Recognizing underlying sparsity in optimization

Kim, S.; Kojima, M.; Toint, P.

Published in:
Mathematical Programming

DOI:
10.1007/s10107-008-0210-4

Publication date:
2009

Document Version
Early version, also known as pre-print

Recognizing Underlying Sparsity in Optimization[♯]

Sunyoung Kim[⋆], Masakazu Kojima[†] and Philippe Toint[‡].
May 2006. Revised May 2007, December 2007.

Abstract.
Exploiting sparsity is essential to improve the efficiency of solving large optimization problems. We present a method for recognizing the underlying sparsity structure of a nonlinear partially separable problem, and show how the sparsity of the Hessian matrices of the problem's functions can be improved by performing a nonsingular linear transformation in the space corresponding to the vector of variables. A combinatorial optimization problem is then formulated to increase the number of zeros of the Hessian matrices in the resulting transformed space, and a heuristic greedy algorithm is applied to this formulation. The resulting method can thus be viewed as a preprocessor for converting a problem with hidden sparsity into one in which sparsity is explicit. When it is combined with the sparse semidefinite programming (SDP) relaxation by Waki *et al.* for polynomial optimization problems (POPs), the proposed method is shown to extend the performance and applicability of this relaxation technique. Preliminary numerical results are presented to illustrate this claim.

♯ This manuscript was also issued as Report 06/02, Department of Mathematics, University of Namur, 61, rue de Bruxelles, B-5000 Namur, Belgium, EU.

⋆ Department of Mathematics, Ewha Women's University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. The research was supported by Kosef R01-2005-000-10271-0. skim@ewha.ac.kr

† Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. This research was supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. kojima@is.titech.ac.jp

‡ Department of Mathematics, The University of Namur, 61, rue de Bruxelles, B5000 - Namur, Belgium, EU. philippe.toint@fundp.ac.be

1 Introduction

Sparsity plays a crucial role in solving large-scale optimization problems in practice because its exploitation greatly enhances the efficiency of many numerical solution algorithms. This is in particular the case for sparse SDP relaxation for polynomial optimization problems (POPs) [22], our original motivation, but the observation is much more widely relevant: it is indeed often highly desirable to find sparse formulations of large optimization problems of interest. Sparsity is however fragile in the sense that it is not invariant under linear transformations of the problem variables. This is in contrast with another description of problem structure: partial separability, a concept originally proposed by Griewank and Toint [8] in connection with the efficient implementation of quasi-Newton methods for large unconstrained minimization. While these authors showed that every sufficiently smooth sparse problem of this type must be partially separable, the present paper explores the reverse implication: our objective is indeed to show that partial separability can often be used to improve exploitable sparsity.

Let f be a real-valued and twice continuously differentiable (C^2) function defined on the n-dimensional Euclidean space $\mathbb{R}^n$. f is said to be *partially separable* if it is represented as the sum of element C^2 functions $f_\ell : \mathbb{R}^m \to \mathbb{R}$ ($\ell = 1, \ldots, m$) such that

$$f_\ell(\boldsymbol{x}) = \hat{f}_\ell(\boldsymbol{A}_\ell \boldsymbol{x}) \text{ for every } \boldsymbol{x} \in \mathbb{R}^n, \tag{1}$$

where $\boldsymbol{A}_\ell$ denotes an $n_\ell \times n$ matrix with full row rank, $n_\ell < n$ for each ℓ and $\hat{f}_\ell$ is a real-valued and C^2 function defined on $\mathbb{R}^{n_\ell}$. In practice, the dimension n_ℓ of the domain of each element function $\hat{f}_\ell$ is often much smaller than the dimension n of the *problem space* $\mathbb{R}^n$. The vectors $\boldsymbol{u}_\ell = \boldsymbol{A}_\ell \boldsymbol{x}$, called *internal variables* of the ℓ-th element, are thus of much smaller size than $\boldsymbol{x}$. Since the Hessian matrices $\nabla_{u_\ell u_\ell} \hat{f}_\ell(\boldsymbol{u}_\ell)$ of the element functions $\hat{f}_\ell(\boldsymbol{u}_\ell)$ ($\ell = 1, \ldots, m$) and $\nabla_{xx} f(x)$ of f are such that

$$\nabla_{xx} f(\boldsymbol{x}) = \sum_{\ell=1}^{m} \boldsymbol{A}_\ell^T \, \nabla_{u_\ell u_\ell} \hat{f}_\ell(\boldsymbol{u}_\ell) \boldsymbol{A}_\ell \text{ for every } \boldsymbol{x} \in \mathbb{R}^n, \tag{2}$$

(where the superscript T denotes the transpose of a matrix), we see that, when the matrices $\boldsymbol{A}_\ell$ are known and constant, the Hessian $\nabla_{xx} f(x)$ can be then obtained from the family of small $n_\ell \times n_\ell$ element Hessians $\nabla_{u_\ell u_\ell} \hat{f}_\ell(\boldsymbol{u}_\ell)$. In partitioned quasi-Newton methods (e.g. [8, 9, 10]), this observation is exploited by updating an approximation $\boldsymbol{B}_\ell$ of each element Hessian in its range $\mathbb{R}^{n_\ell}$, instead of an approximation $\boldsymbol{B}$ of the $n \times n$ Hessian matrix $\nabla_{xx} f(x)$ in the entire problem space. Under this assumption on the size of the dimension n_ℓ, the size of each $\boldsymbol{B}_\ell$ is much smaller than $\boldsymbol{B}$, so that updates in the smaller dimensional space associated with each element considerably improve the computational efficiency of the resulting minimization methods. Research on partial separability has focused on local convergence of partitioned quasi-Newton methods [10], convex decompositions of partially separable functions [11], and a detection of partially separability using automatic differentiation [5]. The LANCELOT optimization package [3, 7] makes efficient numerical use of partial separability. See also [20].

We now explore the links between partial separability and sparsity further. For every real valued function h on $\mathbb{R}^n$, we call $\boldsymbol{w} \in \mathbb{R}^n$ an *invariant direction* of h if $h(\boldsymbol{x} + \lambda \boldsymbol{w}) = h(\boldsymbol{x})$ for every $\boldsymbol{x} \in \mathbb{R}^n$ and every $\lambda \in \mathbb{R}$. The set $\mathrm{Inv}(h)$ of all invariant directions of h forms

a subspace of $\mathbb{R}^n$, which we call the *invariant subspace* of h, in accordance with [20] (it is called the null space of h in [8]). We also see that the condition

$$h(\boldsymbol{x}+\boldsymbol{w}) = h(\boldsymbol{x}) \text{ for every } \boldsymbol{x} \in \mathbb{R}^n \text{ and every } \boldsymbol{w} \in \text{Inv}(h)$$

is characterized by the existence of an $n_h \times n$ matrix $\boldsymbol{A}$ (with $n_h = n - \dim[\text{Inv}(h)]$) and a function $\hat{h} : \mathbb{R}^{n_h} \to \mathbb{R}$ such that

$$h(\boldsymbol{x}) = \hat{h}(\boldsymbol{A}\boldsymbol{x}) \text{ for every } \boldsymbol{x} \in \mathbb{R}^n \tag{3}$$

[11, 20], where the rows of $\boldsymbol{A}$ form a basis of the n_h-dimensional subspace orthogonal to $\text{Inv}(h)$. Obviously, $0 \leq \dim[\text{Inv}(h)] \leq n$. When the dimension of the invariant subspace $\text{Inv}(h)$ is positive, we call h *partially invariant*. Thus, every partially separable function is described as a sum of partially invariant functions. Again, the desirable and commonly occurring situation is that n_h is small with respect to n.

Because we are ultimately interested in sparsity for Hessian matrices, we restrict our attention to twice continuously differentiable partially invariant functions throughout the paper. We first emphasize that the Hessian matrix $\nabla_{xx}h(\boldsymbol{x})$ of a partially invariant function h is not always sparse. However, it is easy to see [8] that the invariant subspace $\text{Inv}(h)$ is contained in the null space of the Hessian matrix $\nabla_{xx}h(\boldsymbol{x})$, which is to say that

$$\nabla_{xx}h(\boldsymbol{x})\boldsymbol{w} = \boldsymbol{0} \text{ for every } \boldsymbol{w} \in \text{Inv}(h) \text{ and every } \boldsymbol{x} \in \mathbb{R}^n.$$

Let $N = \{1, \ldots, n\}$ and $\boldsymbol{e}_j$ denote the j-th unit coordinate column vector in $\mathbb{R}^n$ ($j \in N$). The case where $\boldsymbol{e}_j \in \text{Inv}(h)$ for some $j \in N$ is of particular interest in the following discussion, since $h : \mathbb{R}^n \to \mathbb{R}$ can then be represented as a function of the $n-1$ variables $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$, that is

$$h(\boldsymbol{x}) = h((x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n)^T) \text{ for every } \boldsymbol{x} \in \mathbb{R}^n.$$

Now define

$$K(h) = \{j \in N : \boldsymbol{e}_j \in \text{Inv}(h)\}. \tag{4}$$

Then, h is a function of the variables x_i for $i \in N \backslash K(h)$, and

$$\frac{\partial^2 h(\boldsymbol{x})}{\partial x_i x_j} = 0 \text{ for every } \boldsymbol{x} \in \mathbb{R}^n \text{ if } i \in K(h) \text{ or } j \in K(h),$$

which is to say that each $i \in K(h)$ makes all elements of the i-th column and i-th row of the Hessian matrix $\nabla_{xx}h(\boldsymbol{x})$ identically zero. The size of the set $K(h)$ therefore provides a measure of the amount of sparsity in the Hessian of h. The property just discussed may also be reformulated as

$$\begin{aligned} &\{(i,j) \in N \times N : i \in K(h) \text{ or } j \in K(h)\} \\ &\quad \subseteq \left\{(i,j) \in N \times N : \frac{\partial^2 h(\boldsymbol{x})}{\partial x_i x_j} = 0 \text{ for every } \boldsymbol{x} \in \mathbb{R}^n\right\}, \end{aligned}$$

where the latter set is, in general, larger than the former. For example, if $h : \mathbb{R}^n \to \mathbb{R}$ is a linear function of the form $h(\boldsymbol{x}) = \sum_{i=1}^n a_i x_i$ for some nonzero $a_i \in \mathbb{R}$ ($i \in N$), the

former set is empty since $K(h) = \emptyset$ while the latter set coincides with $N \times N$. Since the two sets are equivalent in many nonlinear functions and because the former set is often more important in some optimization methods, including in the sparse SDP relaxation for polynomial optimization problems (POPs), we concentrate on the former set in what follows.

We are now ready to state our objective. Consider a family of partially invariant C^2 functions $f_\ell : \mathbb{R}^n \to \mathbb{R}$ $(\ell \in M)$ where $M = \{1, 2, \ldots, m\}$ and let $\boldsymbol{P}$ be an $n \times n$ nonsingular matrix. Then, the family of transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ satisfies the properties that

$$\left.\begin{array}{l} g_\ell \text{ is a function of variables } z_i \ (i \in N \backslash K(g_\ell)) \text{ only} \\ \dfrac{\partial^2 g_\ell(\boldsymbol{z})}{\partial z_i z_j} = 0 \ (i \in K(g_\ell) \text{ or } j \in K(g_\ell)) \end{array}\right\} \ (\ell \in M). \tag{5}$$

The purpose of this paper is to propose a numerical method for finding an $n \times n$ nonsingular matrix $\boldsymbol{P}$ such that

$$\text{the sizes of all } K(g_\ell) \ (\ell \in M) \text{ are large evenly throughout } \ell \in M. \tag{6}$$

Note that this condition very often induces sparsity in the Hessian of the partially separable function constructed from the transformed functions $g_\ell(\boldsymbol{z})$, but not necessarily so, as is shown by the following (worst case) example: if we assume that M contains $n(n-1)/2$ indices and that corresponding sets $K(g_\ell)$ are given by

$$N \setminus \{1, 2\}, \ldots, N \setminus \{1, n\}, N \setminus \{2, 3\}, \ldots, N \setminus \{2, n\}, \ldots, N \setminus \{n-1, n\},$$

respectively, then every $K(g_\ell)$ is of size $n - 2$, yet the transformed Hessian is fully dense. This situation is however uncommon in practice, and (6) very often induces sparsity in the transformed Hessian, even if this sparsity pattern may not always be directly exploitable by all optimization algorithms.

We remark here that a partially separable function may admit several different decompositions into element functions. We are interested only by "maximal" partial separability structure in the sense of the paper [11], i.e., structures where the invariant subspaces are maximal (the invariant subspace of an element is never a subspace of that of another element). However, there are examples where even maximal structures are not unique. For our purpose, we give preference to structures where the dimensions of the associated subspaces do not vary too much in size.

In the unconstrained case, algorithms such as Newton's method or structured quasi-Newton methods [9, 24] are considerably more efficient when the Hessian of interest is sparse but also, whenever direct linear algebra methods are used to solve the linearized problem, when this Hessian admits a sparse Cholesky factorization. Our interest in methods of this type thus leads us to measure sparsity in terms of *correlative sparsity* [22] (briefly stated, a symmetric matrix is *correlatively sparse* when it can be decomposed into sparse Cholesky factors; see Section 2 for a more formal definition). The ideal goal would thus be to find a nonsingular linear transformation $\boldsymbol{P}$ such that the family of transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ $(\ell \in M)$ attains correlative sparsity. Unfortunately, achieving this goal for general problems is very complex and extremely expensive. We therefore settle for the more practically reasonable objective to propose a method that aims at the necessary condition

(6) in the hope of obtaining approximate correlative sparsity. This is also consistent with applications where the Cholesky factorization is not considered, but sparsity is nevertheless important, such as conjugate-gradient based algorithms. As an aside, we note that optimizing the sparsity pattern of the Hessian of a partially separable function has been attempted before (see [2]), but without the help of a nonsingular linear transformation in the problem space, which is the central tool in our approach.

In addition to the unconstrained optimization problems mentioned in the previous paragraph, the same technique can also be considered for uncovering correlative sparsity in the constrained case. More precisely, we then wish to find $\boldsymbol{P}$ such that the family of transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ ($\ell \in M$) satisfies condition (6) in a constrained optimization problem

$$\text{minimize } f_1(\boldsymbol{x}) \quad \text{subject to} \quad f_\ell(\boldsymbol{x}) \leq 0 \ \ (\ell \in M\backslash\{1\}). \tag{7}$$

Here, each f_ℓ is assumed to be partially invariant or partially separable. Because the sparsity of the Hessian matrices of the Lagrangian function and/or of some C^2 penalty or barrier function often determines the computational efficiency of many numerical methods for solving (7), our objective in this context is thus to improve the sparsity of the Hessians of these functions by applying a suitably chosen nonsingular linear transformation $\boldsymbol{P}$.

Returning to our motivating application, we may view a POP as a special case of the nonlinear optimization problem (7) in which all f_ℓ ($\ell \in M$) are polynomials in $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$. The SDP relaxation proposed by Lasserre [17] is known to be very powerful for solving POPs in theory, but so expensive that it can only be applied to small-sized instances (at most 30 variables, say). A sparse SDP relaxation for solving correlatively sparse POPs was proposed in Waki *et al* [22] to overcome this computational difficulty, and shown to be very effective for solving some larger-scale POPs. The use of this technique is also theoretically supported by the recent result by Lasserre [18] who shows convergence of sparse relaxation applied to correlatively sparse POPs. See also [14, 15, 16]. We should however point out that the sparse relaxation is known to be weaker than its dense counterpart, as it considers fewer constraints on the relaxed problem. As a result, the solution of the sparsely relaxed problem may be (and sometimes is, as we will discuss in Section 4) less accurate than if the (potentially impractical) dense relaxation were used. The method proposed in this paper nevertheless considers exploiting the practical advantages of sparse relaxation further by converting a given POP with partially invariant polynomial functions f_ℓ ($\ell \in M$) into a correlatively sparse one, therefore increasing the applicability of the technique. This is discussed further in Section 4.

We now comment briefly on the Cartesian sparse case, which was already studied in [11], where each f_ℓ is represented as in (1) for some $n_\ell \times n$ submatrix $\boldsymbol{A}_\ell$ of the $n \times n$ identity matrix. In this case, we see that, for each $\ell \in M$,

$$\text{Inv}(f_\ell) \ = \ \{\boldsymbol{w} \in \mathbb{R}^n : w_i = 0 \ (i \in N\backslash K(f_\ell))\}.$$

Thus, $\#K(f_\ell) = \dim[\text{Inv}(f_\ell)]$ for each ℓ, where $\#S$ denotes the cardinality of the set S. On the other hand, we know that for any $n \times n$ nonsingular matrix $\boldsymbol{P}$, the transformed function $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ is such that

$$\#K(g_\ell) = \#\{i \in N : \boldsymbol{e}^i \in \text{Inv}(g_\ell)\} \leq \dim[\text{Inv}(g_\ell)] = \dim[\text{Inv}(f_\ell)] = \#K(f_\ell),$$

where the penultimate equality follows from the identity $\mathrm{Inv}(g_\ell) = \boldsymbol{P}^{-1}\mathrm{Inv}(f_\ell)$, which is shown in Section 2.1. This indicates that the choice of $\boldsymbol{P}$ as the identity is the best to attain condition (6), and any further linear transformation is unnecessary.

The paper is organized as follows: Section 2 provides some preliminary material. We first discuss how a linear transformation in the problem space affects its invariant subspace, and then give a definition of correlative sparsity. An example is also presented for illustrating some basic definitions. Section 3 contains the description of a numerical method for finding a nonsingular linear transformation $\boldsymbol{z} \in \mathbb{R}^n \to \boldsymbol{P}\boldsymbol{z} \in \mathbb{R}^n$ such that the family of transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ $(\ell \in M)$ satisfies condition (6). The method consists of two algorithms: a heuristic greedy algorithm for a combinatorial optimization problem, which is formulated to maximize the sizes of $K(g_\ell)$ $(\ell \in M)$ lexicographically, and an algorithm for testing the feasibility of a candidate solution of the optimization problem. The latter algorithm is a probabilistic algorithm in the sense that the candidate solution is determined to be feasible or infeasible with probability one. In Section 4, we describe an application of the proposed method to POPs with some preliminary numerical results. Section 5 is finally devoted to concluding remarks and perspectives.

2 Preliminaries

2.1 Linear transformations in the problem space

We first examine how a linear transformation $\boldsymbol{z} \in \mathbb{R}^n \to \boldsymbol{x} = \boldsymbol{P}\boldsymbol{z} \in \mathbb{R}^n$ in the problem space affects the invariant subspace of a partially invariant C^2 function h and the sparsity of its Hessian matrix, where $\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)$ denotes an $n \times n$ nonsingular matrix. The main observation is that the transformed function $g(\boldsymbol{z}) = h(\boldsymbol{P}\boldsymbol{z})$ satisfies

$$g(\boldsymbol{z} + \boldsymbol{P}^{-1}\boldsymbol{w}) = h(\boldsymbol{P}(\boldsymbol{z} + \boldsymbol{P}^{-1}\boldsymbol{w})) = h(\boldsymbol{P}\boldsymbol{z} + \boldsymbol{w}) = h(\boldsymbol{P}\boldsymbol{z}) = g(\boldsymbol{z})$$

for every $\boldsymbol{z} \in \mathbb{R}^n$ and every $\boldsymbol{w} \in \mathrm{Inv}(h)$, which implies that

$$\mathrm{Inv}(g) = \boldsymbol{P}^{-1}\mathrm{Inv}(h). \tag{8}$$

This simply expresses that the invariant subspace is a geometric concept which does not depend on the problem space basis. Now suppose that some column vectors of $\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)$ are chosen from $\mathrm{Inv}(h)$. Then,

$$\begin{aligned} K(g) &= \{j \in N : \boldsymbol{e}_j \in \mathrm{Inv}(g)\} = \left\{j \in N : \boldsymbol{P}^{-1}\boldsymbol{p}_j \in \mathrm{Inv}(g)\right\} \\ &= \left\{j \in N : \boldsymbol{p}_j \in \mathrm{Inv}(h)\right\}. \end{aligned} \tag{9}$$

This means that g can be represented as a function of the variables z_i $(i \in N \backslash K(g))$ only. As a result, we obtain from (5) and (9) that *we can reduce the density (or increase the sparsity) of the Hessian matrix $\nabla_{zz} g(\boldsymbol{z})$ of the transformed function $g(\boldsymbol{z}) = h(\boldsymbol{P}\boldsymbol{z})$ by including more linearly independent vectors from the invariant subspace Inv(h) in the columns of $\boldsymbol{P}$.*

2.2 Correlative sparsity

In order to define what we mean by correlative sparsity, we follow [22] and introduce the *correlative sparsity pattern (csp) set* of the family of partially invariant C^2 functions g_ℓ

($\ell \in M$) as

$$E(g_\ell : \ell \in M) = \bigcup_{\ell \in M} (N \backslash K(g_\ell)) \times (N \backslash K(g_\ell)) \subset N \times N.$$

We know from (5) that

$$\frac{\partial^2 g_\ell(\boldsymbol{z})}{\partial z_i z_j} = 0 \ ((i,j) \notin E(g_\ell : \ell \in M)) \ \text{ for every } \boldsymbol{z} \in \mathbb{R}^n.$$

Hence $\#E(g_\ell : \ell \in M)$, the cardinality of the csp set $E(g_\ell : \ell \in M)$, measures the sparsity of the family of functions g_ℓ ($\ell \in M$): a family of partially invariant functions $g_\ell : \mathbb{R}^n \to \mathbb{R}$ ($\ell \in M$) is sparse if

$$\text{the cardinality } \#E(g_\ell : \ell \in M) \text{ of the csp set } E(g_\ell : \ell \in M) \text{ is small.} \tag{10}$$

We may assume without loss of generality that each $j \in N$ is not contained in some $K(g_\ell)$; otherwise some z_j is not involved in any g_ℓ ($\ell \in M$). Hence $(j,j) \in E(g_\ell : \ell \in M)$ for every $j \in M$, which indicates that $n \leq \#E(g_\ell : \ell \in M) \leq n^2$. We may thus consider that (10) holds if $\#E(g_\ell : \ell \in M)$ is of order n. This indicates that (10) is stronger than (6).

The correlative sparsity of a family of partially invariant C^2 functions $g_\ell : \mathbb{R}^n \to \mathbb{R}$ ($\ell \in M$) can then be defined in two ways. The first uses the *csp graph* $G(g_\ell : \ell \in M)$, which is defined as the undirected graph with node set N and edge set

$$E' = \{\{i,j\} : (i,j) \in E(g_\ell : \ell \in M), \ i < j\}.$$

(For simplicity of notation, we identify the edge set E' with $E(g_\ell : \ell \in M)$.) We then say that a family of partially invariant functions $g_\ell : \mathbb{R}^n \to \mathbb{R}$ ($\ell \in M$) is *correlatively sparse* if

$$\text{the csp graph } G(g_\ell : \ell \in M) \text{ has a sparse chordal extension.} \tag{11}$$

See [1] for the definition and some basic properties of chordal graphs. The second definition uses the *csp matrix* $\boldsymbol{R} = \boldsymbol{R}(g_\ell : \ell \in M)$ defined as

$$R_{ij} = \begin{cases} \star & \text{if } i = j \ \text{ or } (i,j) \in E(g_\ell : \ell \in M), \\ 0 & \text{otherwise.} \end{cases}$$

The family of partially invariant functions $g_\ell : \mathbb{R}^n \to \mathbb{R}$ ($\ell \in M$) is then said to be *correlatively sparse* if

$$\begin{array}{l} \text{the csp matrix } \boldsymbol{R} \text{ with a simultaneous reordering of its rows and} \\ \text{columns can be factored into the product of a sparse lower triangular} \\ \text{matrix and its transpose (the symbolic Cholesky factorization).} \end{array} \tag{12}$$

This last condition indicates that computing the Cholesky factor of the Hessian associated with a correlatively sparse family of invariant functions is inexpensive.

2.3 An illustrative example

Consider the partially separable function $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$\left.\begin{array}{rcl} f(\boldsymbol{x}) & = & \displaystyle\sum_{\ell=1}^{n+1} f_\ell(\boldsymbol{x}), \\ f_\ell(\boldsymbol{x}) & = & -x_\ell + x_\ell^2 \ (\ell = 1, \ldots, n) \ \text{ and } f_{n+1}(\boldsymbol{x}) = \left(\displaystyle\sum_{i=1}^{n} x_i \right)^4. \end{array}\right\} \tag{13}$$

Then a simple calculation shows that

$$\begin{array}{l} \mathrm{Inv}(f_\ell) = \{\boldsymbol{w} \in \mathbb{R}^n : w_\ell = 0\}, \ \dim[\mathrm{Inv}(f_\ell)] = n-1 \ (\ell = 1, \ldots, n), \\ \mathrm{Inv}(f_{n+1}) = \left\{ \boldsymbol{w} \in \mathbb{R}^n : \displaystyle\sum_{i=1}^{n} w_i = 0 \right\} \ \text{ and } \dim[\mathrm{Inv}(f_{n+1})] = n-1, \\ K(f_\ell) = \{1, \ldots, \ell-1, \ell+1, \ldots, n\} \ (\ell = 1, \ldots, n), \ K(f_{n+1}) = \emptyset. \end{array}$$

The fact that $K(f_{n+1})$ is empty makes the Hessian matrix $\nabla_{xx} f(\boldsymbol{x})$ fully dense, although f is partially separable. Increasing the size of $K(f_{n+1})$ is a key to finding a nonsingular linear transformation $\boldsymbol{P}$ that reduces the density of the Hessian matrix $\nabla_{zz} g(\boldsymbol{z})$ of the transformed function $g(\boldsymbol{z}) = f(\boldsymbol{P}\boldsymbol{z}) = \sum_{\ell=1}^{n+1} f_\ell(\boldsymbol{P}\boldsymbol{z})$. Let

$$\boldsymbol{P} \ = \ (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n), \quad \text{with} \quad \boldsymbol{p}_j = \boldsymbol{e}_j - \boldsymbol{e}_{j+1} \ (j = 1, \ldots, n-1), \quad \boldsymbol{p}_n = \boldsymbol{e}_n. \tag{14}$$

We then see that

$$\left.\begin{array}{l} \boldsymbol{p}_j \in \mathrm{Inv}(f_1) \ (j = 2, \ldots, n), \quad \boldsymbol{p}_j \in \mathrm{Inv}(f_2) \ (j = 3, \ldots, n), \\ \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell) \ (j = 1, \ldots, \ell-2, \ell+1, \ldots, n) \ (\ell = 3, \ldots, n-1), \\ \boldsymbol{p}_j \in \mathrm{Inv}(f_n) \ (j = 1, \ldots, n-2), \quad \boldsymbol{p}_j \in \mathrm{Inv}(f_{n+1}) \ (j = 1, \ldots, n-1). \end{array}\right\} \tag{15}$$

If we apply the nonsingular linear transformation $\boldsymbol{P}$, the transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ $(\ell = 1, \ldots, n+1)$ are such that

$$\left.\begin{array}{ll} K(g_1) = \{2, \ldots, n\}, & g_1 \text{ is a function of } z_1, \\ K(g_2) = \{3, \ldots, n\}, & g_2 \text{ is a function of } z_1 \text{ and } z_2, \\ K(g_\ell) = \{1, \ldots, \ell-2, \ell+1, \ldots, n\}, & g_\ell \text{ is a function of } z_{\ell-1} \text{ and } z_\ell \\ & (\ell = 3, \ldots, n-1), \\ K(g_n) = \{1, \ldots, n-2\}, & g_n \text{ is a function of } z_{n-1} \text{ and } z_n, \\ K(g_{n+1}) = \{1, \ldots, n-1\}, & g_{n+1} \text{ is a function of } z_n. \end{array}\right\} \tag{16}$$

Condition (6) therefore holds.

From the relations above, we also see that the csp set of the family of transformed functions g_ℓ $(\ell \in M)$ is given by $E(g_\ell : \ell \in M) = \{(i,j) \in N \times N : |i-j| \leq 1\}$. As a consequence, the csp matrix $\boldsymbol{R}(g_\ell : \ell \in M)$ and the Hessian matrix $\nabla_{zz} g(\boldsymbol{z})$ are tridiagonal, and their Cholesky factorization can therefore be performed without any fill-in. Consequently, the family of transformed functions g_ℓ $(\ell \in M)$ is correlatively sparse.

Rewriting the nonsingular linear transformation $\boldsymbol{x} = \boldsymbol{P}\boldsymbol{z}$ as $x_1 = z_1,\ x_i = z_i - z_{i-1}$ $(i = 2, \ldots, n)$ confirms the observation above, since we have that

$$\left.\begin{array}{rcl} g_1(\boldsymbol{z}) & = & -z_1 + z_1^2, \\ g_\ell(\boldsymbol{z}) & = & -(z_\ell - z_{\ell-1}) + (z_\ell - z_{\ell-1})^2 \ (\ell = 2, \ldots, n),\ g_{n+1}(\boldsymbol{z}) = z_n^4, \\ g(\boldsymbol{z}) & = & \displaystyle\sum_{\ell=1}^{n+1} g_\ell(\boldsymbol{z}) = -z_n + \sum_{i=1}^{n-1} \left(2z_i^2 - 2z_i z_{i+1}\right) + z_n^2 + z_n^4. \end{array}\right\} \tag{17}$$

Because of its tridiagonal Hessian, the transformed function g can clearly be more efficiently minimized by Newton's method for minimization than the original function f.

2.4 Computation of the invariant subspace of a polynomial function

If a basis of the invariant subspace $\mathrm{Inv}(h)$ of a partially invariant function $h : \mathbb{R}^n \to \mathbb{R}$ is given in advance, we can easily derive an $n_h \times n$ matrix $\boldsymbol{A}$ and a function $\hat{h} : \mathbb{R}^{n_h} \to \mathbb{R}$ for which (3) holds. Now, we assume that neither any basis of the invariant subspace $\mathrm{Inv}(h)$ of a polynomial function $h : \mathbb{R}^n \to \mathbb{R}$ nor any representation of the form (3) for h is known. We briefly discuss how we decide whether h is partially invariant and how the invariant subspace $\mathrm{Inv}(h)$ is computed. By definition, we can characterize each $\boldsymbol{w} \in \mathrm{Inv}(h)$ as

$$0 \ = \ h(\boldsymbol{x} + \lambda \boldsymbol{w}) - h(\boldsymbol{x}) \text{ for every } \lambda \in \mathbb{R} \text{ and } \boldsymbol{x} \in \mathbb{R}^n.$$

Then we know that the right-hand side of the identity is a polynomial in $(\boldsymbol{x}, \lambda) \in \mathbb{R}^{n+1}$ with coefficients polynomial in $\boldsymbol{w} \in \mathbb{R}^n$. Since the polynomial in $(\boldsymbol{x}, \lambda) \in \mathbb{R}^{n+1}$ is identically zero, the polynomials in $\boldsymbol{w} \in \mathbb{R}^n$ from the coefficients must vanish. We thus obtain a system of polynomial equations in $\boldsymbol{w} \in \mathbb{R}^n$ which determine $\mathrm{Inv}(h)$.

We illustrate the method presented above for a polynomial $h(\boldsymbol{x})$ in $\boldsymbol{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$ of the form

$$h(\boldsymbol{x}) \ = \ (-3x_1 + 2x_2 + 2x_3 - 5)x_1 + (x_2 + 2x_3 - 3)x_2 + (x_3 - 3)x_3 + 3.$$

Then

$$\begin{array}{rcl} h(\boldsymbol{x} + \lambda \boldsymbol{w}) - \boldsymbol{h}(\boldsymbol{x}) & = & -(5w_1 + 3w_2 + 3w_3)\lambda + 2(-3w_1 + w_2 + w_3)\lambda x_1 \\ & & +(2w_1 + 2w_2 + 2w_3)\lambda x_2 + (2w_1 + 2w_2 + 2w_3)\lambda x_3 \\ & & +(-3w_1^2 + 2w_1 w_2 + w_2^2 + 2w_1 w_3 + 2w_2 w_3 + w_3^2)\lambda^2 \\ & & \text{for every } \lambda \in \mathbb{R} \text{ and } \boldsymbol{x} \in \mathbb{R}^3. \end{array}$$

Hence we obtain a system of polynomial equations

$$\begin{array}{rcl} 0 & = & 5w_1 + 3w_2 + 3w_3,\ 0 = -3w_1 + w_2 + w_3, \\ 0 & = & 2w_1 + 2w_2 + 2w_3,\ 0 = -3w_1^2 + 2w_1 w_2 + w_2^2 + 2w_1 w_3 + 2w_2 w_3 + w_3^2. \end{array}$$

Solving this system of equations provides $w_1 = 0$ and $w_3 = -w_2$. Thus $\mathrm{Inv}(h)$ turns out to be a 1-dimensional subspace generated by $(0, 1, -1) \in \mathbb{R}^3$.

3 A numerical method for improving sparsity

Throughout this section we consider a family of partially invariant C^2 functions $f_\ell : \mathbb{R}^n \to \mathbb{R}$ ($\ell \in M$). For every index set $S \subseteq M$, we define the subspace

$$\mathrm{Inv}[S] = \bigcap_{\ell \in S} \mathrm{Inv}(f_\ell),$$

and its dimension $\delta(S)$. (Notice the square bracket notation in $\mathrm{Inv}[\cdot]$ indicating that its argument is an index set of partially invariants functions.) $\mathrm{Inv}[S]$ is the intersection of the invariant subspaces over all partially invariant functions f_ℓ whose index ℓ is in S, and each $\boldsymbol{w} \in \mathrm{Inv}[S]$ is thus an invariant direction for this particular collection of partially invariant functions. In addition, $\mathrm{Inv}[\emptyset] = \mathbb{R}^n$, $\delta(\emptyset) = n$ and $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ are a basis of $\mathrm{Inv}[\emptyset]$. In the following discussion, the problem of finding an $n \times n$ nonsingular matrix $\boldsymbol{P}$ such that the family of transformed functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ ($\ell \in M$) satisfies condition (6) is reformulated as a problem of choosing a basis $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ of $\mathbb{R}^n$ from $\mathrm{Inv}[S_1], \ \ldots \ , \mathrm{Inv}[S_n]$ for some family of index sets $S_1, \ldots, S_n \subseteq M$. For simplicity, we use the notation $\boldsymbol{S} = (S_1, \ldots, S_n) \subseteq M^n$ in what follows.

We organize the discussion by first describing the feasible set for our problem, that is which $\boldsymbol{S}$ are admissible. We then motivate its reformulation as a combinatorial maximization problem, and finally outline an algorithm for its solution.

3.1 Feasibility

In order to describe the feasible set for our maximization problem, we consider the following combinatorial condition on $\boldsymbol{S}$:

$$\begin{aligned} \mathcal{F}(n): \quad & \text{there exists a set of linearly independent vectors } \boldsymbol{p}_j \ (j = 1, \ldots, n) \\ & \text{such that, for all } j, \ S_j = \{\ell \in M : \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell)\}. \end{aligned}$$

We immediately note that $\boldsymbol{S} = \emptyset^n$ (*i.e.*, $S_j = \emptyset$ ($j \in N$)) satisfies this condition. Indeed, we have to find a basis $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ of $\mathbb{R}^n$ such that none of these vectors belong to any invariant subspace $\mathrm{Inv}(f_\ell)$ ($\ell \in M$), which is clearly possible because the union of all these invariant subspaces is of measure zero in $\mathbb{R}^n$.

For any $\boldsymbol{S}$, now define

$$L_\ell(\boldsymbol{S}) = \{j \in N : \ell \in S_j\} \quad (\ell \in M), \tag{18}$$

which identifies the particular collection of index sets S_j that contain ℓ. Then, obviously, if $\boldsymbol{S}$ satisfies $\mathcal{F}(n)$ with $\boldsymbol{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$,

$$L_\ell(\boldsymbol{S}) = \big\{j \in N : \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell)\big\} = K(g_\ell), \tag{19}$$

where the last equality follows from (9). Combining this identity with (5), we thus deduce that

$$\left.\begin{aligned} & L_\ell(\boldsymbol{S}) = K(g_\ell), \\ & g_\ell \text{ is a function of variables } z_i \ (i \in N \backslash L_\ell(\boldsymbol{S})) \text{ only}, \\ & \frac{\partial^2 g_\ell(\boldsymbol{z})}{\partial z_i z_j} = 0 \ (i \in L_\ell(\boldsymbol{S}) \ \text{ or } j \in L_\ell(\boldsymbol{S})) \end{aligned}\right\} \ (\ell \in M). \tag{20}$$

We illustrate these concepts with the example of Section 2.3, where we let

$$\left.\begin{array}{lcl} M & = & \{1,\ldots,n+1\}, \\ S_1 & = & \{3,4,\ldots,n,n+1\},\ S_2=\{1,3,4,\ldots,n,n+1\}, \\ S_j & = & \{1,\ldots,j-1,j+2,\ldots,n,n+1\}\ (j=3,\ldots,n-2), \\ S_{n-1} & = & \{1,2,\ldots,n-2,n+1\},\ S_n=\{1,\ldots,n-1\}. \end{array}\right\} \tag{21}$$

Then, $\boldsymbol{S}$ satisfies condition $\mathcal{F}(n)$ with $\boldsymbol{p}_j$ $(j \in N)$ given in (14). We then see from (15) that

$$\left.\begin{array}{l} L_1(\boldsymbol{S})=\{2,\ldots,n\},\ L_2(\boldsymbol{S})=\{3,\ldots,n\}, \\ L_\ell(\boldsymbol{S})=\{1,\ldots,\ell-2,\ell+1,\ldots,n\}\ (\ell=3,\ldots,n-1), \\ L_n(\boldsymbol{S})=\{1,\ldots,n-2\},\ L_{n+1}(\boldsymbol{S})=\{1,\ldots,n-1\}, \end{array}\right\} \tag{22}$$

which coincide with the $K(g_\ell)$ $(\ell = 1,\ldots,n+1)$ given in (16), respectively. Recall that the nonsingular linear transformation $\boldsymbol{P}$ in the problem space of the partially invariant functions f_ℓ $(\ell \in M)$ given in (13) yields the transformed functions g_ℓ given in (17). We easily verify that the relations (20) hold.

3.2 The lexicographic maximization problem

In view of the discussion above, we may replace the requirement (6) on the nonsingular linear transformation $\boldsymbol{P}$ by the following: for an $\boldsymbol{S}$ satisfying condition $\mathcal{F}(n)$,

$$\text{the sizes of } L_\ell(\boldsymbol{S})\ (\ell \in M) \text{ are large evenly throughout } \ell \in M. \tag{23}$$

We now formulate an optimization problem whose solutions will achieve this requirement.

For every $r \in N$ and every $(S_1,\ldots,S_r) \subseteq M^r$, define, in a manner similar to (18),

$$L_\ell(S_1,\ldots,S_r)=\{j \in \{1,\ldots,r\} : \ell \in S_j\} \quad (\ell \in M) \tag{24}$$

and also

$$\boldsymbol{\sigma}(S_1,\ldots,S_r)=(\#L_{\pi(1)}(S_1,\ldots,S_r),\ldots,\#L_{\pi(m)}(S_1,\ldots,S_r)), \tag{25}$$

which is an m-dimensional vector where $(\pi(1),\ldots,\pi(m))$ denotes a permutation of $(1,\ldots,m)$ such that

$$\#L_{\pi(1)}(S_1,\ldots,S_r) \le \cdots \le \#L_{\pi(m)}(S_1,\ldots,S_r).$$

Each component of $\boldsymbol{\sigma}(S_1,\ldots,S_r)$ is therefore associated with a partially invariant function and gives the number of linearly independent vectors $\boldsymbol{p}_j$ $(j=1,\ldots,r)$ invariant for this function for any choice of these vectors associated to $S_1,\ldots,S_r$ by $\mathcal{F}(r)$, these numbers being sorted in increasing order. We therefore aim at finding a $\boldsymbol{\sigma}(S_1,\ldots,S_r)$ with uniformly large components, if possible.

To illustrate the definition of the sets $L_\ell(S_1,\ldots,S_r)$ and of $\boldsymbol{\sigma}(S_1,\ldots,S_r)$, we return once more to the example of Section 2.3. If $r=1$ and $S_1=\{3,4,\ldots,n+1\}$, then

$$\left.\begin{array}{l} L_1(S_1)=L_2(S_1)=\emptyset \text{ and } L_\ell(S_1)=\{1\}\ (\ell=3,\ldots,n+1), \\ \#L_1(S_1)=\#L_2(S_1)=0, \quad \#L_\ell(S_1)=1 \quad (\ell=3,\ldots,n+1), \\ (\pi(1),\pi(2),\ldots,\pi(n+1))=(1,2,\ldots,n+1), \\ \boldsymbol{\sigma}(S_1)=(\#L_1(S_1),\#L_2(S_1),\ldots,\#L_{n+1}(S_1))=(0,0,1,\ldots,1). \end{array}\right\} \tag{26}$$

If, on the other hand, $r = 2$, $S_1 = \{3, 4, \ldots, n+1\}$ and $S_2 = \{1\}$, then

$$\left.\begin{array}{l} L_1(S_1, \{1\}) = \{2\}, \quad L_2(S_1, \{1\}) = \emptyset \text{ and } L_\ell(S_1, \{1\}) = \{1\} \quad (\ell = 3, \ldots, n+1), \\ \#L_1(S_1, \{1\}) = 1, \quad \#L_2(S_1, \{1\}) = 0, \quad \#L_\ell(S_1, \{1\}) = 1 \quad (\ell = 3, \ldots, n+1), \\ (\pi(1), \pi(2), \ldots, \pi(n+1)) = (2, 1, \ldots, n+1), \\ \boldsymbol{\sigma}(S_1, \{1\}) = (\#L_2(S_1, \{1\}), \#L_1(S_1, \{1\}), \ldots, \#L_{n+1}(S_1, \{1\})) = (0, 1, 1, \ldots, 1). \end{array}\right\} \quad (27)$$

Finally, when $r = n$ and $\boldsymbol{S}$ is given by (21), we have observed that (22) holds. It then follows that

$$\left.\begin{array}{l} \#L_1(\boldsymbol{S}) = \#L_{n+1}(\boldsymbol{S}) = n-1, \ \#L_\ell(\boldsymbol{S}) = n-2 \quad (\ell = 2, \ldots, n), \\ (\pi(1), \ldots, \pi(n), \pi(n+1)) = (2, \ldots, n, 1, n+1), \\ \boldsymbol{\sigma}(\boldsymbol{S}) = (\#L_2(\boldsymbol{S}), , \ldots, \#L_n(\boldsymbol{S}), \#L_1(\boldsymbol{S}), \#L_{n+1}(\boldsymbol{S})) \\ \qquad = (n-2, \ldots, n-2, n-1, n-1). \end{array}\right\} \quad (28)$$

Now suppose that $(S_1^1, \ldots, S_{r_1}^1) \subseteq M^{r_1}$ and $(S_1^2, \ldots, S_{r_2}^2) \subseteq M^{r_2}$ satisfy $\mathcal{F}(r_1)$ and $\mathcal{F}(r_2)$, respectively. Then, we say that $\boldsymbol{\sigma}(S_1^1, \ldots, S_{r_1}^1)$ is lexicographically larger than $\boldsymbol{\sigma}(S_1^2, \ldots, S_{r_2}^2)$ if

$$\begin{array}{l} \sigma_\ell(S_1^1, \ldots, S_{r_1}^1) = \sigma_\ell(S_1^2, \ldots, S_{r_2}^2) \ (\ell = 1, \ldots, k-1) \ \text{ and} \\ \sigma_k(S_1^1, \ldots, S_{r_1}^1) > \sigma_k(S_1^2, \ldots, S_{r_2}^2) \end{array}$$

for some $k \in M$. Recall that, because of $\mathcal{F}(r)$, each component ℓ of $\boldsymbol{\sigma}(S_1, \ldots, S_r)$ gives the numbers $\#L_\ell(S_1, \ldots, S_r)$ of linearly independent vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, r)$ which are invariant directions for f_ℓ $(\ell \in M)$, these components appearing in $\boldsymbol{\sigma}(S_1, \ldots, S_r)$ in increasing order. Hence $\boldsymbol{\sigma}(S_1^1, \ldots, S_{r_1}^1)$ (or $\boldsymbol{p}_j^1$ $(j = 1, \ldots, r)$) is preferable to $\boldsymbol{\sigma}(S_1^2, \ldots, S_{r_2}^2)$ (or $\boldsymbol{p}_j^2$ $(j = 1, \ldots, r)$) for our criterion (23). (Comparing $\boldsymbol{\sigma}(S_1)$, $\boldsymbol{\sigma}(S_1, \{1\})$ and $\boldsymbol{\sigma}(\boldsymbol{S})$, respectively given in (26), (27) and (28), we see that $\boldsymbol{\sigma}(S_1, \{1\})$ is lexicographically larger than $\boldsymbol{\sigma}(S_1)$, and $\boldsymbol{\sigma}(\boldsymbol{S})$ is lexicographically largest among the three.)

It is thus meaningful, in view of our objective (23), to find a set vector $\boldsymbol{S}$ that makes $\boldsymbol{\sigma}(\boldsymbol{S})$ lexicographically as large as possible. As a consequence, finding good solutions of the optimization problem

$$\begin{array}{ll} \mathcal{P}(n): & \text{lexicographically maximize } \boldsymbol{\sigma}(\boldsymbol{S}) \\ & \text{by choosing } \boldsymbol{S} \text{ subject to condition } \mathcal{F}(n) \end{array}$$

is of direct interest.

3.3 A combinatorial algorithm

We now consider a heuristic greedy algorithm to (approximately) solve problem $\mathcal{P}(n)$. The main idea is to consider a family of subproblems of $\mathcal{P}(n)$, defined, for $r = 1, \ldots, n$, by

$$\begin{array}{ll} \mathcal{P}(r): & \text{lexicographically maximize } \boldsymbol{\sigma}(S_1, \ldots, S_r) \\ & \text{by choosing } (S_1, ..., S_r) \text{ subject to condition } \mathcal{F}(r). \end{array}$$

Having introduced all the necessary ingredients, we are now in position to provide a first motivating description of our algorithm.

1. We start with $r = 1$, $S_j = \emptyset$ $(j \in N)$ and $L_\ell(S_1) = \emptyset$ $(\ell \in M)$, where r is the outer-loop iterations counter. Suppose that $r = 1$ or that a $(S_1, \ldots, S_{r-1})$ satisfying $\mathcal{F}(r-1)$ and the corresponding $L_\ell(S_1, \ldots, S_{r-1})$ have been determined in iterations $1, \ldots, r-1$ with $2 \le r \le n$. At the r-th iteration, we first compute a permutation $(\pi(1), \ldots, \pi(m))$ of $(1, \ldots, m)$ such that

$$\#L_{\pi(1)}(S_1, \ldots, S_r) \le \cdots \le \#L_{\pi(m)}(S_1, \ldots, S_r)$$

with $S_r = \emptyset$. Thus $(S_1, \ldots, S_{r-1}, S_r)$ is a feasible solution of $\mathcal{P}(r)$ with the objective value

$$\boldsymbol{\sigma}(S_1, \ldots, S_r) = (\#L_{\pi(1)}(S_1, \ldots, S_r), \ldots, \#L_{\pi(m)}(S_1, \ldots, S_r)).$$

2. We then attempt to generate lexicographically larger feasible set vectors for $\mathcal{P}(r)$ by adding $\pi(k)$ to S_r, for $k = 1, \ldots, m$, each time enlarging S_r provided $(S_1, \ldots, S_r)$ satisfies the condition

$$\begin{aligned} \mathcal{F}_w(r): \quad & \text{there exists a set of linearly independent vectors } \boldsymbol{p}_j \ (j = 1, \ldots, r) \\ & \text{such that } \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell : \ell \in S_j) \ (j = 1, \ldots, r), \end{aligned}$$

which is a weaker version of condition $\mathcal{F}(r)$.

For example, suppose that we have $\boldsymbol{\sigma}(S_1) = (0, 0, 1, \ldots, 1)$ with $S_1 = \{3, 4, \ldots, n, n+1\}$ as shown in (26). Let $r = 2$ and $S_2 = \emptyset$. Then, $\boldsymbol{\sigma}(S_1, S_2) = \boldsymbol{\sigma}(S_1) = (0, 0, 1, \ldots, 1)$. In order to increase $\boldsymbol{\sigma}(S_1, S_2)$ lexicographically, we first try to add $\pi(1) = 1$ to $S_2 = \emptyset$, then the resulting $\boldsymbol{\sigma}(S_1, \{1\}) = (0, 1, 1, \ldots, 1)$ as shown in (27) would become lexicographically larger than $\boldsymbol{\sigma}(S_1)$. Note that we could choose $\pi(2) = 2$ instead of $\pi(1) = 1$ since

$$\#L_1(S_1, \emptyset) = \#L_2(S_1, \emptyset) = 0 < \#L_\ell(S_1, \emptyset) = 1 \quad (\ell > 2),$$

but the choice of any other index $\ell > 2$ would result in $\boldsymbol{\sigma}(S_1, \{\ell\}) = (0, 0, 1, \ldots, 1, 2)$, which is lexicographically smaller than $\boldsymbol{\sigma}(S_1, \{1\}) = \boldsymbol{\sigma}(S_1, \{2\}) = (0, 1, 1, \ldots, 1)$. Therefore, the index $\pi(1)$ is the best first choice among $M = \{1, \ldots, n+1\}$ for lexicographically increasing $\boldsymbol{\sigma}(S_1, S_2)$, which is why we include it in S_2 first, before trying to include $\pi(2), \pi(3), \ldots, \pi(m)$.

3. For each $k = 1, \ldots, m$, we then update $(S_1, \ldots, S_r)$ with fixing $(S_1, \ldots, S_{r-1})$ and choosing

$$S_r = \begin{cases} S_r \cup \{\pi(k)\} & \text{if } (S_1, \ldots, S_{r-1}, S_r \cup \{\pi(k)\}) \text{ satisfies } \mathcal{F}_w(r), \\ S_r & \text{otherwise.} \end{cases} \tag{29}$$

If S_r is augmented in (29), we also update

$$L_{\pi(k)}(S_1, \ldots, S_r) = L_{\pi(k)}(S_1, \ldots, S_r) \cup \{r\}$$

accordingly.

4. At the end of this inner loop (i.e., for $k = n$), we have computed a set vector $(S_1, \ldots, S_r)$ which satisfies $\mathcal{F}_w(r)$ by construction, as well as an associated set of linearly independent vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_r$. We prove below that it also satisfies $\mathcal{F}(r)$, and is hence feasible for problem $\mathcal{P}(r)$.

5. Note that $(S_1, \ldots, S_n)$ with $S_\ell = \emptyset$ $(\ell = r+1, \ldots, n)$ is a feasible solution of problem $\mathcal{P}(n)$. If $S_r = \emptyset$, we know that there is no feasible solution $\boldsymbol{S}' = (S'_1, \ldots, S'_n)$ of $\mathcal{P}(n)$ satisfying $(S'_1, \ldots, S'_{r-1}) = (S_1, \ldots, S_{r-1})$ except the feasible solution $(S_1, \ldots, S_n)$ just computed; hence $(S_1, \ldots, S_n)$ is the best greedy feasible solution of problem $\mathcal{P}(n)$ and we terminate the iteration. If $r = n$, we have obtained the best greedy feasible solution of problem $\mathcal{P}(n)$, and we also terminate the iteration. Otherwise, the $(r+1)$-th outer iteration is continued.

For making the above description coherent, we still need to prove the result announced in item 4.

Theorem 3.1. *Assume that the algorithm described above produces the set vector $(S_1, \ldots, S_r)$ at the end of inner iteration r. Then $(S_1, \ldots, S_r)$ is feasible for problem $\mathcal{P}(r)$.*

Proof: By construction, we know that $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$ for a set of linearly independent vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_r$. We will show that

$$S_j = \{\ell \in M : \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell)\} \quad (j = 1, \ldots, r), \tag{30}$$

which implies that $\mathcal{F}(r)$ holds with these $\boldsymbol{p}_j$ $(j = 1, \ldots, r)$ and therefore that $(S_1, \ldots, S_r)$ is feasible for problem $\mathcal{P}(r)$, as desired. First note that the inclusion

$$\boldsymbol{p}_j \in \mathrm{Inv}[S_j] \quad (j = 1, \ldots, r) \tag{31}$$

(which is ensured by property $\mathcal{F}_w(r)$) implies that $S_j \subseteq \{\ell \in M : \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell)\}$ for $j = 1, \ldots, r$. We now prove that the reverse inclusion holds. Assume, on the contrary, that $S_q \not\supseteq \{\ell \in M : \boldsymbol{p}_q \in \mathrm{Inv}(f_\ell)\}$ for some $q \in \{1, \ldots, r\}$. Then there exists an $\ell \in M$ such that $\ell \notin S_q$ and $\boldsymbol{p}_q \in \mathrm{Inv}(f_\ell)$. We now recall the mechanism of the q-th outer iteration. We first set $S_q = \emptyset$ and computed a permutation π such that $\#L_{\pi(1)}(S_1, \ldots, S_q) \leq \cdots \leq \#L_{\pi(m)}(S_1, \ldots, S_q)$. Because $(\pi(1), \ldots, \pi(m))$ is a permutation of $(1, \ldots, m)$, there is a unique k such that $\ell = \pi(k)$, and we thus have that

$$\pi(k) \notin S_q \quad \text{and} \quad \boldsymbol{p}_q \in \mathrm{Inv}(f_{\pi(k)}). \tag{32}$$

Let us focus our attention on the $(k-1)$-th and k-th inner iterations of the q-th outer iteration. At inner iteration $(k-1)$, the set $S_q^{k-1} = \{\pi(s) \in S_q : 1 \leq s \leq k-1\}$ must have been generated since S_q^{k-1} is a subset of S_q and is expanded to S_q as the inner iteration proceeds[1]. We then updated S_q^{k-1} to S_q^k according to (29) and depending on whether there exists a set of linearly independent $\boldsymbol{p}_j^k$ $(j = 1, \ldots, q)$, say, such that $(S_1, \ldots, S_{q-1}, S_q^{k-1} \cup \{\pi(k)\})$ satisfies $\mathcal{F}_w(q)$ with these vectors, that is

$$\begin{aligned} & \boldsymbol{p}_j^k \in \mathrm{Inv}[S_j] \quad \text{for } j = 1, \ldots, q-1 \\ \text{and} \quad & \boldsymbol{p}_q^k \in \mathrm{Inv}[S_q^{k-1} \cup \{\pi(k)\}] = \mathrm{Inv}[S_q^{k-1}] \cap \mathrm{Inv}(f_{\pi(k)}). \end{aligned}$$

Now observe that the vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, q)$ satisfy these conditions. They are indeed linearly independent, as we noted above, and

$$\begin{aligned} \boldsymbol{p}_j &\in \mathrm{Inv}[S_j] \text{ for } j = 1, \ldots, q-1 && \text{(by (31))}, \\ \boldsymbol{p}_q &\in \mathrm{Inv}[S_q] \subseteq \mathrm{Inv}[S_q^{k-1}] && \text{(by (31) and } S_q^{k-1} \subseteq S_q), \\ \boldsymbol{p}_q &\in \mathrm{Inv}(f_{\pi(k)}) && \text{(by the second relation of (32))}. \end{aligned}$$

[1]Here S_q^{k-1} denotes the value of the set S_q at the end of the $(k-1)$-th inner iteration.

The existence of suitable vectors $\boldsymbol{p}_j^k$ $(j = 1, \ldots, q)$ is therefore guaranteed, giving that $(S_1, \ldots, S_{r-1}, S_q^{k-1} \cup \{\pi(k)\})$ satisfies $\mathcal{F}_w(q)$ with the $\boldsymbol{p}_j^k = \boldsymbol{p}_j$ $(j = 1, \ldots, q)$. Thus S_q^{k-1} must have been updated to $S_q^k = S_q^{k-1} \cup \{\pi(k)\}$ in (29) and, as a consequence, $\pi(k) \in S_q^k \subseteq S_q$, which yields the desired contradiction with the first relation of (32). ∎

Observe that, while this theorem shows that $(S_1, \ldots, S_r)$ is feasible for problem $\mathcal{P}(r)$ at the end of the inner iteration, its proof indicates why it may not be the case before this inner iteration is completed.

Of course, for our approach to be practical, we still need to check property $\mathcal{F}_w(r)$ for problem $\mathcal{P}(r)$ in item 2 of our algorithmic outline. This is the object of the next subsection.

3.4 A probabilistic method for checking $\mathcal{F}_w(r)$

The main idea behind our method for checking $\mathcal{F}_w(r)$ is that a random linear combination of basis vectors almost surely does not belong to a proper subspace. To express this more formally, consider any $S \subseteq M$ and let $\boldsymbol{b}(S)_1, \ldots, \boldsymbol{b}(S)_{\delta(S)} \in \mathbb{R}^n$ denote a basis of $\mathrm{Inv}[S]$. Each vector in $\mathrm{Inv}[S]$ can then be represented as a linear combination of this basis. If we use the notation

$$\mathcal{E}_r = \prod_{k=1}^{r} \mathbb{R}^{\delta(S_k)} \quad \text{and} \quad \boldsymbol{p}_S(\boldsymbol{\alpha}) = \sum_{i=1}^{\delta(S)} \alpha_i \boldsymbol{b}(S)_i$$

for every $S \subseteq M$ and every $\boldsymbol{\alpha} \in \mathbb{R}^{\delta(S)}$, we may then prove the following geometric result.

Theorem 3.2. *Assume that $(S_1, \ldots, S_r) \subseteq M^r$ satisfies $\mathcal{F}_w(r)$. Then the set*

$$\Lambda(S_1, \ldots, S_r) = \left\{ (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_r) \in \mathcal{E}_r : \begin{array}{l} \boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_r}(\boldsymbol{\alpha}_r) \\ \textit{are linearly independent} \end{array} \right\} \tag{33}$$

is open and dense in $\mathcal{E}_r$.

Proof: Let $(\bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_r) \in \Lambda(S_1, \ldots, S_r)$, which is nonempty by assumption. Then, $\boldsymbol{p}_{S_1}(\bar{\boldsymbol{\alpha}}_1), \ldots, \boldsymbol{p}_{S_r}(\bar{\boldsymbol{\alpha}}_r)$ are linearly independent, and the $n \times r$ matrix $\big(\boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_r}(\boldsymbol{\alpha}_r)\big)$ contains an $r \times r$ submatrix $\boldsymbol{V}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_r)$, say, which is nonsingular at $(\bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_r)$. By continuity of the determinant of $\boldsymbol{V}(\cdot)$ with respect to its arguments, it remains nonsingular, and hence $\boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_r}(\boldsymbol{\alpha}_r)$ are linearly independent, in an open neighborhood of $(\bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_r)$. We have thus proved that $\Lambda(S_1, \ldots, S_r)$ is an open subset of $\mathcal{E}_r$. To prove that it is dense in this space, let $(\hat{\boldsymbol{\alpha}}_1, \ldots, \hat{\boldsymbol{\alpha}}_r)$ be an arbitrary point in $\mathcal{E}_r$. We show that, in any open neighborhood of this point, there is a $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_r)$ such that $\boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_r}(\boldsymbol{\alpha}_r)$ are linearly independent. Let $(\bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_r) \in \Lambda(S_1, \ldots, S_r)$. As discussed above, we assume that an $r \times r$ submatrix $\boldsymbol{V}(\bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_r)$ of $\big(\boldsymbol{p}_{S_1}(\bar{\boldsymbol{\alpha}}_1), \ldots, \boldsymbol{p}_{S_r}(\bar{\boldsymbol{\alpha}}_r)\big)$ is nonsingular. For every $t \in \mathbb{R}$, let

$$\phi(t) \;=\; \det\left[\boldsymbol{V}((1-t)\hat{\boldsymbol{\alpha}}_1 + t\bar{\boldsymbol{\alpha}}_1, \ldots, (1-t)\hat{\boldsymbol{\alpha}}_r + t\bar{\boldsymbol{\alpha}}_r)\right].$$

Then, $\phi(t)$ is a polynomial which is not identically zero because $\phi(1) \neq 0$. Hence $\phi(t) = 0$ at most a finite number of t's, so that $\phi(\epsilon) \neq 0$ for every sufficiently small positive ϵ. Therefore the vectors

$$\boldsymbol{p}_{S_1}((1-\epsilon)\hat{\boldsymbol{\alpha}}_1 + \epsilon\bar{\boldsymbol{\alpha}}_1), \ldots, \boldsymbol{p}_{S_r}((1-\epsilon)\hat{\boldsymbol{\alpha}}_r + \epsilon\bar{\boldsymbol{\alpha}}_r)$$

are linearly independent for every sufficiently small positive ϵ. ∎

The main interest of Theorem 3.2 is that it provides a (probabilistic) way to test whether a given $(S_1, \ldots, S_r) \subseteq M^r$ does not satisfy $\mathcal{F}_w(r)$. Consider indeed a random choice of $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_r) \in \mathcal{E}_r$ and compute

$$\boldsymbol{p}_{S_j}(\boldsymbol{\alpha}_j) = \sum_{i=1}^{\delta(S_j)} (\alpha_j)_i \boldsymbol{b}(S_j)_i \in \mathrm{Inv}[S_j] \ (j = 1, \ldots, r).$$

Then, the vectors $\boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_r}(\boldsymbol{\alpha}_r)$ are almost surely linearly independent whenever $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$. Thus, if these vectors turn out to be linearly dependent, $\mathcal{F}_w(r)$ almost surely fails. These observations are embodied in the following algorithm.

Algorithm 3.3.

Step 1. Compute a basis $\boldsymbol{b}(S_j)_1, \ldots, \boldsymbol{b}(S_j)_{\delta(S_j)}$ of $\mathrm{Inv}[S_j]$ for $(j = 1, \ldots, r)$.

Step 2. For $j = 1, \ldots, r$, randomly choose a vector $\boldsymbol{\alpha}_j$ from a uniform distribution over the box

$$B_j = [-1, 1]^{\delta(S_j)}. \tag{34}$$

and compute the vector $\boldsymbol{p}_{S_j}(\boldsymbol{\alpha}_j) = \sum_{i=1}^{\delta(S_j)} (\alpha_j)_i \boldsymbol{b}(S_j)_i$.

Step 3. Check if the computed $\boldsymbol{p}_{S_j}(\boldsymbol{\alpha}_j)$ $(j = 1, 2, \ldots, r)$ are linearly independent. If this is the case, $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$ by definition. Otherwise, the decision that $(S_1, \ldots, S_r)$ does not satisfy this property is correct with probability one.

Observe that, when $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$, the above algorithm provides an associated set of linearly independent vectors $\boldsymbol{p}_j = \boldsymbol{p}_{S_j}(\boldsymbol{\alpha}_j)$ $(j = 1, \ldots, r)$ as a by-product. At the end of the algorithm outlined in the previous paragraph, this property and Theorem 3.1 thus imply that the desired nonsingular matrix $\boldsymbol{P}$ is given by $\boldsymbol{P} = \big(\boldsymbol{p}_{S_1}(\boldsymbol{\alpha}_1), \ldots, \boldsymbol{p}_{S_n}(\boldsymbol{\alpha}_n)\big)$.

We now make an important observation. Assume that we successively wish to check $\mathcal{F}_w(r-1)$ for $(S_1, \ldots, S_{r-1})$ and $\mathcal{F}_w(r)$ for $(S_1, \ldots, S_{r-1}, S_r)$, as is the case in the algorithm outlined in the previous paragraph: we then may view the vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{r-1}$ randomly generated for $S_1, \ldots, S_{r-1}$ in the first of these two tasks as a suitable choice of random vectors for the same collection of subsets $S_1, \ldots, S_{r-1}$ in the second task. As a consequence, the verification of $\mathcal{F}_w(r)$ for $(S_1, \ldots, S_{r-1}, S_r)$ by Algorithm 3.3 (the second task) may be replaced, in these conditions and given $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{r-1}$, by the following simpler procedure.

Algorithm 3.4.

Step 1. Compute a basis $\boldsymbol{b}(S_r)_1, \ldots, \boldsymbol{b}(S_r)_{\delta(S_r)}$ of $\mathrm{Inv}[S_r]$.

Step 2. Randomly choose a vector $\boldsymbol{\alpha}_r$ from a uniform distribution over the box B_r defined by (34) and compute $\boldsymbol{p} = \sum_{i=1}^{\delta(S_r)} (\alpha_r)_i \boldsymbol{b}(S_r)_i$.

Step 3. Check if $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{r-1}, \boldsymbol{p}$ are linearly independent. If this is the case, $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$. Otherwise, $(S_1, \ldots, S_r)$ almost surely does not satisfy this property.

We will make use of this simplified verification algorithm in the next section.

We conclude this discussion with a comment. Instead of Algorithm 3.3, we could consider deterministic algorithms for checking $\mathcal{F}_w(r)$. One such algorithm is as follows. Suppose that a collection of bases $\mathcal{B}(S_j) = \{\boldsymbol{b}(S_j)_1, \ldots, \boldsymbol{b}(S_j)_{\delta(S_j)}\}$ of $\mathrm{Inv}(f_\ell : \ell \in S_j)$ $(j = 1, \ldots, r)$ has been computed. Then we can prove that $(S_1, \ldots, S_r)$ satisfies $\mathcal{F}_w(r)$ if and only if there exist $\boldsymbol{b}(S_j) \in \mathcal{B}(S_j)$ $(j = 1, \ldots, r)$ that are linearly independent. This can be reformulated as the problem of finding a maximal common independent set of two matroids over a finite index set $E = \cup_{j=1}^{r} E_j$, where $E_j = \{(j, i) : i = 1, \ldots, \delta(S_j)\}$ $(j = 1, \ldots, r)$. One is a partition matroid $\mathcal{M}_1 = (\mathcal{I}_1, E)$ where

$$\mathcal{I}_1 = \{I \subseteq E : \#(I \cap E_j) \leq 1 \ (j = 1, \ldots, r)\}.$$

The other is a linear matroid $\mathcal{M}_2 = (\mathcal{I}_2, E)$ where

$$\mathcal{I}_2 = \{I \subseteq E : \boldsymbol{b}(S_j)_i \ ((j, i) \in I) \text{ are linearly independent}\}.$$

If we apply a basic matroid intersection algorithm [4] to the pair $\mathcal{M}_1$ and $\mathcal{M}_2$, then the theoretical computational cost is of $O((\#E)^2 r + (\#E) r^2 c(r, n))$ arithmetic operations, where $c(r, n)$ stands for the cost for checking whether r n-dimensional column vectors are linearly independent. Note that $c(r, n) = O(r^2 n)$ arithmetic operations if we use the standard Gaussian elimination scheme. On the other hand, the probabilistic algorithm (more precisely, Steps 3 and 4 of Algorithm 3.3) consists of $O((\#E)n)$ arithmetic operations for computing the n-dimensional column vectors $\boldsymbol{p}_{S_j}(\boldsymbol{\alpha}_j)$ $(j = 1, \ldots, r)$ at Step 3 and $c(r, n)$ arithmetic operations for checking out their linear independence at Step 4. As we have just observed, the cost is reduced further in the context of our lexicographic minimization problem: when Algorithm 3.4 is used instead of Algorithm 3.3, the term $O((\#E)n)$ is reduced to $O((\#E_r)n)$ since the random vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, r-1)$ have been computed previously. The probabilistic algorithm is thus expected to be computationally much cheaper than the matroid intersection formulation. Another motivation for using the probabilistic algorithm is that it is very simple and easy to implement for preliminary numerical experiments.

3.5 The complete algorithm

We finally combine Algorithm 3.4 with the greedy algorithm given in Section 3.1 for the combinatorial optimization problem $\mathcal{P}(n)$, to generate a nonsingular matrix $\boldsymbol{P}$ that improves sparsity of our initial family of partially invariant functions. This gives the following computational procedure.

Algorithm 3.5.

Step 0. Let $r = 1$, $S_j = \emptyset$ $(j \in N)$ and $L_\ell(S_1) = \emptyset$ $(\ell \in M)$.

Step 1. Compute a permutation $\pi(1), \ldots, \pi(m)$ of $1, \ldots, m$ by sorting $\#L_\ell(S_1, \ldots, S_r)$ $(\ell \in M)$ such that $\#L_{\pi(1)}(S_1, \ldots, S_r) \leq \cdots \leq \#L_{\pi(m)}(S_1, \ldots, S_r)$. Let $k = 1$.

Step 2. Let $S_r = S_r \cup \{\pi(k)\}$ and $L_{\pi(k)}(S_1, \ldots, S_r) = L_{\pi(k)}(S_1, \ldots, S_r) \cup \{r\}$.

Step 2-1. Compute a basis $\boldsymbol{b}(S_r)_1, \ldots, \boldsymbol{b}(S_r)_{\delta(S_r)}$ of $\mathrm{Inv}[S_r]$.

Step 2-2. Randomly choose a vector $\boldsymbol{\alpha}_r$ from a uniform distribution over the box B_r defined by (34) and compute $\boldsymbol{p} = \sum_{i=1}^{\delta(S_r)} (\alpha_r)_i \boldsymbol{b}(S_r)_i$.

Step 2-3. Check if the vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, r-1)$ and $\boldsymbol{p}$ are linearly independent. If it is the case, jump to Step 4.

Step 3. Reset $S_r = S_r \backslash \{\pi(k)\}$ and $L_{\pi(k)}(S_1, \ldots, S_r) = L_{\pi(k)}(S_1, \ldots, S_r) \backslash \{r\}$.

Step 4. If $k < m$, increment k by one and return to Step 2. Else, go to Step 6 if $S_r = \emptyset$.

Step 5. Define $\boldsymbol{p}_r = \boldsymbol{p}$. If $r = n$, then output $(S_1, \ldots, S_n)$ and $\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)$, and stop. Otherwise let $r = r + 1$ and return to Step 1.

Step 6. Randomly choose vectors $\boldsymbol{p}_r, \ldots, \boldsymbol{p}_n$ uniformly distributed over the box

$$B = \{\boldsymbol{p} \in \mathbb{R}^n : -1 \leq p_i \leq 1 \ (i = 1, 2, \ldots, n)\}.$$

Output $(S_1, \ldots, S_n)$ and $\boldsymbol{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$ and stop.

Upon termination, the hopefully sparse dependence of the transformed functions g_ℓ on the new transformed variables $\boldsymbol{z}$ can be uncovered from the sets $L_\ell(\boldsymbol{S})$ and the relations (20). Some comments on this algorithm are now in order.

1. We have used the simplified Algorithm 3.4 in the body of Algorithm 3.5, exploiting information already computed during outer iterations 1 to $r - 1$.

2. To execute Step 2-1, we have chosen to compute and update a $q \times n$ nonsingular upper triangular matrix $\boldsymbol{V}$ whose row vectors form a basis of the space spanned by all the row vectors of the matrices $\boldsymbol{A}_\ell$ $(\ell \in S_r)$ defined in (1). Since

$$\mathrm{Inv}[S_r] = \{\boldsymbol{w} \in \mathbb{R} : \boldsymbol{A}_\ell \boldsymbol{w} = \boldsymbol{0} \ (\ell \in S_r)\} = \{\boldsymbol{w} \in \mathbb{R} : \boldsymbol{V}\boldsymbol{w} = \boldsymbol{0}\}$$

and $\boldsymbol{V}$ is an upper triangular matrix, a basis of $\mathrm{Inv}[S_r]$ is easily computed by solving $\boldsymbol{V}\boldsymbol{w} = \boldsymbol{0}$. In particular, if $\boldsymbol{V}$ is an $n \times n$ square matrix, we know that $\mathrm{Inv}[S_r]$ reduces to the origin; we can hence skip Steps 2-1 to 2-4 and immediately go to Step 3. When we restart the inner iteration with $k = 1$ and $S_r = \emptyset$ at Step 1, $\boldsymbol{V}$ is set to the $0 \times n$ empty matrix. If $\pi(k)$ is added to S_r at Step 2, $\boldsymbol{V}$ is updated to a $q' \times n$ upper triangular matrix $\boldsymbol{V}'$ whose row vectors form a basis of the space spanned by all the row vectors of the matrices $\boldsymbol{A}_\ell$ $(\ell \in S_r \cup \{\pi(k)\})$. This update is also obtained by applying a row elimination process to transform the matrix

$$\begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{A}_{\pi(k)} \end{pmatrix}$$

into an upper triangular matrix. Note that a procedure using orthogonal transformations instead of Gaussian elimination is also possible for improved numerical stability.

3. We have also chosen to carry out Step 2-3 by using the Gaussian elimination to compute and update an $r \times n$ upper triangular matrix $\boldsymbol{U}_r$ whose row vectors form a basis of the space spanned by $\boldsymbol{p}_j$ $(j = 1, \ldots, r)$. At step 0, $\boldsymbol{U}_0$ is set to the $0 \times n$ empty matrix, and $\boldsymbol{U}_1 = \boldsymbol{p}_1^T$ at Step 2-3. For $r = 2, \ldots, n$, the linear independence of

the column vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, r-1)$ and $\boldsymbol{p}$ is tested at Step 2-3 by applying a row elimination process to transform the $r \times n$ matrix

$$\begin{pmatrix} \boldsymbol{U}_{r-1} \\ \boldsymbol{p}^T \end{pmatrix}$$

into an upper triangular matrix This yields an $r \times n$ nonsingular update $\boldsymbol{U}_r$ for $\boldsymbol{U}_{r-1}$ if and only if the column vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, r-1)$ and $\boldsymbol{p}$ are linear independent; conversely, the column vectors are linearly dependent if and only if all components of the r-th row of the updated triangular matrix are identically zero. Again, a variant using orthogonal transformations is possible.

4. Step 6 exploits the fact that, if $S = \emptyset$, then $\mathrm{Inv}[S] = \mathbb{R}^n$. We then apply Theorem 3.2 to choose $\boldsymbol{p}_r, \ldots, \boldsymbol{p}_n$.

4 Preliminary numerical experiments on polynomial optimization problems

We now intend to show that Algorithm 3.5 achieves its objective by applying it to several instances of our motivating problem. We show, in particular, that it makes the sparse SDP relaxation [22] for solving optimization problems with partially invariant polynomial functions applicable to a wider range of problems.

Let $M_o \neq \emptyset$ and M_c be a partition of the index set $M = \{1, \ldots, m\}$; $M = M_o \cup M_c$ and $M_o \cap M_c = \emptyset$. Let $f_\ell : \mathbb{R}^n \to \mathbb{R}$ $(\ell \in M)$ be polynomials. We consider a partially separable POP of the form

$$\text{(globally) minimize } \sum_{k \in M_o} f_k(\boldsymbol{x}) \text{ subject to } f_\ell(\boldsymbol{x}) \geq 0 \ (\ell \in M_c). \tag{35}$$

Given a general POP of the form (35) with a family of partially invariant polynomial functions $f_\ell : \mathbb{R}^n \to \mathbb{R}$ $(\ell \in M)$, it is interesting to see whether there exists a nonsingular linear transformation $\boldsymbol{P}$ on the problem space such that the family of transformed polynomial functions $g_\ell(\boldsymbol{z}) = f_\ell(\boldsymbol{P}\boldsymbol{z})$ $(\ell \in M)$ becomes correlatively sparse. We can use Algorithm 3.5 for this purpose and apply it to produce a nonsingular matrix $\boldsymbol{P}$ and a set vector $\boldsymbol{S}$. As indicated above, we may then use $L_\ell(\boldsymbol{S})$ $(\ell \in M)$ to construct the csp matrix $\boldsymbol{R}(g_\ell : \ell \in M)$, and check condition (12) (or (11)) to see whether the resulting POP with the family of transformed polynomials $g_\ell(\boldsymbol{z})$ $(\ell \in M)$ is correlatively sparse.

As a first example, we start by considering the practical solution of our example (13), which, as we already observed, is partially separable yet completely dense. However, the application of a linear transformation $\boldsymbol{P}$ given in (14) to the problem space yields the transformed functions (17), with a tridiagonal csp matrix, allowing for a trivial sparse Cholesky factorization. The transformed problem is thus correlatively sparse, and the sparse SDP relaxation is thus expected to work effectively for minimizing $g(\boldsymbol{z})$. This expectation is fulfilled in practice since SparsePOP [23, 22], an implementation of the sparse relaxation in Matlab[2] using SeDuMi [21], could solve this POP with $n = 1,000$ in 20.3 seconds (15.4

[2]Copyright by The Mathworks, Inc.

seconds for converting the POP into an SDP relaxation problem and 3.7 seconds for solving the resulting SDP by SeDuMi).

Three kinds of problems are tested in our numerical experiments: sparse POPs over the unit simplex with objective functions from [6, 19], concave quadratic minimization problems with transportation constraints from [12], and randomly generated low-rank quadratic optimization problems. We show how much efficiency is gained when solving the problems from the increased correlative sparsity induced by the transformation.

Algorithm 3.5 was implemented in Matlab. After applying the transformation with the algorithm, we use the SparsePOP package for solving the resulting POPs. All numerical experiments were conducted using the Matlab toolbox SeDuMi [21] on a Macintosh Dual 2.5GHz PowerPC G5 with 2GB DDR SDRAM. We use the notation described in Table 1 for the description of numerical results. In all results reported, the relaxation order (i.e., the highest degree of the moment matrices) is set to 2. We note that the inequality and equality constraints of the POPs are satisfied at the approximate optimal solutions of all POPs tested since all the constraints are linear.

n	the number of variables of the POP
sizeA	the size of the SDP problem in the SeDuMi input format
#nzL	the number of nonzeros in the sparse Cholesky factor of the csp matrix, computed by Matlab functions symmad and chol
#nzA	the number of nonzeros in the coefficient matrix A of the SDP problem to be solved by SeDuMi
m.sdp.b	the maximum size of SDP blocks in the coefficient matrix A
a.sdp.b	the average size of SDP blocks in the coefficient matrix A
rel.err	the relative error of SDP and POP objective values
cpu	cpu time consumed by SeDuMi in seconds

Table 1: Notation

4.1 Sparse POPs over the unit simplex

The sparse POPs over the unit simplex which we consider have the form

$$\begin{array}{lll} \text{(globally) minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & \boldsymbol{x} \in S = \{\boldsymbol{x} \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1,\ x_i \geq 0\ (i = 1, 2, \ldots, n)\}, \end{array} \tag{36}$$

in which we insert the following well-known objective functions (see, for instance, [19] and the CUTEr collection [6]):

- the Broyden tridiagonal function

$$\begin{aligned} f_{\mathrm{Bt}}(\boldsymbol{x}) &= ((3-2x_1)x_1 - 2x_2 + 1)^2 + \sum_{i=2}^{n-1} ((3-2x_i)x_i - x_{i-1} - 2x_{i+1} + 1)^2 \\ &\quad + ((3-2x_n)x_n - x_{n-1} + 1)^2 . \end{aligned} \tag{37}$$

- the chained Wood function

$$f_{\rm cW}(\boldsymbol{x}) = 1 + \sum_{i\in J} \big(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 + 90(x_{i+3} - x_{i+2}^2)^2 + (1 - x_{i+2})^2 + 10(x_{i+1} + x_{i+3} - 2)^2 + 0.1(x_{i+1} - x_{i+3})^2\big), \quad (38)$$

  where $J = \{1, 3, 5, \ldots, n-3\}$ and n is a multiple of 4.

- the generalized Rosenbrock function

$$f_{\rm gR}(\boldsymbol{x}) = 1 + \sum_{i=2}^{n} \Big\{100\left(x_i - x_{i-1}^2\right)^2 + (1 - x_i)^2\Big\}. \quad (39)$$

Because these three functions have a banded csp matrix, their unconstrained minization does not require any transformation for sparsity. In fact, the unconstrained problems were (globally) solved up to $n = 1000$ by SparsePOP, as shown in [23], and their local solution is known to be efficient for even larger sizes. However, the unit simplex constraint added in (36) makes the csp matrix induced by the resulting constrained minimization problem fully dense, which implies that the performance of SparsePOP for global optimization is then equivalent to that of the dense relaxation [17].

Table 2 shows the numerical results of (36) with $f(\boldsymbol{x}) = f_{\rm Bt}(\boldsymbol{x})$ by SparsePOP. As n increases, we notice that the numbers of nonzeros in the Cholesky factor (#nzL) rapidly grow when no transformation is applied. The size of the SDP problems (sizeA), its number of nonzeros (#nzA), its maximal and average block size (m.sdp.b and a.sdp.b) are much smaller with the transformation than without it. As a result, the cpu time consumed is also much smaller with the transformation than without.

Column rel.err indicates that accurate solutions were not obtained except for $n = 4$, since the relative errors range from 1.1e-2 to 5.5e-3. The asterisk mark * in cpu time means that SeDuMi terminated, for reasons that are not clear, without achieving the desired accuracy given in its parameter pars.eps, whose default value is 1.0e-9, due to numerical problems. Instead, the solution was obtained with accuracy 1.0e-3 specified with the parameter pars.bigeps.

Numerical results of minimizing the chained Wood function (38) and the generalized Rosenbrock function (39) over the unit simplex are given in Tables 3 and 4, respectively. The numbers in the columns of #nzL, sizeA, #nzA, m.sdp.b and a.sdp.b decrease sharply when solving the problems with the transformation. We notice that our technique makes the solution of larger problems with $n = 100$ and $n = 200$ possible. Again, the effect of the transformation on cpu time is very positive. The asterisk mark $*$ indicates SeDuMi numerical problems, as mentioned before. The accuracy of the obtained solutions using the transformation is relatively good, although the solutions with no transformation show smaller relative errors.

No transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	10	[69, 770]	1020	15	5.59	3.0e-10	0.29
8	36	[494, 5634]	7272	45	10.1	1.8e-02	*5.31
12	78	[1819, 20098]	25220	91	14.6	4.4e-02	*152.10
100	5050	–	–	–	–	–	–
200	20100	–	–	–	–	–	–
Transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	9	[69, 770]	1114	15	5.59	3.7e-08	0.24
8	28	[272, 3073]	3868	21	7.06	2.9e-02	*1.31
12	60	[863, 11434]	13988	36	10.7	3.7e-02	*8.63
100	419	[5811, 73905]	93717	28	9.70	1.1e-02	*50.75
200	819	[11711, 149205]	191773	28	9.75	5.5e-03	*111.09

Table 2: Minimization of the Broyden tridiagonal function (37) over the unit simplex

No transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	10	[39, 549]	799	8	5.18	4.6e-06	0.22
12	78	[580, 10495]	15617	34	13.4	4.8e-08	5.09
20	210	[2485, 46217]	68435	76	21.7	3.8e-10	233.60
100	5050	–	–	–	–	–	–
200	20100	–	–	–	–	–	–
Transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	10	[34, 386]	569	10	4.40	2.2e-05	0.27
12	50	[268, 3292]	5022	13	6.94	2.1e-02	2.05
20	94	[459, 5163]	7971	13	6.66	4.8e-07	*3.18
100	473	[2976, 33975]	52137	23	7.44	1.1e-07	*29.02
200	936	[8267, 94524]	137705	18	8.54	9.7e-08	*104.12

Table 3: Minimization of the chained Wood function (38) over the unit simplex

No transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	10	[49, 626]	876	11	5.35	1.2e-11	0.28
8	36	[374, 4738]	6376	37	9.85	1.4e-10	2.99
12	78	[1455, 17330]	22452	79	14.3	1.2e-10	111.93
100	5050	–	–	–	–	–	–
200	20100	–	–	–	–	–	–
Transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
4	9	[44, 495]	687	10	4.50	9.8e-08	0.32
8	26	[189, 2166]	2973	21	6.19	3.6e-08	0.81
12	43	[328, 3624]	4887	16	6.52	1.8e-07	1.36
100	317	[3678, 42761]	57487	16	7.68	6.5e-06	*19.30
200	606	[5017, 54363]	77170	16	6.41	2.7e-06	*21.69

Table 4: Minimization of the generalized Rosenbrock function (39) over the unit simplex

4.2 Concave quadratic optimization problems with transportation constraints

We next consider the following transportation problem (test problem 8 in Chapter 2 of [12]):

$$\left.\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = \sum_{i=1}^{m}\sum_{j=1}^{k}\left(a_{ij}x_{ij} + b_{ij}x_{ij}^2\right) \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} x_{ij} = c_j \ (j=1,\ldots,k),\ \sum_{j=1}^{k} x_{ij} = d_i \ (i=1,\ldots,m) \\ & x_{ij} \geq 0 \ (i=1,\ldots,m, j=1,\ldots,n), \end{array}\right\} \tag{40}$$

The coefficients $a_{ij},\ b_{ij},\ c_j$, and d_i are randomly chosen with uniform distribution as

$$\left.\begin{array}{l} a_{ij} \in \{200, 201, \ldots, 800\},\ b_{ij} \in \{-6,-5,-4,-3,-2\}, \\ d_k \in \{2,3,\ldots,9\},\ c_j \in \{2,3,\ldots\},\ \sum_{j=1}^{k} c_j = \sum_{i=1}^{m} d_i, \end{array}\right\} \tag{41}$$

where we impose $m \leq k$.

The above values for the coefficients makes the problem less concave than those presented in [12], so that the resulting SDP relaxation is more efficient. In particular, small values for $|b_{ij}|$, c_j and d_i weaken the influence of the concave quadratic terms in the feasible region. As a result, the effects of the transformation obtained from Algorithm 3.5 are more clearly visible. We also note that the problem (40) without any transformation is already correlatively sparse to some extent, but not enough for a successful application of SparsePOP to a larger-scaled problem, as we will see below.

The numerical results of the transportation problem with the coefficient values as in [12], which is also listed in [13] as ex2_1_8, are included in Table 5. In this problem, $m = 4$ and $k = 6$, the number of variables being then 24. Note that the sparse SDP relaxation with relaxation order 2 is applied to both non-transformed and transformed cases in Table 5. We notice in those results that solving this problem using the transformation reduces the

Transformation	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
No	153	[1003, 13209]	19391	13	9.6	1.4e-05	4.41
Yes	124	[697, 7494]	14830	10	7.1	1.3e-01	*4.37

Table 5: Transportation problem ex2_1_8 in [13]

magnitude of #nzL, sizeA, #nzA, and a.sdp.b. However, it does not provide an accurate solution when compared with solving without the transformation. This may be partially due to numerical problems in SeDuMi, which is indicated by the asterisk mark in cpu time for the transformed case (note that when SeDuMi experiences numerical difficulty, it usually consumes more cpu time before it provides an inaccurate solution). A more fundamental reason for the larger relative error resulting from the sparse SDP relaxation of the transformed problem is that the sparse SDP relaxation sometimes does not work as effectively as the dense relaxation as shown numerically in [22]. This is confirmed by the observation that the sparse SDP relaxation with the relaxation order 3 could solve the transformed problem with rel.err = 2.9e-04, albeit at a cost of 1089.24 seconds. Moreover, the problem is too small to really show the effect of the transformation. Larger and numerically stable problems are needed to see how the transformation works for the problem. These observations have lead to the choices (41) of the coefficients for numerical tests.

Figure 1 exhibits the Cholesky factor of the csp matrix before and after the transformation for (40) with $m = 9$ and $k = 9$. The pictures are obtained by reordering the rows and columns of the csp matrix according to the symmetric approximate minimum degree permutation (Matlab function "symamd") before applying the Cholesky factorization to the csp matrix. The transformation decreases the number of nonzero elements in the Cholesky factor of the csp matrix from 1897 to 855. As shown in [22], it is crucial to have more sparsity in the Cholesky factor of the csp matrix to obtain a solution of POPs efficiently. The transformation is very effective at increasing the sparsity of (40).

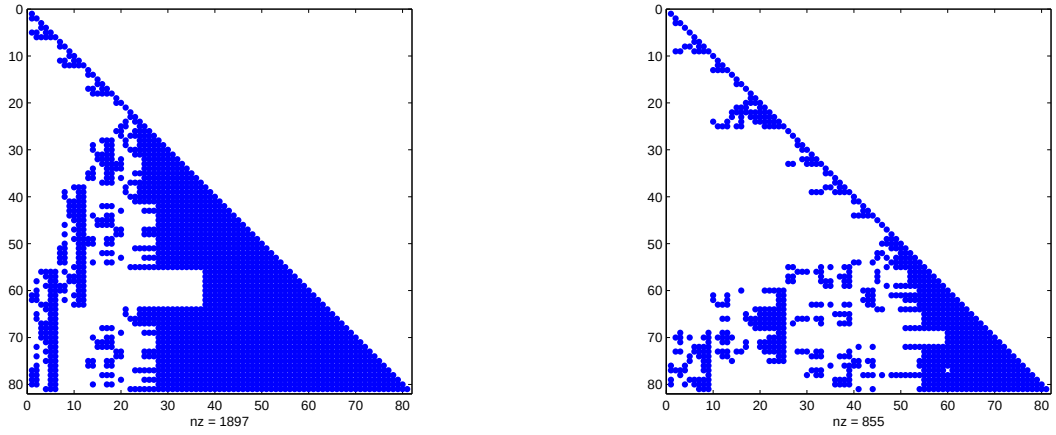

Figure 1: Cholesky factor of the csp matrix of (40) before and after the transformation with $m = 9$ and $k = 9$

No transformation								
m	k	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
5	5	228	[1149, 15003]	22079	14	10.2	1.0e-09	4.74
5	10	981	[6328, 88584]	129312	27	17.1	9.7e-10	344.27
5	15	2187	–	–	–	–	–	–
5	20	3802	–	–	–	–	–	–
6	6	371	[3459, 49234]	72021	21	15.4	3.7e-10	82.61
7	7	687	[8303, 123012]	179297	29	21.0	2.3e-09	832.61
8	8	1177	–	–	–	–	–	–
9	9	1897	–	–	–	–	–	–
Transformation								
m	k	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
5	5	136	[713, 7750]	1376	10	7.10	1.3e-08	1.96
5	10	347	[2400, 24914]	46866	14	8.76	1.1e-09	18.33
5	15	599	[4672, 43766]	76813	17	9.16	2.5e-03	*89.65
5	20	757	[6024, 57920]	105231	16	9.23	7.9e-03	*160.86
6	6	282	[2478, 25190]	47627	17	10.2	2.4e-08	57.12
7	7	388	[2842, 27898]	51126	15	9.27	7.8e-10	21.63
8	8	702	[7217, 69925]	128576	22	12.5	3.5e-03	*662.96
9	9	855	[9585, 86186]	141775	25	11.6	5.5e-03	*947.87

Table 6: Modified transportation problems

We display numerical results for problem (40) using coefficients (41) in Table 6. Note that m and k completely determine the correlatively sparse structure of the problem (40) and the nonsingular transformation $\boldsymbol{P}$, and that any choice of random coefficients a_{ij}, b_{ij}, c_j and d_i satisfying (41) is irrelevant to #nzL, sizeA, #nzA, m.sdp.b and a.sdp.b in Table 6; hence their choice does not affect much cpu though rel.err of the transformed problem may vary, as shown in Table 6. The number of variables is $m \times k$. We notice that the numbers in the columns #nzL, sizeA, #nzA, m.sdp.b, a.sdp.b are again much smaller with the transformation, the gain increasing with the number of variables. Indeed, the transformation is a key for computing a solution for large-sized problems, as shown in the rows of (m,k)=$(5, 15), (5, 20)$, $(8, 8)$, and $(9, 9)$. We see under "Transformation" in Table 6 that the problem with (m, k) equal to (7,7) required less cpu time than for (6,6). This is because the proposed method works more effectively in reducing the density of problem (7,7) than for the problem (6,6), as is confirmed by the values in the m.sdp.b and a.sdp.b. As expected, the relative errors reported by SparsePOP with the transformation are larger than without when both solutions were obtained.

4.3 Low rank quadratic optimization problems

We finally consider a linearly constrained quadratic optimization problem (QOP) of the form

$$\left.\begin{array}{rll} \text{(globally) minimize} & \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} & \\ \text{subject to} & 1 - (\boldsymbol{a}^j)^T\boldsymbol{x} \geq 0 \ (j = 1, \ldots, m) & 0 \leq x_i \leq 1 \ (i = 1, \ldots, n). \end{array}\right\} \quad (42)$$

Here $\boldsymbol{Q}$ denotes an $n \times n$ symmetric matrix, $\boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{a}^j \in \mathbb{R}^n$ $(j = 1, \ldots, m)$. Let r be the rank of $\boldsymbol{Q}$. The quadratic term $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ in the objective function can be written as

$$\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} = \sum_{k=1}^{r} \lambda_k \left((\boldsymbol{q}^k)^T\boldsymbol{x}\right)^2, \tag{43}$$

where λ_k $(k = 1, \ldots, r)$ denote the eigenvalues of $\boldsymbol{Q}$, and $\boldsymbol{q}^k$ denotes normalized eigenvectors corresponding to the eigenvalue λ_k $(k = 1, \ldots, r)$. Letting

$$\begin{aligned}
f_k(\boldsymbol{x}) &= \lambda_k \left((\boldsymbol{q}^k)^T\boldsymbol{x}\right)^2 \ (k = 1, \ldots, r), \\
f_{r+1}(\boldsymbol{x}) &= \boldsymbol{c}^T\boldsymbol{x}, \\
f_{j+r+1}(\boldsymbol{x}) &= 1 - (\boldsymbol{a}^j)^T\boldsymbol{x} \ (j = 1, \ldots, m), \\
f_{i+r+1+m} &= x_i \ (i = 1, \ldots, n), \\
f_{i+r+1+m+n} &= 1 - x_i \ (i = 1, \ldots, n), \\
M_o &= \{1, \ldots, r+1\}, M_c \ = \ \{r+2, \ldots, r+1+m+2n\}, \\
M &= M_o \cup M_c,
\end{aligned}$$

we can rewrite the QOP (42) in the form (35). We note that the problem is partially separable, that each function f_j is partially invariant with an invariant subspace of dimension $n-1$ $(\ell \in M)$, and that $\mathrm{Inv}(f_{i+r+1+m}) = \mathrm{Inv}(f_{i+r+1+m+n})$ $(i = 1, \ldots, n)$. Therefore, if $r+1+m+n$ is moderate relative to the dimension n of the variable vector $\boldsymbol{x} \in \mathbb{R}^n$, Algorithm 3.5 is expected to work effectively on the family f_ℓ $(\ell \in M)$ for transforming the QOP (42) into a correlatively sparse QOP.

We report here on the minimization of low rank concave quadratic forms subject to the unit box constraint with $r = 4$, $m = 0$ and $n = 10,\ 20,\ 40,\ 60,\ 80,\ 100$. The coefficients of the quadratic form $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ described in (43) were generated as follows: vectors $\boldsymbol{q}^k \in \mathbb{R}^n$ $(k = 1, 2, \ldots, r)$ and real numbers λ_k $(k = 1, 2, \ldots, r)$ were randomly generated such that

$$\begin{aligned}
&\|\boldsymbol{q}^k\| = 1 \ (k = 1, \ldots, r), \ (\boldsymbol{q}^j)^T\boldsymbol{q}^k = 0 \ (j \neq k), \\
&-1 < \lambda_k < 0 \ (k = 1, 3, 5, \ldots), \ 0 < \lambda_k < 1 \ (k = 2, 4, 6, \ldots).
\end{aligned}$$

The numerical results are shown in Tables 7. In this table, we observe that the transformation reduces #nzL, sizeA, #nzA, m.sdp.b, and a.sdp.b for all dimensions tested, and makes the solution of the problems for $n = 10,\ 20,\ 30$ faster than without the transformation. In particular, we see a critical difference in the size and cpu time between the transformed and untransformed problems for $n = 30$. The problems of $n \geq 40$ could not be handled without the transformation. As seen in the previous numerical experiments, the accuracy of the solutions obtained using the transformation is deteriorating as n becomes large, which is in accordance with the weaker nature of the sparse SDP relaxation. We remark here that any random choice of the coefficient vectors $\boldsymbol{c}$, $\boldsymbol{a}^j$ and $\boldsymbol{q}^k$ is irrelevant to #nzL, sizeA, #nzA, m.sdp.b and a.sdp.b with probability 1 even in the case of transformation (because the output $(S_1, \ldots, S_n)$ of Algorithm 3.5 applied to (42) with fixed m and n is independent from any generic choice of the coefficient vectors), while it affects the quality of the resulting sparse SDP relaxation problems and the numerical stability in solving them, that is, rel.err and cpu in the case of transformation.

No transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
10	55	[285, 5511]	7910	11	11.0	1.0e-10	1.01
20	210	[1770, 39221]	56820	21	21.0	5.9e-10	74.13
30	465	[5455, 127131]	184730	31	31.0	6.2e-10	3261.90
40	820	–	–	–	–	–	–
60	1830	–	–	–	–	–	–
80	3240	–	–	–	–	–	–
100	5050	–	–	–	–	–	–
Transformation							
n	#nzL	sizeA	#nzA	m.sdp.b	a.sdp.b	rel.err	cpu
10	46	[211, 2963]	7863	8	7.61	3.8e-07	0.89
20	120	[623, 7957]	21038	9	8.51	7.9e-02	*12.52
30	194	[1073, 12887]	34838	9	8.68	3.0e-07	7.30
40	238	[1523, 17817]	48638	9	8.76	1.7e-06	11.04
60	347	[2423, 27677]	76238	9	8.84	1.0e-01	*36.26
80	446	[3323, 37537]	103838	9	8.88	7.5e-02	*65.39
100	539	[4223, 47397]	131438	9	8.91	1.4e-03	*55.31

Table 7: Minimization of low rank quadratic forms subject to the unit box constraint, the QOP (42) with $r = 4$ and $m = 0$

Summarizing our numerical experience, we may conclude that the use of the proposed "sparsifying transformation" has a very positive impact of the sparsity structure of the transformed problems, in turn making the solution of large but originally dense instances realistic.

5 Concluding remarks and perspectives

We have proposed a numerical method for finding a linear transformation of the problem variables that reveals the underlying sparsity of partially separable nonlinear optimization problems. This method can be applied to reformulate very general classes of unconstrained and constrained optimization problems into a sparse equivalent. Its impact is potentially significant, as many of the existing algorithms for optimization exploit sparsity for efficient numerical computations.

We have shown in Section 4 that the method works effectively when incorporated into the sparse SDP relaxation method for polynomial optimization problems (POPs), even within the limits imposed by the weaker nature of the sparse relaxation. Used as a preprocessor for converting a given POP described by a family of partially invariant functions into a correlatively sparse formulation, it allows the sparse SDP relaxation method to solve the converted POP more efficiently. This makes larger dimensional POPs solvable.

A potentially important issue in our technique is the conditioning of the problem space linear transformation, as ill conditioned transformations could affect the numerical stability of the transformed problem. Recall that Algorithm 3.5 outputs $S_1, \ldots, S_n \subseteq M$ as well

as a $n \times n$ nonsingular matrix $\boldsymbol{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$ such that $S_j = \{\ell \in M : \boldsymbol{p}_j \in \mathrm{Inv}(f_\ell)\}$ $(j = 1, \ldots, n)$. Using these $S_1, \ldots, S_n \subseteq M$, we can consider the problem of minimizing the condition number of $\boldsymbol{P}' = (\boldsymbol{p}'_1, \ldots, \boldsymbol{p}'_n)$ subject to $\boldsymbol{p}'_j \in \mathrm{Inv}(S_\ell)$ $(j = 1, \ldots, n)$ and $\|\boldsymbol{p}'_1\| = 1$. This problem is too difficult to solve exactly, but we might be able to develop some heuristic method. As reported in Section 4, SeDuMi often terminated with numerical errors when solving SDP relaxation problems of transformed POPs. However, we believe that these difficulties may not be directly attributable to a poor conditioning of the problem space transformation: the condition number of the computed transformations indeed ranged roughly from the order of 10^2 to the order of 10^4, which remains moderate. The fact that the sparse SDP relaxation is less expensive, but weaker than the dense one [17], may be the main reason of the large relative errors in most cases; our technique can thus be viewed as an additional incentive for further research on improving the efficiency of sparse relaxations and understanding the numerical difficulties reported by SeDuMi.

Our ultimate objective is to find a nonsingular linear transformation in the problem space that substantially enhances exploitable sparsity. This goal is clearly not restricted to methods for solving POPs, and is probably very hard to achieve. It may vary in its details if different classes of numerical methods are used: if we consider handling linearized problems by iterative methods such as conjugate-gradients, the trade-off between the amount of sparsity and the problem conditioning may become more important than correlative sparsity, a concept clearly motivated by direct linear solvers and efficient Cholesky factorizations. It is also worthwhile to note that our approach is not restricted to nonlinear problems either: the key object here remains the matrix of the form (2), which may result from a partially separable nonlinear problem as introduced here, or from a purely linear context, such as the assembly of a finite element matrix.

Thus the authors are very much aware that the present paper only constitutes a first step towards this objective. Many challenging issues remain.

One such issue is the further developments of the formulation. Although the technique of transforming the problem into a combinatorial lexicographic maximization problem has indeed showed its potential, we do not exclude that other formulations could bring further benefits, both in terms of the properties of the problem space transformation (which we haven't really touched here) and in terms of numerical efficiency. Alternative algorithms for the present formulation are also of interest.

Even in the proposed framework, the current Matlab code is admittedly far from optimized. It is for instance interesting to note that the current code required 598 seconds for computing a 200×200 transformation and 96 additional seconds for transforming the problem functions, when applied to the minimization of the Broyden tridiagonal function in 200 variables over the unit simplex. The total cpu time of 694 seconds for the transformation thus currently largely exceeds the 111 seconds necessary for solving the SDP relaxation problem. Is this inherent to our approach, or could the cost of finding and applying the transformation be reduced and amortized better with respect to the repeated use of the simplified problem in computationally intensive techniques such as SDP relaxation, interior point methods or other algorithms?

Clearly, only continued investigation and numerical experience will assess the true potential of "sparsity enhancing" techniques of the type proposed here. But the authors feel that the initial approach is promising.

References

[1] J. R. S. Blair and B. Peyton, "An introduction to chordal graphs and clique trees," In: A. George, J. R. Gilbert and J. W. H. Liu eds, *Graph Theory and Sparse Matrix Computation*, Springer, New York, pp.1-29, 1993.

[2] A. R. Conn, N. I. M. Gould and Ph. L. Toint, "Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach," In: W. W. Hager, D. W. Hearn and P. M. Pardalos eds, *Large Scale Optimization: State of the Art*, Kluwer, Dordrecht, (1994) 82–94.

[3] A. R. Conn, N. I. M. Gould and Ph. L. Toint, LANCELOT, *A Fortran Package for large-Scale Nonlinear Optimization (Release A)*, Springer, Heidelberg, 1992.

[4] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Saunders College Publishing, Fort Worth, 1976.

[5] D. M. Gay, "Automatically finding and exploiting partially separable structure in nonlinear programming problems," Bell Laboratories, Murray Hill, NJ (1996).

[6] N. I. M. Gould, D. Orban and Ph. L. Toint, "CUTEr, a Constrained and Unconstrained Testing Environment, revisited", *TOMS,* **29** (2003) 373–394.

[7] N. I. M. Gould, D. Orban and Ph. L. Toint, "GALAHAD, a Library of thread-safe Fortran packages for Large-Scale Nonlinear Optimization", *TOMS,* **29** (2003) 353–372.

[8] A. Griewank and Ph. L. Toint, "On the unconstrained optimization of partially separable functions", in M. J. D. Powell, ed., *Nonlinear Optimization 1981*, Academic Press, New York (1982) 301-312.

[9] A. Griewank and Ph. L. Toint, "Partitioned variable metric updates for large structured optimization problems", *Numerische Mathematik*, **39** (1982) 119-137.

[10] A. Griewank and Ph. L. Toint, "Local convergence analysis for partitioned partitioned quasi-Newton updates", *Numerische Mathematik*, **39** (1982) 429-448.

[11] A. Griewank and Ph. L. Toint, "On the existence of convex decomposition of partially separable functions", *Mathematical Programming*, **28** (1984) 25-49.

[12] C. Floudas, P. Pardalos, C. Adjiman, W. Esposito, Z. Gümüs, S. Harding, J. Klepeis, C. Meyer and C. Schweiger *Handbook of test problems in local and global optimization* Kluwer Academic Publishers, 1999

[13] GLOBAL Library, `http://www.gamsworld.org/global/globallib.htm`

[14] S. Kim, M. Kojima and H. Waki, "Generalized Lagrangian duals and sums of squares relaxation of sparse polynomial optimization problems". *SIAM Journal on Optimization*, **15** (2005) 697-719.

[15] M. Kojima, S. Kim and H. Waki, "Sparsity in sums of squares of polynomials", *Mathematical Programming*, **103** (2005) 45-62.

[16] M. Kojima, M. Muramatsu, "A note on SOS and SDP relaxations for polynomial optimization problems over symmetric cones". To appear in *Computational Optimization and Applications*.

[17] J. B. Lasserre, "Global optimization with polynomials and the problems of moments", *SIAM Journal on Optimization*, **1** (2001) 796-817.

[18] J. B. Lasserre, "Convergent semidefinite relaxation in polynomial optimization with sparsity", working paper, LAAS-CNRS, 7 Avenue du Colonel Roche, 31077 Toulouse, France, November 2005.

[19] J. J. More, B. S. Garbow and K. E. Hillstrom, "Testing Unconstrained Optimization Software", *ACM Trans. Math. Soft.*, **7**, (1981) 17–41.

[20] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York (1999).

[21] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones", *Optimization Methods and Software*, **11 & 12** (1999) 625–653.

[22] H. Waki, S. Kim, M. Kojima and M. Muramatsu, "Sums of Squares and Semidefinite Programming Relaxations for Polynomial Optimization Problems with Structured Sparsity". *SIAM Journal on Optimization*, **17** (2006) 218–242.

[23] H. Waki, S. Kim, M. Kojima and M. Muramatsu, "SparsePOP : a Sparse Semidefinite Programming Relaxation of Polynomial Optimization Problems". Research Report B-414, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8552, Japan, March 2005.

[24] N. Yamashita, "Sparse quasi-Newton updates with positive definite matrix completion", Technical Report 2005-0008, Applied Mathematics and Physics, Kyoto University 606-8501, Kyoto, Japan, August 2005.