbf{S}_{k+1}^T\mathbf{U}_{k+1} = \mathrm{prodUpdate}(\mathbf{S}_k^T\mathbf{U}_k, \mathbf{S}_k, \mathbf{U}_k, \mathbf{s}_k, \mathbf{u}_k)$
9: $\mathbf{U}_{k+1}^T\mathbf{U}_{k+1} = \mathrm{prodUpdate}(\mathbf{U}_k^T\mathbf{U}_k, \mathbf{U}_k, \mathbf{U}_k, \mathbf{u}_k, \mathbf{u}_k)$
10: $\mathbf{D}_{k+1}^{\mathrm{U}} = \mathrm{diag}(\mathbf{S}_{k+1}^T\mathbf{U}_{k+1})$
11: $\mathbf{R}_{k+1}^{\mathrm{U}} = \mathrm{triu}(\mathbf{S}_{k+1}^T\mathbf{U}_{k+1}, 0)$
12: Compute: σ_{k+1}
13: $\mathbf{H}_0 = (1/\sigma_{k+1})\mathbf{I}$, update $\mathbf{M}_{k+1}$, $\mathbf{\Theta}_{k+1}$ using Theorem 2.1, $k = k + 1$
14: **end while**
15: **return** $\mathbf{x}_k$

Note that $\mathbf{\Theta}_k^T\mathbf{g}_k$ on Line 3 in Algorithm 3.1 is computed as $\begin{bmatrix} \mathbf{S}_k^T\mathbf{g}_k \\ \mathbf{H}_0(\mathbf{U}_k^T\mathbf{g}_k) \end{bmatrix}$ so that only one linear solve with $\mathbf{H}_0 = (\mathbf{K}_0 + \mathbf{A}_0^M)^{-1}$ is computed, when the algorithm does not use a multiple of the identity as the initialization.

Algorithm 3.2 is expected to be computationally more expensive than Algorithm 3.1 because it tests for the positive definiteness of $\mathbf{K}_k + \mathbf{A}_k$ in Line 3 and it computes search directions by the solve in Line 6. However, the structured quasi-Newton approximation in Algorithm 3.2 may be a more accurate approximation of

Algorithm 3.2 Limited Memory Structured-BFGS-Plus (L-S-BFGS-P)

Initialize: $k = 0$, $m > 0$, $\epsilon > 0$, $\sigma_k > 0$, $0 < c_1 \leq c_2$, $\mathbf{x}_k$, $\mathbf{g}_k = \nabla f(\mathbf{x}_k) = \nabla \hat{k}(\mathbf{x}_k) + \nabla \hat{u}(\mathbf{x}_k)$, $\mathbf{K}_k = \nabla^2 \hat{k}(\mathbf{x}_k)$, $\mathbf{S}_k = 0, \mathbf{U}_k = 0, \mathbf{V}_k = 0, \mathbf{D}_k^{\mathrm{U}} = 0, \mathbf{L}_k^{\mathrm{U}} = 0, \mathbf{D}_k^{\mathrm{V}} = 0, \mathbf{L}_k^{\mathrm{V}} = 0$, $\mathbf{\Omega}_k = 0$, $\mathbf{S}_k^T \mathbf{U}_k = 0$, $\mathbf{S}_k^T \mathbf{V}_k = 0$, $\mathbf{S}_k^T \mathbf{S}_k = 0$, $\mathbf{A}_k = \sigma_k \mathbf{I}$
while $\|\mathbf{g}_k\|_\infty > \epsilon$ **do**
  if $(\mathbf{K}_k + \mathbf{A}_k) \not\succ 0$ **then**
    Find $\delta > 0$ such that $(\mathbf{K}_k + \mathbf{A}_k + \delta \mathbf{I}_n) \succ 0$
  end if
  Solve:

$$(\mathbf{K}_k + \mathbf{A}_k)\mathbf{p}_k = -\mathbf{g}_k$$

  Strong Wolfe line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_k,$$

  where $\alpha > 0$, $\mathbf{x}_{k+1}$ satisfies strong Wolfe conditions (cf. [18] and (3.5)), $\mathbf{s}_k = \mathbf{x}_{k+1} = \mathbf{x}_k$.
  Updates: $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$, $\mathbf{K}_{k+1} = \nabla^2 \hat{k}(\mathbf{x}_{k+1})$, $\mathbf{v}_k = \mathbf{K}_{k+1}\mathbf{s}_k$, $\mathbf{u}_k = \mathbf{v}_k + (\nabla \hat{u}(\mathbf{x}_{k+1}) - \nabla \hat{u}(\mathbf{x}_k))$
  $\mathbf{S}_{k+1} = \mathrm{colUpdate}(\mathbf{S}_k, \mathbf{s}_k)$
  $\mathbf{U}_{k+1} = \mathrm{colUpdate}(\mathbf{U}_k, \mathbf{u}_k)$
  $\mathbf{V}_{k+1} = \mathrm{colUpdate}(\mathbf{V}_k, \mathbf{v}_k)$
  $\mathbf{S}_{k+1}^T \mathbf{U}_{k+1} = \mathrm{prodUpdate}(\mathbf{S}_k^T \mathbf{U}_k, \mathbf{S}_k, \mathbf{U}_k, \mathbf{s}_k, \mathbf{u}_k)$
  $\mathbf{S}_{k+1}^T \mathbf{V}_{k+1} = \mathrm{prodUpdate}(\mathbf{S}_k^T \mathbf{V}_k, \mathbf{S}_k, \mathbf{V}_k, \mathbf{s}_k, \mathbf{v}_k)$
  $\mathbf{S}_{k+1}^T \mathbf{S}_{k+1} = \mathrm{prodUpdate}(\mathbf{S}_k^T \mathbf{S}_k, \mathbf{S}_k, \mathbf{S}_k, \mathbf{s}_k, \mathbf{s}_k)$
  $\mathbf{L}_{k+1}^{\mathrm{U}} = \mathrm{tril}(\mathbf{S}_{k+1}^T \mathbf{U}_{k+1}, -1)$
  $\mathbf{L}_{k+1}^{\mathrm{V}} = \mathrm{tril}(\mathbf{S}_{k+1}^T \mathbf{V}_{k+1}, -1)$
  $\mathbf{D}_{k+1}^{\mathrm{U}} = \mathrm{diag}(\mathbf{S}_{k+1}^T \mathbf{U}_{k+1})$
  $\mathbf{D}_{k+1}^{\mathrm{V}} = \mathrm{diag}(\mathbf{S}_{k+1}^T \mathbf{U}_{k+1})$
  Compute: σ_{k+1}
  $\mathbf{A}_0 = (1/\sigma_{k+1})\mathbf{I}$, update $\mathbf{\Omega}_{k+1} = [\, \mathbf{V}_{k+1} + \mathbf{A}_0 \mathbf{S}_{k+1} \; \mathbf{U}_k \,]$

$$\mathbf{A}_{k+1} = \mathbf{A}_0 - \mathbf{\Omega}_{k+1} \begin{bmatrix} \mathbf{D}_{k+1}^{\mathrm{V}} + \mathbf{L}_{k+1}^{\mathrm{V}} + (\mathbf{L}_{k+1}^{\mathrm{V}})^T + \mathbf{S}_{k+1}^T \mathbf{A}_0 \mathbf{S}_{k+1} & \mathbf{L}_{k+1}^{\mathrm{U}} \\ (\mathbf{L}_{k+1}^{\mathrm{U}})^T & -\mathbf{D}_{k+1}^{\mathrm{U}} \end{bmatrix}^{-1} \mathbf{\Omega}_{k+1}^T$$

  $k = k + 1$
end while
return $\mathbf{x}_k$

the true Hessian (see [18]), which may result in fewer iterations or better convergence properties. Unlike Algorithm 3.2, Algorithm 3.1 does not require solves with large linear systems.

3.4. Large Scale Computation Considerations. This section discusses computational complexity and memory requirements of the structured Hessian approximations when the problems become large. In particular, if n is very large the Hessian matrices $\mathbf{K}_k$ typically exhibit additional structure, such as being diagonal or sparse. When $\mathbf{K}_k$ is sparse and solves with it can be done efficiently, the compact represen-

tation of $\mathbf{A}_k^{\mathrm{M}}$ and $\mathbf{A}_k^{\mathrm{P}}$ can be exploited to compute inverses of $\mathbf{K}_k + \mathbf{A}_k$ efficiently. This is because the matrices $\mathbf{K}_k + \mathbf{A}_k$, (with limited memory $\mathbf{A}_k$ from Theorem 2.1 or Theorem 2.3, respectively), have the form with $m \ll n$:

$$\mathbf{K}_k + \mathbf{A}_k \equiv \widehat{\mathbf{K}}_0 - \begin{bmatrix} & \\ \boldsymbol{\Xi}_k & \\ & \end{bmatrix} \big[\mathbf{M}_k\big]^{-1} \big[\text{———} \quad \boldsymbol{\Xi}_k^T \quad \text{———}\big]. \tag{3.6}$$

If $\mathbf{A}_k^{\mathrm{M}}$ is used in (3.6) then $\widehat{\mathbf{K}}_0 = \mathbf{K}_0 + \mathbf{A}_0^{\mathrm{M}}$ and $\boldsymbol{\Xi}_k$, $\mathbf{M}_k$ correspond to the remaining terms in Theorem 2.1. Using $\mathbf{A}_k^{\mathrm{P}}$ in (3.6) then $\widehat{\mathbf{K}}_0 = \mathbf{K}_k + \mathbf{A}_0^{\mathrm{P}}$ and $\boldsymbol{\Xi}_k$, $\mathbf{M}_k$ correspond to the remaining terms in Theorem 2.3. Because of its structure the matrix in (3.6) can be inverted efficiently by the Sherman-Morrison-Woodbury formula as long as solves with $\widehat{\mathbf{K}}_0$ can be done efficiently. Next, L-S-BFGS-M and L-S-BFGS-P are discussed in the situation when solves with $\widehat{\mathbf{K}}_0$ are done efficiently. Afterwards we relate these methods to S-BFGS-M, S-BFGS-P and BFGS, L-BFGS.

3.4.1. Computations for L-S-BFGS-M. The most efficient computations are achieved when $\widehat{\mathbf{K}}_0$ is set as a multiple of the identity matrix $\sigma_k \mathbf{I}$ (cf. Subsection 3.2 with $\mathcal{O}(n(4m+1)+3m^2)$ multiplications). This approach however omits the $\mathbf{K}_0$ term. Nevertheless, when $\mathbf{K}_0$ has additional structure such that factorizations and solves with it can be done in, say nl multiplications, search directions can be computed efficiently in this case, without omitting $\mathbf{K}_0$. In particular, the search direction is computed as $\mathbf{p}_k^{\mathrm{M}} = -(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{M}})^{-1}\mathbf{g}_k = -\mathbf{H}_k^{\mathrm{M}}\mathbf{g}_k$ where $\mathbf{H}_k^{\mathrm{M}}$ is the inverse from Corollary 2.2. The initialization matrix is $\mathbf{H}_0^{\mathrm{M}} = (\sigma_k \mathbf{I} + \mathbf{K}_0)^{-1}$. To determine the search direction two matrix vector products with the $n \times 2m$ matrices $[\mathbf{S}_k \quad \mathbf{H}_0^{\mathrm{M}}\mathbf{U}_k]$ are required, at complexity $\mathcal{O}(4nm+2nl)$. The product with the $2m \times 2m$ middle matrix is done at $\mathcal{O}(2nm+nl+2m^2)$. Subsequently, $-\mathbf{H}_0^{\mathrm{M}}\mathbf{g}_k$ is obtained at nl multiplications. The total complexity is thus $\mathcal{O}(n(6m+4l)+2m^2)$. Note that if σ_k is set to a constant value, say $\sigma_k = \bar{\sigma}$, then the complexity can be further reduced by storing the matrix $\bar{\mathbf{U}}_k = [\,\bar{\mathbf{u}}_{k-m} \ldots \bar{\mathbf{u}}_{k-1}\,]$, where $\bar{\mathbf{u}}_i = (\mathbf{K}_0 + \bar{\sigma}\mathbf{I})^{-1}\mathbf{u}_i$. The computational cost in this situation is $\mathcal{O}(n(4m+l)+3m^2)$, excluding the updating cost of the vector $\bar{\mathbf{u}}_i$ at order nl. With a constant σ_k only one factorization of $(\mathbf{K}_0 + \bar{\sigma}\mathbf{I})$ is required.

3.4.2. Computations for L-S-BFGS-P. When $\mathbf{A}_k^{\mathrm{P}}$ is used in (3.6) with $\widehat{\mathbf{K}}_0 = (\mathbf{K}_k + \mathbf{A}_0^{\mathrm{P}})$ and $\widehat{\mathbf{Q}}_k = \widehat{\mathbf{K}}_0^{-1}\mathbf{Q}_k$, $\widehat{\mathbf{U}}_k = \widehat{\mathbf{K}}_0^{-1}\mathbf{U}_k$ the inverse has the form

$$(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}})^{-1} = \widehat{\mathbf{K}}_0^{-1}\left(\mathbf{I}_n + \boldsymbol{\Xi}_k\left(\mathbf{M}_k - \boldsymbol{\Xi}_k^T\widehat{\mathbf{K}}_0^{-1}\boldsymbol{\Xi}_k\right)^{-1}\boldsymbol{\Xi}_k^T\widehat{\mathbf{K}}_0^{-1}\right),$$

where $\boldsymbol{\Xi}_k^T\widehat{\mathbf{K}}_0^{-1}\boldsymbol{\Xi}_k = \begin{bmatrix} \mathbf{Q}_k^T\widehat{\mathbf{Q}}_k & \mathbf{Q}_k^T\widehat{\mathbf{U}}_k \\ \mathbf{U}_k^T\widehat{\mathbf{Q}}_k & \mathbf{U}_k^T\widehat{\mathbf{U}}_k \end{bmatrix}$ and $\boldsymbol{\Xi}_k$, $\mathbf{M}_k$ are defined in Theorem 2.3. Assuming that $\mathbf{M}_k$, $\mathbf{Q}_k$, $\mathbf{U}_k$ had previously been updated, computing the search direction $\mathbf{p}_k^{\mathrm{P}} = -(\mathbf{K}_k + \mathbf{A}_k^{\mathrm{P}})^{-1}\mathbf{g}_k$ may be done as follows; First, $\widehat{\mathbf{Q}}_k$, $\widehat{\mathbf{U}}_k$ are computed in $\mathcal{O}(2nlm)$ multiplications. Then the $2m \times 2m$ matrix $\boldsymbol{\Xi}_k^T\widehat{\mathbf{K}}_0^{-1}\boldsymbol{\Xi}_k$ is formed in $\mathcal{O}(3nm^2)$ multiplications. Combining the former terms and solving with the (small) $2m \times 2m$ matrix explicitly, the direction $\mathbf{p}_k^{\mathrm{P}}$ is computed in $\mathcal{O}(n(2lm+3m^2+4m+1)+m^3)$ multiplications. Note that this approach requires an additional $2nm$ storage locations for the matrices $\widehat{\mathbf{Q}}_k$, $\widehat{\mathbf{U}}_k$. Two additional remarks; first, since $\mathbf{Q}_k = \mathbf{V}_k + \mathbf{A}_0^{\mathrm{P}}\mathbf{S}_k$, the update of $\mathbf{Q}_k$ uses $\mathcal{O}(nl)$ multiplications to form a new $\mathbf{v}_k$ and additional nm

TABLE 1

Comparison of computational demands for BFGS,L-BFGS,S-BFGS-M,S-BFGS-P,L-S-BFGS-M,L-S-BFGS-P, excluding storage of $\mathbf{K}_k$ and where solves with $\mathbf{K}_k$ are assumed to cost $\mathcal{O}(nl)$ multiplications and vector multiplies cost $\mathcal{O}(l)$. Terms of order $\mathcal{O}(m)$ or lower are omitted.

Method	Search Direction	Memory	Update
BFGS	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
L-BFGS ((1.3), [5])	$\mathcal{O}(n(4m+2)+m^2)$	$\mathcal{O}(2nm+\frac{3}{2}m^2)$	$\mathcal{O}(1)$
S-BFGS-M ((1.5),[18])	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
S-BFGS-P ((1.6),[18])	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
L-S-BFGS-M ((3.4))	$\mathcal{O}(n(4m+1)+m^2)$	$\mathcal{O}(2nm+\frac{3}{2}m^2)$	$\mathcal{O}(2nm+l))$
L-S-BFGS-M ((3.6))	$\mathcal{O}(n(6m+4l)+m^2)$	$\mathcal{O}(2nm+\frac{3}{2}m^2)$	$\mathcal{O}(n(m+l))$
L-S-BFGS-P ((3.6))	$\mathcal{O}(n(2lm+3m^2+$ $4m+1)+m^3)$	$\mathcal{O}(4nm+3m^2)$	$\mathcal{O}(n(3m+l))$

multiplications if $\mathbf{A}_0^{\mathrm{P}} = \sigma_k \mathbf{I}$. If σ_k remains constant, say $\sigma_k = \bar{\sigma}$, then the update of $\mathbf{Q}_k$ is done at only $\mathcal{O}(nl)$ multiplications, because $\mathbf{A}_0^{\mathrm{P}}\mathbf{S}_k$ does not need to be recomputed each iteration. Second, if $\mathbf{K}_k = \mathbf{K}_0$, in other words if $\mathbf{K}_k$ is a constant matrix then Theorem 2.1 and Theorem 2.3 reduce to the same expressions yielding the same computational complexities.

3.4.3. Memory Usage and Comparison. This section addresses the memory usage of the proposed representations and relates their computational complexities to existing methods. As a overall guideline, the representations from (3.6) use $2nm+4m^2$ storage locations, excluding the $\widehat{\mathbf{K}}_0$ term. This estimate is refined if the particular structure of the matrix $\mathbf{M}_k$ is taken into consideration. For example, the matrices $\mathbf{T}_k^{\mathrm{U}}$ and $\mathbf{D}_k^{\mathrm{U}}$ from Theorem 2.1 are upper triangular and diagonal, respectively. Thus, when $\mathbf{H}_0^{\mathrm{M}} = \sigma_k \mathbf{I}$, and when the matrix $\mathbf{U}_k^T\mathbf{U}_k \in \mathbb{R}^{m\times m}$ is stored and updated, the memory requirement for the limited memory version of $\mathbf{H}_k^{\mathrm{M}}$ in Theorem 2.1 are $\mathcal{O}(2nm + \frac{3}{2}m^2 + m)$ locations. We summarize the computational demands of the different methods in the Table 1: Note that when $m \ll n$ and $l \ll n$ L-BFGS, L-S-BFGS-M and L-S-BFGS-P enable computations with complexity lower than n^2 and therefore allow for large values of n.

4. Numerical Experiments. This section describes the numerical experiments for the proposed methods in Section 3. The numerical experiments are carried out in MATLAB 2016a on a MacBook Pro @2.6 GHz Intel Core i7, with 32 GB of memory. The experiments are divided into four parts. In Experiment I, we investigate an optimal initialization strategy. Using the outcomes of the first experiment, Experiment II compares the limited memory methods with the full-memory methods. For consistency, the tests in this experiment are on the same 61 CUTEst [14] problems as in [18], unless otherwise noted. In Experiment III, the proposed methods are used in two structured problem applications. In the first application, we use classification data from LIBSVM (a library for support vector machines [6]) in order to solve regularized logistic regression problems. The second application is an optimal control problem from PDE-constrained optimization. In Experiment IV, the proposed methods and L-BFGS and IPOPT [22] solvers are compared.

Performance profiles as in [15] are provided. These profiles are an extension of the well known profiles of Dolan and Moré [10]. We compare the number of iterations and the total computational time for each solver on the test set of problems. The

performance metric $\rho_s(\tau)$ with a given number of test problems n_p is

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

where $t_{p,s}$ is the "output" (i.e., iterations or time) of "solver" s on problem p. Here 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. The extended performance profiles are same as the classical ones for $\tau \geq 1$. In the profiles we include include a dashed vertical grey line, to indicate this point. In all experiments the line search parameters are set to $c_1 = 1e^{-4}$ and $c_2 = 0.9$.

4.1. Experiment I. This experiment investigates the initialization strategies from Section 3. To this end, the problems in this experiment are not meant to be overly challenging, yet they are meant to enable some variations. Therefore, we define the quadratic functions

$$Q_i(\mathbf{x}; \phi, r) \equiv \frac{1}{2}\mathbf{x}^T(\phi \cdot \mathbf{I} + \mathbf{Q}_i\mathbf{D}_i\mathbf{Q}_i^T)\mathbf{x},$$

with scalar parameters $0 < \phi$, $1 \leq r \leq n$ and where $\mathbf{D}_i \in \mathbb{R}^{r\times r}$ is a diagonal matrix and $\mathbf{Q}_i \in \mathbb{R}^{n\times r}$ has orthonomal gaussian columns. Note that r eigenvalues of the Hessian $\nabla^2 Q_i$ are the diagonal elements of $\phi \cdot \mathbf{I} + \mathbf{D}_i$, while the remaining $(n-r)$ eigenvalues are ϕ. Therefore, by varying ϕ, r, and the elements of $\mathbf{D}_i$, Hessian matrices with different spectral properties are formed. In particular, when $r \ll n$, the eigenvalues are clustered around ϕ. In the experiments of this section two values are investigated; specifically, $\phi = \{1, 1000\}$. The structured objective functions from (1.4) are defined by

$$\widehat{k}(\mathbf{x}) = \mathbf{x}^T\mathbf{g} + Q_1(\mathbf{x}; \phi, r), \qquad \widehat{u}(\mathbf{x}) = Q_2(\mathbf{x}; \phi, r). \tag{4.1}$$

We refer to the objective functions $f(\mathbf{x}) = \widehat{k}(\mathbf{x}) + \widehat{u}(\mathbf{x})$ defined by (4.1) as *structured quadratics*. The problems in this experiment have dimensions $n = j \cdot 100$ with corresponding $r = j \cdot 10$ for $1 \leq j \leq 7$. Since some of the problem data in this experiment is randomly generated (e.g., the orthonormal matrices $\mathbf{Q}_i$), the experiments are repeated five times for each n. The reported results are of the average values of the five individual runs. For all solvers we set $m = 8$ (memory parameter), $\epsilon = 5 \times 10^{-6}$ ($\|\mathbf{g}_k\|_\infty \leq \epsilon$), and maximum iterations to 10,000.

4.1.1. Experiment I.A: L-S-BFGS-M. Experiment I.A compares the four L-S-BFGS-M initializations on the structured quadratic objective functions with eigenvalues clustered around 1. In particular, $\phi = 1$, and the elements of $\mathbf{D}_i$ are uniformly distributed in the interval $[0, 999]$. The results are displayed in Figure 1. We observe that in terms of number of iterations, Init. 4 (red) and Init. 3 (purple) perform similarly and that also Init. 2 (green) and Init. 1 (blue) perform similarly. Overall, Init. 4 and Init. 3 requirer fewer iterations on the structured quadratics. Moreover, the solid lines are above the dashed ones for both pairs. This indicates that including only gradient information in $\widehat{\mathbf{u}}_k$ and in the initialization strategy, as opposed to also including 2^{nd} derivative information from $\mathbf{u}_k$, may be desirable for this problem. Init. 1 and Init. 2 are fastest on these problems. Even though these initializations require a larger number of iterations, they can be faster because the line searches terminate more quickly. Next, the four L-S-BFGS-M initializations are compared on structured

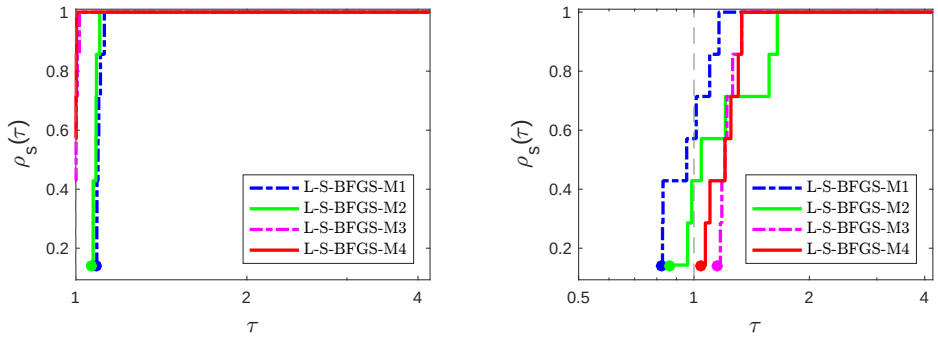

FIG. 1. *Comparison of initialization strategies for L-S-BFGS-M on problems with eigenvalues clustered around 1 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1$. *Left: number of iterations; right: time.*

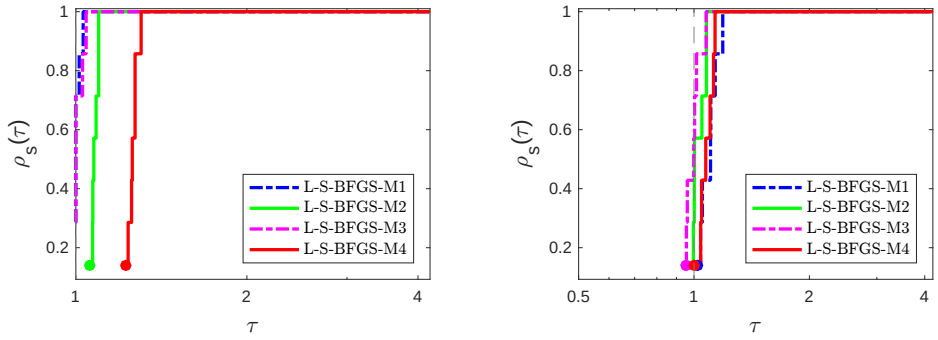

FIG. 2. *Comparison of initialization strategies for L-S-BFGS-M on problems with eigenvalues clustered around 1,000 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1000$. *Left: number of iterations; right: time.*

quadratic objective functions with eigenvalues clustered around 1000. In particular, $\phi = 1000$, and the elements of $\mathbf{D}_i$ are uniformly distributed in the interval $[-999, 0]$. The results are displayed in Figure 2.

For the large clustered eigenvalues Init. 1 and 3 require the fewest iterations, while Init. 3 appears fastest overall.

4.1.2. Experiment I.B: L-S-BFGS-P. Experiment I.B compares the four L-S-BFGS-P initializations. As in Section 4.1.1 experiments on problems with eigenvalues clustered at 1 and at 1000 are performed. The respective outcomes are in Figure 3 and Figure 4.

In Figure 3 we observe that, similar to Figure 1, Init. 3 and Init. 4 do best in iterations, while Init. 1 does best in time.

For the experiments in Figure 4, Init. 2 and Init. 3 do best in iterations. To analyze the properties of the scaling factor σ_k in greater detail, Section 4.1.3 describes experiments that relate σ_k to eigenvalues.

4.1.3. Experiment I.C: Eigenvalue Estimation. In Experiment I.C we investigate the dynamics of σ_k in the four initialization strategies from (3.1) on a fixed problem as the iteration count k increases. In particular, we use one representative run from the average results of the preceding two subsections, where $n = 100$ and

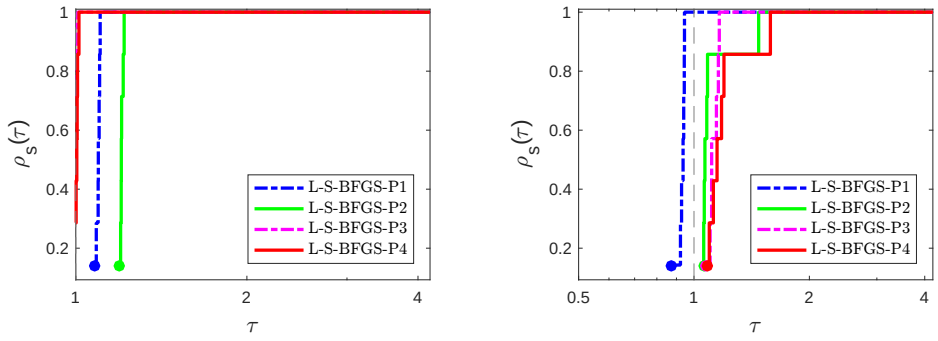

FIG. 3. *Comparison of initialization strategies for L-S-BFGS-P on problems with eigenvalues clustered around 1 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1$. *Left: number of iterations; right: time.*

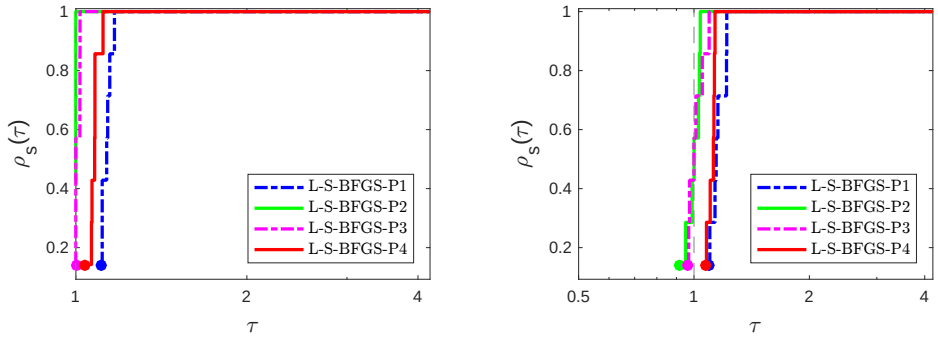

FIG. 4. *Comparison of initialization strategies for L-S-BFGS-P on problems with eigenvalues clustered around 1,000 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1000$. *Left: number of iterations; right: time.*

$r = 10$. In Figure 5 the evolution of σ_k of all four initializations for both; L-S-BFGS-M and L-S-BFGS-P is displayed on a structured quadratic problem with eigenvalues clustered at 1. In Figure 6 the same quantities are displayed for structured quadratic problems with eigenvalues clustered at 1000. In green $\bar{\lambda}_{1\leq n}$ and $\bar{\lambda}_{1\leq r}$ are displayed, which correspond to the median taken over the first $1, 2, \cdots, n$ (all) and the first $1, 2, \cdots, r$ eigenvalues, respectively. Because in Figure 5 the eigenvalues are clustered around 1, $\bar{\lambda}_{1\leq n} = 1$. In Figure 6 the eigenvalues are clustered around 1000 and $\bar{\lambda}_{1\leq r} = 1000$. In red $\bar{\sigma}_k$ is the average σ_k value over all iterations.

Across all plots in Figures 5 and 6 we observe that the dynamics of σ_k for L-S-BFGS-M and L-S-BFGS-P are similar. Moreover, the average $\bar{\sigma}_k$ is higher for Init. 1 and Init. 2 than for Init. 3 and Init. 4. The variability of Init. 2 appears less than that of Init. 1, while the variability of Init. 4 appears less than that of Init. 3. We observe that Init. 1 and 2 approximate a large eigenvalue well, whereas Init. 3 and Init. 4 approximate smaller eigenvalues better (cf. Figure 5 lower half). Since large σ_k values typically result in shorter step lengths (step computations use $1/\sigma_k$), choosing Init. 1 and Init. 2 result in shorter step lengths on average. Taking shorter average steps can be a desirable conservative strategy when the approximation to the full Hessian matrix is not very accurate. Therefore as a general guideline, Init. 1 and Init. 2 appear more suited for problems in which it is difficult to approximate the

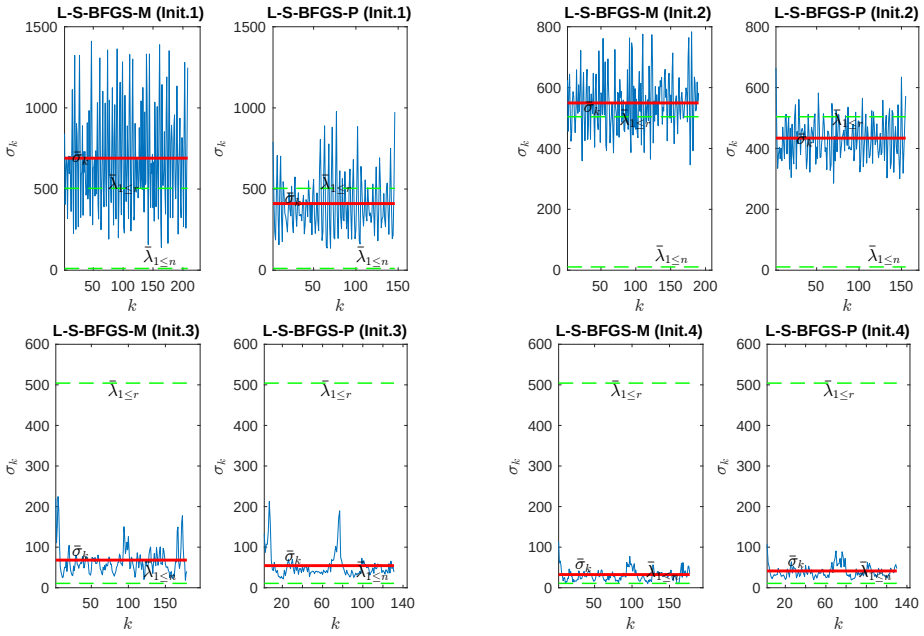

FIG. 5. *Eigenvalue estimation with initialization parameter* σ_k. *The eigenvalues are clustered around 1 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1$.

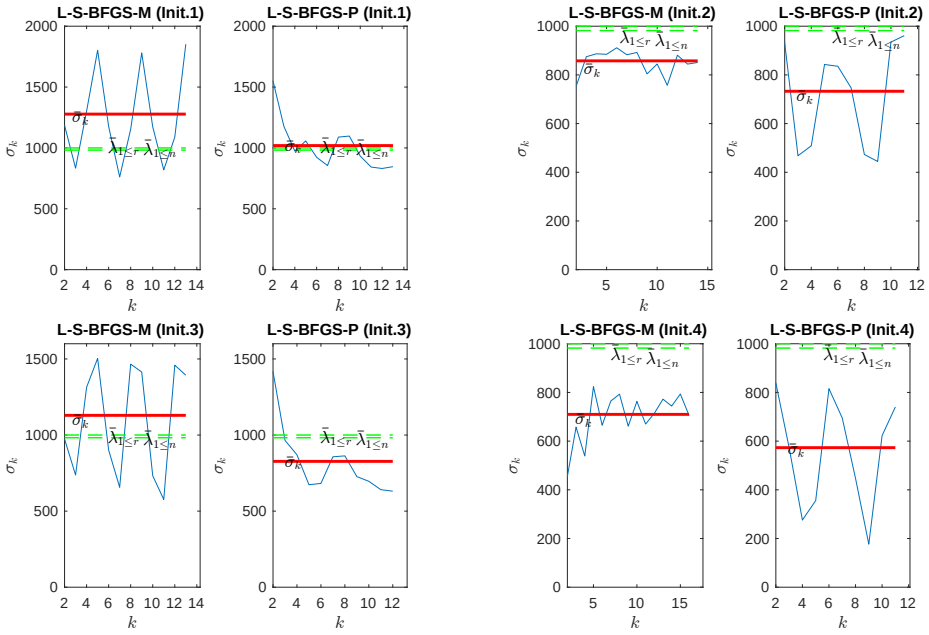

FIG. 6. *Eigenvalue estimation with scaling parameter. The eigenvalues are clustered around 1,000 with* $1 \leq \lambda_r \leq 1000$ *and* $\lambda_{r+1} = \cdots = \lambda_n = 1000$.

Hessian accurately, and Init. 1 and Init. 2 are more suited for problems in which larger step sizes are desirable.

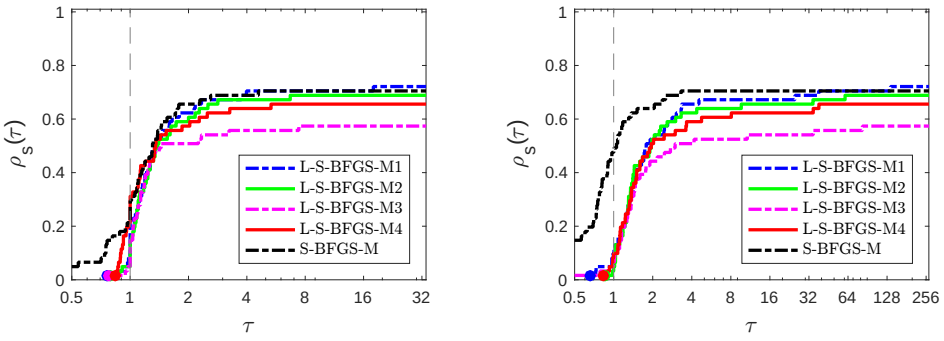

FIG. 7. *Comparison of four initialization strategies of L-S-BFGS-M from (3.1) to the full-recursive method S-BFGS-M (corresponding to (1.5)). The limited memory parameter is $m = 8$. Left: number of iterations; right: time.*

4.2. Experiment II. Experiment II compares the limited memory structured formulas with the full-memory update formulas from Petra et al. [18]. The full-memory algorithms from [18], which use (1.5) and (1.6), are called S-BFGS-M and S-BFGS-P, respectively. The line search procedures of the limited memory structured BFGS algorithms (Algorithms 3.1 and 3.2) are the same as for the full memory algorithms. Moreover, the initializations in the full memory algorithms are set as $\mathbf{A}_0^{\mathrm{M}} = \bar{\sigma}\mathbf{I}_n$ for S-BFGS-M, and $\mathbf{A}_0^{\mathrm{P}} = \bar{\sigma}\mathbf{I}_n$ for S-BFGS-P, where $\bar{\sigma} = 10^i$ for the first $i \geq 0$ that satisfies $(10^i\mathbf{I}_n + \mathbf{K}_0) \succ 0$ (usually $i = 0$). The experiments are divided into two main parts. Experiment II.A. tests the limited memory structured BFGS-Minus versions corresponding to Algorithm 3.1. Experiment II.A. is further subdivided into the cases in which the memory parameters are $m = 8$ and $m = 50$. These values represent a typical value ($m = 8$) and a relatively large value ($m = 50$), cf. e.g., [2]. Experiment II.B. tests the limited memory structured BFGS-Plus versions corresponding to Algorithm 3.2. As before, Experiment II.B. is further subdivided into the cases in which the memory parameters are $m = 8$ and $m = 50$. For all the solvers, we set $\epsilon = 1 \times 10^{-6}$ ($\|\mathbf{g}_k\|_\infty \leq \epsilon$) and maximum iterations to 1,000.

4.2.1. Experiment II.A: L-S-BFGS-M. In Experiment II.A we compare the limited memory implementations of Algorithm 3.1 with initialization strategies in (3.1) with the full-recursive S-BFGS-M method from (1.5). The solvers are tested on all 62 CUTEst problems from [18]. Figure 7 contains the results for the limited memory parameter $m = 8$.

We observe that the full-memory S-BFGS-M (black) does well in terms of number of iterations and execution time. However, L-S-BFGS-M1 (Init. 1, blue), a limited memory version with memory of only $m = 8$, does comparatively well. In particular, this strategy is able to solve one more problem, as indicated by the stair step at the right end of the plot.

Figure 8 shows the results for the limited memory parameter $m = 50$. A larger limited memory parameter makes using limited memory structured matrices more computationally expensive but is also expected to increase the accuracy of the quasi-Newton approximations.

Note that the outcomes of S-BFGS-M (black) in Figure 8 are the same as those in Figure 7, because it does not depend on the memory parameter. For the limited memory versions we observe that the outcomes of L-S-BFGS-M2 (green) improve

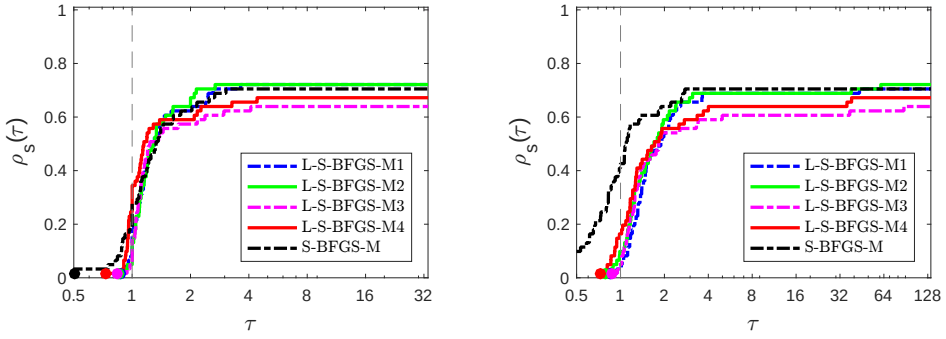

FIG. 8. *Comparison of four initialization strategies of L-S-BFGS-M from (3.1) with the full-recursive method S-BFGS-M (corresponding to (1.5)). The limited memory parameter is $m = 50$. Left: number of iterations; right: time.*

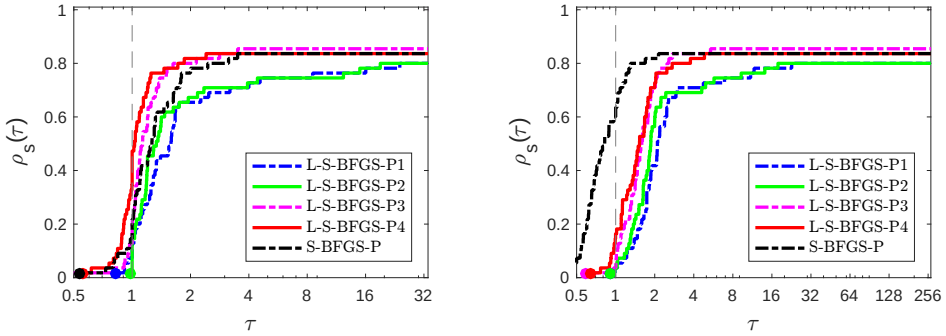

FIG. 9. *Comparison of four initialization strategies of L-S-BFGS-P from (3.1) to the full-recursive method S-BFGS-P (corresponding to (1.6)). The limited memory parameter is $m = 8$. Left: number of iterations; right: time.*

notably, whereas the other limited memory versions remain roughly unchanged. Using the initialization strategies (Init. 1 or Init. 2), limited memory solvers are able to solve one more problem than the full-memory method can, as indicated by the highest ending lines in the plot. We believe that Init. 1 and Init. 2 (see section 4.1.3) generate initialization parameters σ_k that are on average larger than those generated by Init. 3 or Init. 4. These larger values in turn result in shorter average step sizes, which appears advantageous on general nonlinear problems.

4.2.2. Experiment II.B: L-S-BFGS-P. In Experiment II.B we compare the versions of Algorithm 3.2 using the initialization strategies from (3.1) with the full memory recursive S-BFGS-P method (1.6). The solvers are run on 55 of the 62 CUTEst problems from [18] for which $n \leq 2500$. Figure 9 contains the results for the limited memory parameter $m = 8$:

We observe that for a relatively small memory parameter $m = 8$, L-S-BFGS-M3 (Init. 3, purple) solves the most problems. L-S-BFGS-M4 (Init. 4, red) requires the fewest iterations, as indicated by the highest circle on the y-axis in the left panel of Figure 9.

Figure 10 shows the results for the limited memory parameter $m = 50$. A larger parameter makes using limited memory structured matrices more computationally

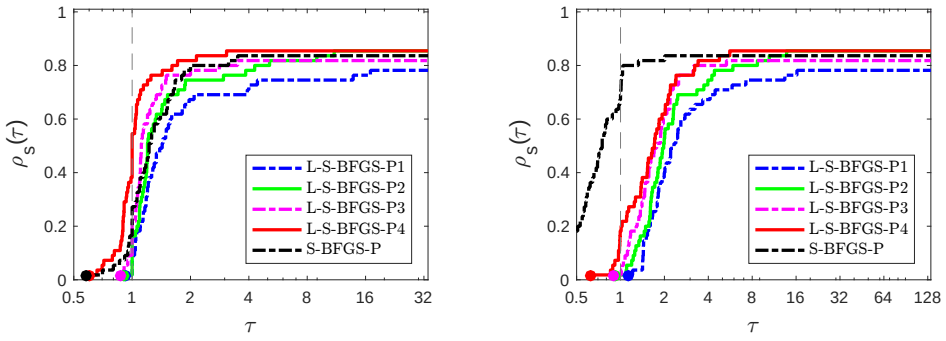

FIG. 10. *Comparison of four initialization strategies of L-S-BFGS-P from (3.1) to the full-recursive method S-BFGS-P (corresponding to (1.6)). The limited memory parameter is $m = 50$. Left: number of iterations; right: time.*

expensive but is also expected to increase the accuracy of the quasi-Newton approximations.

Note that the outcomes of S-BFGS-P in Figure 10 are the same as in Figure 9, because the full-memory solver does not depend on the memory parameter. For a larger memory $m = 50$, the outcomes of L-S-BFGS-P2 (green) and L-S-BFGS-P4 (red) improve notably. Overall, L-S-BFGS-P4 solves the most problems.

From the experiments in this section, we find that initialization strategies Init.1 and Init. 2 appear most desirable for L-S-BFGS-M, whereas Init. 4 and Init. 2 appear most desirable for L-S-BFGS-P.

4.3. Experiment III. This section describes two applications in which the structured algorithms are applied. For all solvers we set $m = 8$ (memory parameter), $\epsilon = 1 \times 10^{-6}$ ($\|\mathbf{g}_k\|_\infty \leq \epsilon$) and maximum iterations to 10000. Since some of the problems in this section are large we use the techniques describe in Subsection 3.4 throughout the experiments.

4.3.1. Experiment III.A: Logistic Regressions. Experiment III.A tests the proposed methods on smooth-structured objective functions from machine learning, as described, for example, in [20]. In particular, logistic regression problems use smooth objective functions for classification tasks (for instance, [4]), which often depend on a large number of data points and many variables. The classification problems are defined by the data pairs $\{\mathbf{d}_i, y_i\}_{i=1}^D$, where the so-called feature vectors $\mathbf{d}_i \in \mathbb{R}^n$ may be large, and the so-called labels $y_i \in \{-1, 1\}$ are scalars. In [21] regularized logistic regression problems are described in which the objective function is composed of two terms. The optimization problems are formulated as

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \ \frac{\lambda}{2}\|\mathbf{x}\|_2^2 + \sum_{i=1}^{D} \log\left(1 + \exp(-y_i \mathbf{x}^T \mathbf{d}_i)\right),$$

where $\lambda > 0$. The regularization term, $\frac{\lambda}{2}\|\mathbf{x}\|^2$, has a second derivative,$\lambda\mathbf{I}$, that is readily available. Therefore, we define the known and unknown components for this problem as

$$\widehat{k}(\mathbf{x}) = \frac{\lambda}{2}\|\mathbf{x}\|_2^2, \qquad \widehat{u}(\mathbf{x}) = \sum_{i=1}^{D} \log\left(1 + \exp(-y_i \mathbf{x}^T \mathbf{d}_i)\right). \tag{4.2}$$

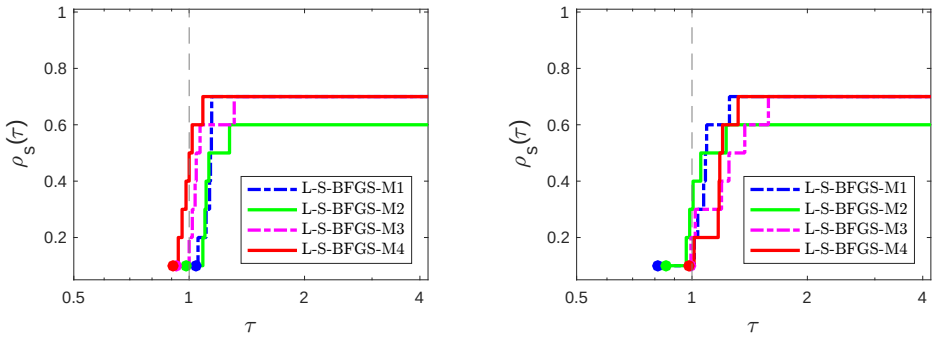

FIG. 11. *Comparison of L-S-BFGS-M solvers on 10 logistic regression classification problems using data from LIBSVM. Left: number of iterations, right: time.*

This experiment's data was obtained from `www.csie.ntu.edu.tw/~cjlin/libsvm/` (retrieved on 10/03/19). Ten problems were used, with problem dimensions listed in Table 2.

TABLE 2

List of dimensions for 10 LIBSVM logistic regression problems. Here D denotes the number of training pairs $\{\mathbf{d}_i, y_i\}_{i=1}^{D}$, and n denotes the number of variables/feature weights (the size of the problem).

Problem	D	n
rcv1	20242	47236
duke	34	7129
gisette	6000	5000
colon_cancer	62	2000
leukemia	38	7129
real_sim	72309	20958
madelon	2000	500
w8a	49749	300
mushrooms	2000	500
a9a	32561	123

Some of the problems are large, with $n \geq 5000$. Because the Hessian of $\widehat{k}(\mathbf{x})$ is constant, the search directions of L-S-BFGS-M and L-S-BFGS-P are the same, when the computations with L-S-BFGS-M are done as described in Section 3.4.1. Thus we focus on presenting the results with this method. The regularization parameter is set as $\lambda = 10^{-3}$. The results of the experiments are shown in Figure 11.

In this experiment L-S-BFGS-M1 (blue) appears the most robust. This solver is based on Init. 1, which typically yields the largest values of σ_k (see section 4.1.3). Overall, we believe that the dashed lines, which typically produce larger values of σ_k, do best on these problems. Larger values of σ_k typically result in shorter steps (see e.g., Line 3 in Algorithm 3.1 where $\mathbf{H}_0 = (1/\sigma_k)\mathbf{I}$). Therefore when, as in this case, the Hessian of $\widehat{u}(\mathbf{x})$ may be difficult to accurately approximate, because it depends on data, it appears advantageous to use initialization strategies, which produce relatively large σ_k.

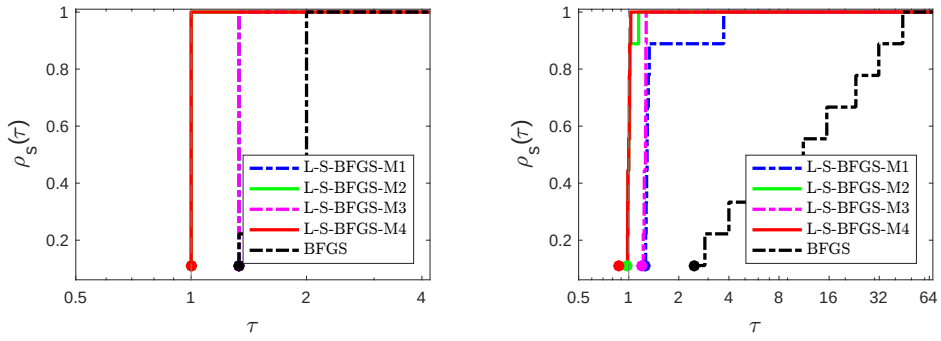

FIG. 12. *Comparison of L-S-BFGS-M solvers on PDE constrained optimal control problems. The dimensions of the problems are* $n = (10 \times j - 2)^2$ *for* $j = 2, 3, \cdots, 10$. *Left: number of iterations, right: time.*

4.3.2. Experiment III.B: Optimal Control Problems. This experiment describes a typical situation in PDE constrained optimization. In particular, if the PDE is nonlinear, then we can compute gradients efficiently using the adjoint equation, but Hessians of the unknown part cannot be computed efficiently. Denoting u as the horizontal axis and v as the vertical axis, then 2D Poisson problems, with an unknown control $x(u, v)$, are defined by the differential equation: $y_{uu} + y_{vv} = x$. The solution $y(u, v)$ has known boundary values on a box $(u, v) \in [0, 1]^2$; in other words, $y(0, v)$, $y(1, v)$, $y(u, 0)$, and $y(u, 1)$ are known. Discretizing the domain and splitting it into an interior and boundary part, we get for the optimal control problem

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\left\{\|\mathbf{x}\|_2^2 + \|\mathbf{y}(\mathbf{x}) - \mathbf{y}^*\|_2^2\right\} \quad \text{subject to} \quad \mathbf{A}\mathbf{y} = \mathbf{x} + \mathbf{g},$$

where $\mathbf{g} \in \mathbb{R}^n$ represents a vector with boundary information, $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a matrix resulting from a 5-point stencil finite difference discretization of the partial derivatives, and $\mathbf{y}^*$ are fixed data values. Because the Hessian of the regularization term, $\frac{1}{2}\|\mathbf{x}\|_2^2$, is straightforward to compute, we define the structured objective function by

$$\widehat{k}(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2, \qquad \widehat{u}(\mathbf{x}) = \frac{1}{2}\|\mathbf{y}(\mathbf{x}) - \mathbf{y}^*\|_2^2, \tag{4.3}$$

using $\mathbf{y}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} + \mathbf{g})$. The number of variables is defined by the formula $n = (10 \times j - 2)^2$, where $j = 2, 3, \cdots, 10$, which corresponds to discretizations with $20, 30, \cdots, 100$ mesh points in one direction. The largest problem has $n = 9604$ variables. For comparison we also include the implementation of a "standard" BFGS method from [18], which uses the same line search as do the limited memory structured methods.

In this experiment, L-S-BFGS-M3 and L-S-BFGS-M1 (dashed purple and blue) appear to do best overall. Both solvers typically generate σ_k values, which are larger than the ones from the solvers with solid lines (cf. observations from section 4.1.3). Larger values of σ_k yield shorter step lengths, which may be regarded as a more conservative strategy. The effect of computing search directions by the Sherman-Morrison-Woodbury formula in Line 3 from Algorithm 3.1, appears in the right-hand plot of Figure 12. In particular, the limited memory versions do not require solves with linear systems that depend on the dimension n, whereas the BFGS implementation from [18] is based on solves with $n \times n$ dimensional systems.

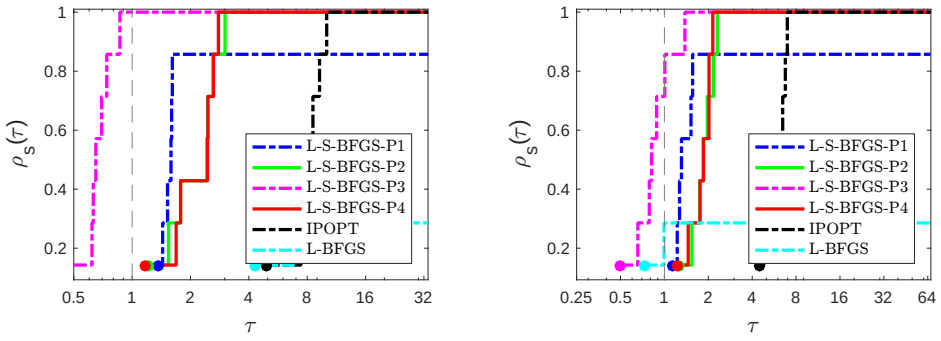

FIG. 13. *Comparison of L-S-BFGS-P on structured objective functions to IPOPT and L-BFGS-B. Left: number of iterations, right: time.*

4.4. Experiment IV. In this experiment our structured solvers are tested next to L-BFGS and IPOPT [22] (we use a precompiled Mex file with IPOPT 3.12.12, MUMPS and MA57). The objective function is a structured quartic function

$$f(\mathbf{x}) = \widehat{k}(\mathbf{x}) + \widehat{u}(\mathbf{x}), \quad \widehat{k}(\mathbf{x}) = \frac{1}{12}\sum_{i=1}^{n}(a_i^2 x_i^4 + 12 x_i g_i), \quad \widehat{u}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} q_i x_i^2, \tag{4.4}$$

where the data a_i, g_i and q_i are random normal variables with $n = j \times 100, 1 \leq j \leq 7$. The starting values are all ones, i.e., $\mathbf{x}_0 = \mathbf{1}$. We specify the limited memory BFGS option for IPOPT using the setting `hessian_approximation='limited memory'` and tolerances by `tol=9.5e-10` and `acceptable_tol=9.5e-10`. The L-BFGS solver uses the same strong Wolfe line search as the algorithms from section 3. For all solvers we set $m = 8$ (memory parameter), and maximum iterations to 10,000. A solver is regarded to have converged when $\|\mathbf{g}_k\|_\infty \leq 9.5 \times 10^{-5}$. The average outcomes of 5 runs of the experiments are in Figure 13.

IPOPT and the L-S-BFGS-P solvers converge to the specified tolerances on all problems. L-BFGS converges on about 28% of the problems. The outcomes of the number of iterations (left plot) and computational times (right plot) in Figure 13 are consistent. In particular, we observe that the differences in the number of iterations are roughly reflected in the difference in the computational times. In this problem the known Hessian is not constant, and including second-order information in the quasi-Newton approximations appears to yield better overall approximations such that fewer iterations are required.

5. Conclusions. In this article we develop the compact representations of the structured BFGS formulas proposed in Petra et al. [18]. Limited memory versions of the compact representations with four initialization strategies are implemented in two line search algorithms. The proposed limited memory compact representations enable efficient search direction computations by the Sherman-Morrison-Woodbury formula and the use of efficient initialization strategies. The proposed methods are compared in a collection of experiments, which include the original full-memory methods. The structured methods typically require fewer total iterations than do the unstructured approaches. Among the four proposed initialization strategies, initializations 1 and 2 appear best for the structured minus methods (L-S-BFGS-M), whereas initializations 4 and 2 appear robust for the structured plus (L-S-BFGS-P) methods.

Acknowledgments. 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. through the Project "Multifaceted Mathematics for Complex Energy Systems." This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] Barzilai, J., and Borwein, J. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis 8*, 1 (01 1988), 141–148.

[2] Boggs, P., and Byrd, R. Adaptive, limited-memory bfgs algorithms for unconstrained optimization. *SIAM Journal on Optimization 29*, 2 (2019), 1282–1299.

[3] Broyden, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics 6*, 1 (03 1970), 76–90.

[4] Byrd, R. H., Chin, G. M., Neveitt, W., and Nocedal, J. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization 21* (2011), 977–995.

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

[6] Chang, C.-C., and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 27:1–27:27.

[7] Dennis, J., and Moré, J. Quasi-newton methods, motivation and theory. *SIAM Review 19* (1977), 46–89.

[8] Dennis, Jr., J. E., Gay, D. M., and Walsh, R. E. An adaptive nonlinear least-squares algorithm. *ACM Trans. Math. Softw. 7*, 3 (Sept. 1981), 348–368.

[9] Dennis, Jr, J. E., Martinez, H. J., and Tapia, R. A. Convergence theory for the structured bfgs secant method with an application to nonlinear least squares. *J. Optim. Theory Appl. 61*, 2 (May 1989), 161–178.

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

[11] Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal 13*, 3 (01 1970), 317–322.

[12] Gill, P. E., and Murray, W. Algorithms for the solution of the nonlinear least-squares problem. *SIAM J. Numer. Anal. 11* (Mar. 2010), 311–365.

[13] Goldfarb, D. A family of variable-metric methods derived by variational means. *Math. Comp. 24* (1970), 23–26.

[14] Gould, N. I. M., Orban, D., and Toint, P. L. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Software 29*, 4 (2003), 373–394.

[15] Mahajan, A., Leyffer, S., and Kirches, C. 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.

[16] Moré, J. J., and Thuente, D. J. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw. 20*, 3 (Sept. 1994), 286–307.

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

[18] Petra, C., Chiang, N., and Anitescu, M. A structured quasi-newton algorithm for optimizing with incomplete hessian information. *SIAM Journal on Optimization 29*, 2 (2019), 1048–1075.

[19] Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Math. Comp. 24* (1970), 647–656.

[20] Sra, S., Nowozin, S., and Wright, S. J. *Optimization for Machine Learning*. The MIT Press, 2011.

[21] Teo, C. H., Vishwanthan, S., Smola, A. J., and Le, Q. V. Bundle methods for regularized risk minimization. *J. Mach. Learn. Res. 11* (Mar. 2010), 311–365.

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

[23] Yabe, H., and Takahashi, T. Factorized quasi-newton methods for nonlinear least squares problems. *Mathematical Programming 11*, 75 (1991).

[24] Zhu, C., Byrd, R., and Nocedal, J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software 23* (1997), 550–560.