Comput Optim Appl (2007) 36: 321–341
DOI 10.1007/s10589-006-9003-y

Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization

Olaf Schenk · Andreas Wächter · Michael Hagemann

Published online: 22 February 2007

Abstract Interior-point methods are among the most efficient approaches for solving large-scale nonlinear programming problems. At the core of these methods, highly ill-conditioned symmetric saddle-point problems have to be solved. We present combinatorial methods to preprocess these matrices in order to establish more favorable numerical properties for the subsequent factorization. Our approach is based on symmetric weighted matchings and is used in a sparse direct LDL^T factorization method where the pivoting is restricted to static supernode data structures. In addition, we will dynamically expand the supernode data structure in cases where additional fill-in helps to select better numerical pivot elements. This technique can be seen as an alternative to the more traditional threshold pivoting techniques. We demonstrate the competitiveness of this approach within an interior-point method on a large set of test problems from the CUTE and COPS sets, as well as large optimal control problems based on partial differential equations. The largest nonlinear optimization problem solved has more than 12 million variables and 6 million constraints.

Keywords Nonconvex nonlinear programming · Interior-point method · Saddle-point problem · Numerical linear algebra · Maximum weight matching

O. Schenk (✉) · M. Hagemann
Departement of Computer Science, University of Basel, Klingelbergstr. 50, 4056 Basel, Switzerland
e-mail: olaf.schenk@unibas.ch

M. Hagemann
e-mail: michael.hagemann@unibas.ch

A. Wächter
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
e-mail: andreasw@watson.ibm.com

1 Introduction

In recent years, a large amount of work has been devoted to the problem of solving large symmetric indefinite systems in saddle-point form efficiently. One reason for this surge in interest is due the success of interior-point methods in nonlinear programming, which at their core require the solution of a series of linear systems in saddle-point form. Consider a nonlinear programming problem given by an objective function $f(x) : \mathbb{R}^n \to \mathbb{R}$ and constraint functions $c(x) : \mathbb{R}^n \to \mathbb{R}^m$, which are both assumed to be twice continuously differentiable. The objective is to find a local solution of the optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.1a}$$

$$\text{s.t.} \quad c(x) = 0, \tag{1.1b}$$

$$x \geq 0. \tag{1.1c}$$

For simplicity in the notation we assume without loss of generality that all variables have only a lower bound. Also note that problems with general inequality constraints can be transformed into the above formulation by introducing slack variables.

In an interior-point optimization framework, the solution of (1.1) is found by a series of Newton-type iterations that require in each step the solution of a linear system of equations of the form

$$Kx = \begin{bmatrix} H & A \\ A^T & -C \end{bmatrix} x = b \tag{1.2}$$

where the $n \times n$ matrix H is symmetric and potentially indefinite, C is a diagonal regularization matrix which is typically small and often zero, and the $n \times m$ matrix A has full column rank. See Sect. 2 for a detailed description of these matrices. For a detailed survey on solution techniques for large linear saddle-point systems the interested reader should consult [2].

Commonly, sparse direct factorization methods for symmetric indefinite systems are used to solve (1.2). These methods compute, for example, sparse Bunch–Kaufman [6] or Duff–Reid [14] factorizations

$$K = PQLDL^T Q^T P^T \tag{1.3}$$

where Q is a pivoting permutation matrix used for numerical stability, P is a fill-in minimization permutation matrix, L is unit lower triangular, and D is a block diagonal matrix with blocks of dimension 1 and 2. Unfortunately, until recently, sparse symmetric indefinite factorization methods were relatively inefficient compared to symmetric positive definite solvers [22]. However, with the invention of fast combinatorial algorithms [11, 36], which improve the diagonal dominance of the linear systems, the situation has dramatically changed. The use of matching-based preprocessing steps for (1.2) leads to symmetric indefinite sparse direct solvers that are almost as efficient as their positive definite counterparts. The key idea is to transform the matrix K in (1.2) into

$$\hat{K} = P_{\mathcal{M}} D_s K D_s^T P_{\mathcal{M}}^T, \tag{1.4}$$

such that the permuted system has a greater block-diagonal dominance than the original matrix, and to use factorization methods that honor the block-diagonal dominance of (1.4). Here, $P_{\mathcal{M}}$ is a permutation matrix that permutes large-off diagonal entries to be close to the diagonal, and the diagonal matrix D_s is a scaling matrix such that both columns and rows of $\hat{K}$ have unit infinity-norm.

The paper is organized as follows. Section 2 provides background on interior-point methods and discusses the requirements for factorization algorithms to solve (1.2). We briefly review the supernodal factorization approach for the direct solution of saddle-point matrices in Sect. 3. The matching algorithms used to reorder the matrix before the factorization are discussed in Sect. 4. Section 5 introduces the test problems and gives various performance statistics and comparisons of the proposed methods.

2 Optimization algorithm

Over the past ten years, a number of primal–dual interior-point methods for general nonlinear continuous optimization problems, such as (1.1), have been proposed (see [19] for a comprehensive survey). They can be derived in roughly two different ways, which we outline in the following. We contrast the two approaches in order to emphasize the importance of a proper handling of the nonconvex case.

The more recent point of view was developed in the 1980's, initially for problems in which $f(x)$ and $c(x)$ are linear. Here, the *perturbed primal–dual equations*

$$\nabla f(x) + \nabla c(x)\lambda - z = 0, \tag{2.1a}$$

$$c(x) = 0, \tag{2.1b}$$

$$XZe - \mu e = 0 \tag{2.1c}$$

play a central role. The vectors λ and z are the Lagrangian multipliers for the equality and bound constraints, and X and Z denote the diagonal matrices with the vector elements of x and z on the diagonal. The vector e denotes the vector of all ones, i.e., $e = (1, \ldots, 1)^T$, of appropriate dimension. For $\mu = 0$, Eqs. (2.1) together with "$x, z \geq 0$" are the first-order optimality conditions for (1.1). Methods in this class obtain a point satisfying those conditions by applying Newton's method to (2.1), where μ takes on strictly positive values, and, in a homotopy approach, is eventually driven to zero. The steps are obtained from

$$\begin{bmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_k e \end{pmatrix}. \tag{2.2}$$

Here $A_k = \nabla c(x_k)$, and W_k denotes the Hessian of the Lagrangian function for (1.1) w.r.t. x. Once a step has been computed, an appropriate step size is chosen that among other things ensures that non-negativity conditions $(x, z > 0)$ are satisfied for each iterate. Using the fact that the Jacobian of the system of Eq. (2.1) is non-singular at a non-degenerate local solution (x_*, λ_*, z_*) of (1.1), it is possible to derive methods that converge superlinearly toward (x_*, λ_*, z_*); see, e.g., [23].

A number of methods have been proposed that apply the primal–dual strategy as outlined above directly to nonlinear optimization problems (e.g., [15, 40]). Progress towards the solution is ensured by monitoring the norm of the optimality conditions ((2.1) with $\mu = 0$) and driving it to zero. While this is appropriate for problems where every solution of (2.1) is a solution for (1.1) (for example, when (1.1) corresponds to a convex optimization problem), we believe it is not suitable in the general nonconvex case. In a sense, these methods ignore the optimization aspect of the problem, and reduce the algorithm to finding a root of the optimality conditions. However, in practice, this can easily lead to convergence to stationary points that are not minimizers.

A different point of view of interior-point methods, originating from research directly for general nonlinear optimization problems, has been discussed and analyzed in detail by Fiacco and McCormick [16] in the 1960s. Given the original problem in the form (1.1), a sequence of corresponding *barrier problems*,

$$\min_{x\in\mathbb{R}^n} \varphi_\mu(x) = f(x) - \mu \sum_{i=1}^{n} \ln\bigl(x^{(i)}\bigr) \tag{2.3a}$$

$$\text{s.t.} \quad c(x) = 0, \tag{2.3b}$$

is solved to increasingly tighter tolerances, while again the barrier parameter μ is driven to zero. Here, it is possible to use techniques (for the solution of the individual barrier problems) that have been developed for general (nonconvex) optimization, such as SQP-type methods. In particular, we may use globalization approaches that have been proposed to handle the case where not all stationary points satisfying the first order optimality conditions are local minimizers. In the following, we restrict our discussion to line search algorithms, such as the one implemented in IPOPT [41], the code used for the numerical results presented in Sect. 5.

The search directions are obtained from solving the linear system

$$\begin{bmatrix} \tilde{W}_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + A_k \lambda_k \\ c(x_k) \end{pmatrix}, \tag{2.4}$$

where $\tilde{W}_k$ is an approximation of the Hessian of the Lagrangian for the barrier problem (2.3). One can show that these steps, together with

$$\Delta z_k = \mu X_k^{-1} e - z_k - \Sigma_k \Delta x_k, \tag{2.5}$$

correspond to solutions of (2.2), if $\tilde{W}_k = W_k + \Sigma_k$ with $\Sigma_k = X_k^{-1} Z_k$. Therefore, at least close to the solution, the algorithms derived from both points of views are very similar, and fast local convergence can be achieved by either.

However, the crucial difference between the two classes of algorithms lies in how they behave when the iterates are not close to a solution. SQP-type method use globalization techniques based on a merit function or a filter method (the approach used in IPOPT, see [42] for details). Here, it is crutial that the search directions generated from (2.4) have appropriate descent properties to guarantee that progress can be made for the globalization framework, so that overall convergence to non-optimal stationary points is less likely or can be avoided. To ensure this, SQP-type line search methods usually require that the projection of $\tilde{W}_k$ (or a modification thereof) onto

the null space of A_k^T is positive definite, or, equivalently, that the matrix in (2.4) is non-singular and has exactly m (the number of constraints) negative eigenvalues [35].

In contrast to this, a method that uses the unmodified Hessian matrix in (2.2) might generate steps that are increasing the objective function even for arbitrarily small step sizes, if the current iterate is feasible and the projection of $\tilde{W}_k$ is indefinite. We therefore strongly believe that it is desirable to include the fact that one is dealing with a minimization problem in the design of an optimization method for nonconvex optimization.

2.1 Requirements for direct solvers

A number of algorithms for the factorization of indefinite symmetric systems provide information about the inertia of the factorized matrix on the fly, e.g., [6]. In the context of an SQP-type line search optimization method it is therefore possible to ensure the required descent properties by performing a "trial" factorization for systems of the form (2.4), where the matrix $\tilde{W}_k$ is modified according to some heuristics, until the inertia of the matrix is correct.

The method implemented in IPOPT is a primal–dual interior-point algorithm that generates iterates for the variables, (x_k, λ_k, z_k), using steps from the linear system (2.2), but for the reasons just given, instead of solving this nonsymmetric system directly, it solves a "regularized" form of (2.4), namely,

$$\begin{bmatrix} W_k + \Sigma_k + \delta_x I & A_k \\ A_k^T & -\delta_c I \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + A_k \lambda_k \\ \zeta(x_k) \end{pmatrix}, \tag{2.6}$$

and obtains the search direction for the bound multipliers from (2.5). The perturbation parameter for the Hessian, $\delta_x \geq 0$, is chosen to ensure that the matrix in (2.6) has the correct inertia, and that therefore the search direction has the required descent properties. In addition, a small perturbation parameter $\delta_c \approx 10^{-8}$ might be chosen in case the matrix A_k appears to be column rank-deficient, to ensure that the matrix in (2.6) is non-singular. The heuristic for choosing δ_x and δ_c implemented in IPOPT is described in [41]; it usually attempts first to work with $\delta_x = \delta_c = 0$ to obtain the pure primal–dual Newton direction for fast convergence, and only if the inertia is not correct with this choice are other values tried.

Another approach for modifying the Hessian matrix is to use an inertia controlling factorization method (see, e.g., [17]), which modifies the matrix on the fly during the factorization and ensures the desired descent properties.

Using an iterative solver for the computation of a good search direction from (2.2) or (2.6) appears difficult, since it seems not possible to ensure the required descent properties. A hybrid approach has been used in trust-region algorithms (see, e.g., [7]), where a matrix of the form (2.4), with $\tilde{W}_k$ replaced by Σ_k, is factorized and used to compute the feasibility component of the search direction, and an iterative method, working in the null space of the constraint Jacobian, is used to compute the remaining component [7]. If the inertia of the linear system is not correct, this iterative procedure will encounter directions of negative curvature, which now can be exploited explicitly, so that a good descent search direction is obtained. Other iterative procedures for the solution of saddle point problems have been proposed, but they usually require

either the matrix H in (1.2) to be positive definite, at least in the null space of A^T (e.g., [3, 21]), or C to be non-singular (e.g., [18]).

However, in this work we use Level-3 BLAS supernodal left-looking direct solver and we investigate various pivoting strategies based on weighted graph matchings.

3 Supernodal factorizations methods

We use a Level-3 BLAS supernodal left-looking factorization as described in [27, 33, 38] and discuss two additional pivoting techniques for symmetric indefinite systems that are described below. The first pivoting technique has also been used in [38]—the second variant is novel and is used to improve the accuracy.

3.1 Pivoting variant I: supernodal Bunch–Kaufman

An interchange among rows and columns of a supernode of diagonal size n_s, referred to as Supernode-Bunch–Kaufman (SBK) pivoting, has no effect on the overall fill-in and this is the mechanism for finding a suitable pivot in our SBK method. However, there is no guarantee that the numerical factorization algorithm would always succeed in finding a suitable pivot within a diagonal block related to a supernode. When the algorithm reaches a point where it cannot factor the supernode based on the previously described 1×1 or 2×2 pivoting, it uses the pivot perturbation strategy described in [38]. The magnitude of the potential pivot is tested against a constant threshold of $\epsilon \cdot \|A\|_1$, where ϵ is the square root of the machine precision. If the pivot is smaller, it is set to $\text{sign}(a_{ii}) \cdot \epsilon \cdot \|A\|_1$—this perturbation trades off some numerical stability for the ability to keep the pivots from getting too small. The result of this pivoting approach is that the factorization is, in general, not accurate and iterative refinement might be necessary.

1. $\gamma_1 := |a_{r1}| = \max_{k=2,\ldots,n_s} |a_{k1}|$
   with diagonal block of size n_s
2. $\gamma_r \geq \gamma_1$ is the magnitude of the largest
   off-diagonal in the $r-$row of the block
3. **if** $\max(|a_{11}|, \gamma_1) \leq \epsilon \cdot \|A\|_1$:
4. use pivot perturbation:
   $\tilde{a}_{11} = sign(a_{11}) \cdot \epsilon \cdot \|A\|_1$
5. use perturbed $\tilde{a}_{11}$ as a 1×1 pivot.
6. **else if** $|a_{11}| \geq \alpha \gamma_1$:
7. use a_{11} as a 1×1 pivot,
8. **else if** $|a_{11}| \cdot \gamma_r \geq \alpha \gamma_1^2$:
9. use $|a_{11}|$ as a 1×1 pivot,
10. **else if** $|a_{rr}| \geq \alpha \gamma_r$:
11. use $|a_{rr}|$ as a 1×1 pivot,
12. **else**
13. use $\begin{pmatrix} a_{11} & a_{r1} \\ a_{r1} & a_{rr} \end{pmatrix}$ as a 2×2 pivot.

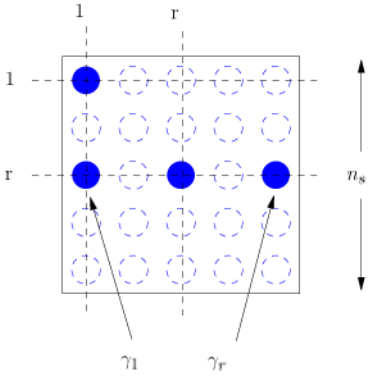
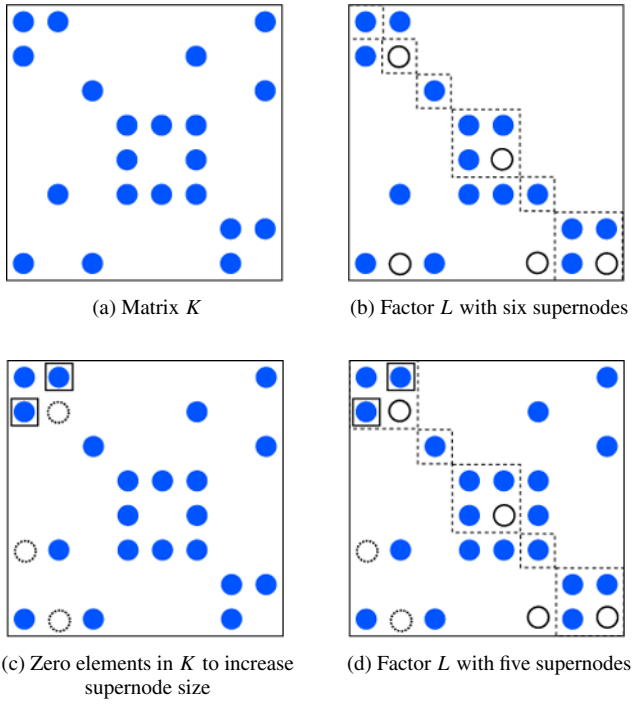


Fig. 1 Supernode Bunch–Kaufman pivot selection with half-machine precision perturbation

Figure 1 describes the usual 1×1 and 2×2 Bunch–Kaufman pivoting strategy [6] within the diagonal block corresponding to a supernode of size n_s. The pivoting strategy is supplemented with half-machine precision perturbation techniques. The Bunch–Kaufman pivoting method computes two scalars γ_1 and γ_r. The scalar γ_1 is the largest off-diagonal element, e.g., $|a_{r1}|$, in the first column of the diagonal block corresponding to the supernode of size n_s. The scalar γ_r is the largest off-diagonal element in the corresponding row r. The scalar $\alpha = (\sqrt{17}+1)/8$ is chosen to minimize the element growth [6]. With this choice, the element growth in the diagonal block after k steps is bounded by the factor $(2.57)^{k-1}$. The algorithm than selects either 1×1 and 2×2 pivots for the factorization. If both $|a_{11}|$ and $|\gamma_1|$ are too small, e.g., smaller than $\epsilon \cdot \|A\|_1$, we apply the pivot perturbation technique as described above.

3.2 Pivoting variant II: use all preselected 2×2 pivots

The 1×1 and 2×2 pivoting search of the SBK method is applied within the block diagonal of a supernode. In the extreme case, the supernode exists of only one column and the SBK pivoting can degenerate to diagonal pivoting. Therefore any permutation that symmetrically permutes large-off diagonal entries close to the diagonal or that identifies suitable 2×2 pivots prior to the numerical factorization would further improve the accuracy. In Sect. 4, we will discuss these permutations which are based on



(a) Matrix K (b) Factor L with six supernodes

(c) Zero elements in K to increase supernode size (d) Factor L with five supernodes

Fig. 2 **a** and **b** Matrix and factor of supernodal Bunch–Kaufman pivoting with supernodes $\{(1), (2), (3), (4, 5), (6), (7, 8)\}$. **b** and **c** Matrix and factor of preselected 2×2 pivots with additional zero elements and supernodes $\{(1, 2), (3), (4, 5), (6), (7, 8)\}$

combinatorial graph algorithms. The strategy in Sect. 4 results in permutation matrices that, if applied to (1.2), provide a permuted system $\tilde{K}$ of the saddle-point system K such that good initial 2×2 diagonal pivots are found prior to the numerical factorization. In order to enforce the use of these preselected pivot blocks during the LDL^T factorization, we merge the 2×2 column structure in L in such a way that these initial 2×2 pivot structure is maintained in the factor L.

This is illustrated in Fig. 2. In the pivoting variant I we will restrict the pivoting search within diagonal-blocks of the supernodes as shown in Fig. 2 (a) and (b). In the pivoting variant II we will identify 2×2 and enforce the use of these preselected pivot blocks by adding additional zero elements to the structure of K. As a result, the size of the supernode increases, e.g. we will merge column/row 1 and 2 into one supernodes of size 2, in which a 2×2 Bunch–Kaufman pivoting is performed instead of two 1×1 diagonal pivot elements.

4 Symmetric matchings for a-priori pivoting

Matching algorithms work on the associate graph representations of the matrices. In our case, the algorithms work on the *bipartite graph* $G_A = (V_r, V_c, E)$, where V_r and V_c are vertex sets of cardinality n_K (where n_K is the size of the matrix K), representing the rows and columns of the matrix respectively, and $E = \{(i, j) \mid a_{ij} \neq 0\}$ is the set of edges connecting the vertices in V_r and V_c.

A matching $\mathcal{M}$ in G_A is, in general, a subset $\mathcal{M} \subseteq E$ with the property that each vertex $v \in \{V_r \cup V_c\}$ is at most incident to one edge in $e \in \mathcal{M}$. If every vertex v is incident to exactly one edge in $\mathcal{M}$, the matching is called a *perfect matching*. A *maximum weight matching* is a matching whose matched edges $e \in \mathcal{M}$ maximize a weight function $w(\cdot)$, with

$$w(\mathcal{M}) = \sum_{(i,j)\in\mathcal{M}} (S)_{ij} \tag{4.1}$$

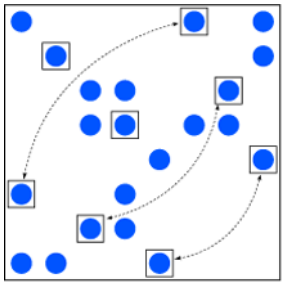
where S is a weight coefficient matrix for the edges of K, with $S = K$ as a simple case. The problem of finding it is known as the *linear sum assignment problem* or *bipartite weighted matching problem*. In general, the weight function (4.1) does not define a unique matching. In our case we are interested in a perfect maximum weight matching, i.e., a matching with maximum weight under the condition that all rows and columns are matched. If such a matching does not exist, the matrix is structurally rank deficient. Furthermore, we want to avoid small or zero entries in the matching. Therefore we apply a simple logarithmic transformation to S, with $S_{ij} = \log|K_{ij}|$ which yields a *maximum product matching*.

The use of combinatorial techniques in the solution of indefinite linear systems goes back to the early eighties. Duff [9] introduced a matching method to permute non-zero entries of the matrix onto the diagonal. Olschowka and Neumaier [36] proposed to use maximum weight matchings as a form of a-priori pivoting. The first implementations of this idea for nonsymmetric sparse matrices were provided by Duff and Koster [12] and Gupta and Ying [24].

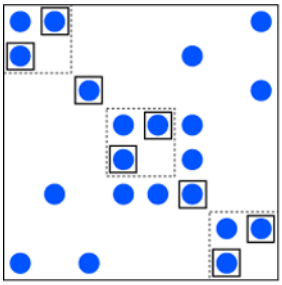
Gilbert and Duff [11] were the first to introduce an algorithm that maintains symmetry for symmetric indefinite problems. They proposed to form 2×2 diagonal

pivots based on the information gathered through symmetrized maximum weight matchings. The matchings were symmetrized by examining the cycle structure of the matching and splitting cycles of length longer than two. Since then, several authors have conducted research in this area. Duff and Pralet [13] elaborate the idea from the original talk and compare various strategies. Schenk and Gärtner combine the matching approach with restricted Bunch–Kaufman [38] pivoting in supernodes, to improve performance and parallelism. Furthermore Hagemann and Schenk investigate preconditioning techniques based on weighted matchings in [25]. In this report, we build on the results in [38], and introduce a novel technique to determine symmetric weighted matchings in saddle-point systems of the form (1.2).

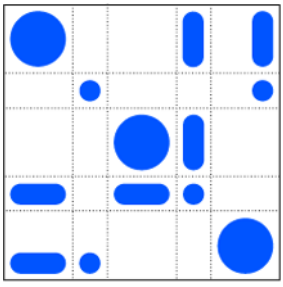
The basic idea of the symmetric a-priori pivoting is to form 2×2 diagonal blocks based on the matched off-diagonal entries of the matrix. In order to find diagonal 2×2 blocks, we ideally want a *symmetric* matching of maximum weight, i.e., a matching of maximum weight where $(i, j) \in \mathcal{M} \Leftrightarrow (j, i) \in \mathcal{M}$. Given the matching, the corresponding permutation $P_{\mathcal{M}}$, which reorders the off-diagonal matched entries (i, j) and (j, i) into 2×2 diagonal blocks, only needs to order nodes i and j consecutively. This is illustrated in Figs. 3 (a) and (b). Furthermore, since we factorize the reordered matrix, we also want to minimize the fill-in by applying a fill-reducing reordering like METIS [26]. In order to maintain the diagonal block structure, we compress the matrix by merging the structure of the rows and columns belonging to a 2×2 diagonal block. See Fig. 3 (c) for an illustration. This yields the reordering P, which depends on $\mathcal{M}$ and $P_{\mathcal{M}}$. It has been shown that this combination can be very



(a) Illustration of $\mathcal{M}s$ in K



(b) $P_{\mathcal{M}} K P_{\mathcal{M}}^T$



(c) Compressed 5×5 matrix

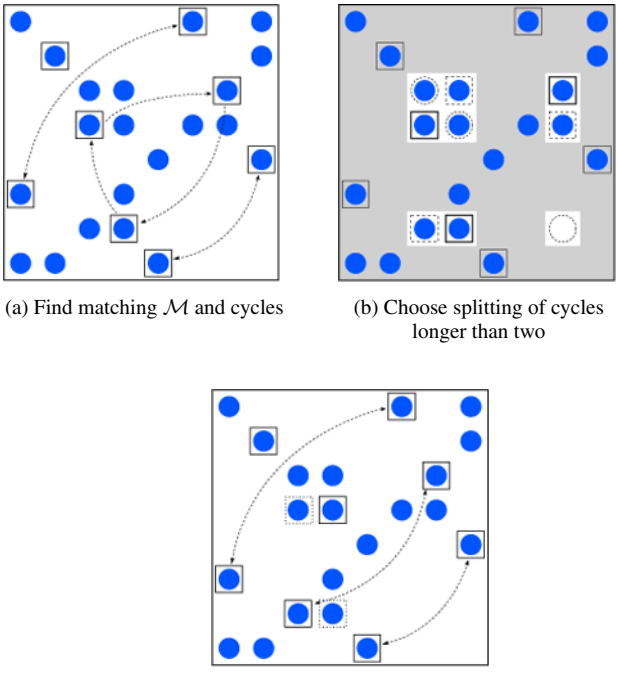
Fig. 3 **a** Symmetric matching $\mathcal{M}s$ with the 2-cycles $\{(1, 6)(6, 1)\}$, $\{(3, 7)(7, 3)\}$ and $\{(5, 8)(8, 5)\}$. **b** Reordered matrix. **c** Compressed graph

cost effective, because the compressed graph is only of dimension n, which often significantly reduces the runtime of METIS [25].

Duff and Pralet [13] showed that the problem of finding symmetric maximum weight matchings in a symmetric matrix is equivalent to finding a non-bipartite matching. Since these general matching algorithms are much more expensive than bipartite matching algorithms, the symmetric maximum weight matchings are typically only approximated. In the following we describe two such approximation techniques.

4.1 Matching variant: complete matching in K and symmetrization

The idea introduced in [11] is to determine a bipartite maximum weight matching of the matrix, and to symmetrize it based on the cycle structure of the matrix. This is necessary, because maximum weight matchings in symmetric bipartite graphs are in general not symmetric. In terms of the cycle structure, a matching $\mathcal{M}$ is symmetric, iff it only contains cycles of length one or two. Longer cycles are composed of entries that do not have "symmetric" counterparts. This is illustrated in Fig. 4. The matching $\mathcal{M}$ contains a cycle of length three with the entries $((3,7),(7,4),(4,3))$. There are three possibilities to split this cycle into a 1-cycle and a 2-cycle. This corresponds to choosing one diagonal entry in the matrix: $(3,3)$, $(4,4)$ or $(7,7)$. In Fig. 4, $(4,4)$ was chosen as the singleton cycle, and entry $(7,3)$ is included into the matching to make it symmetric. See [13, 38] for a more detailed examination of splitting approaches.



(a) Find matching $\mathcal{M}$ and cycles

(b) Choose splitting of cycles longer than two

(c) Symmetrized matching

Fig. 4 After a matching is found, the cycle structure is examined, and cycles of length longer than two are split. The *gray areas* are not accessed in the respective step

The matching algorithm also provides, as a byproduct, a row scaling D_r and a column scaling D_c. These vectors are the dual variables from the corresponding minimization problem of the maximum product matching. The scaling can be symmetrized if the matrix is symmetric:

$$D_s = \sqrt{D_r \cdot D_c}. \tag{4.2}$$

With this scaling, both the columns and the rows have unit infinity-norm. Furthermore, all matched entries are guaranteed to have an absolute value of one:

$$\left|(D_s K D_s^T)_{ij}\right| \text{ is } \begin{cases} = 1 & \text{if } (i,j) \in \mathcal{M}, \\ \leq 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

This kind of scaling is considered optimal for factorizations [32].

The downside of this combinatorial approach is that the maximum weight bipartite matching can be relatively expensive. The runtime of the algorithm is in the order of $\mathcal{O}(n_K(\tau + n_K)\log n_K)$, where τ is the number of non-zero entries in the matrix. Although, in practice it has been observed that it behaves more as $\mathcal{O}(n_K)$ [38]. However, depending on the structure of the matrix, the matching can dominate the runtime of the linear solution step. Therefore we examine another approach, which determines an approximate symmetric matching by taking the symmetric block structure of the problem into account.

4.2 Matching variant: constraint matching in A^T

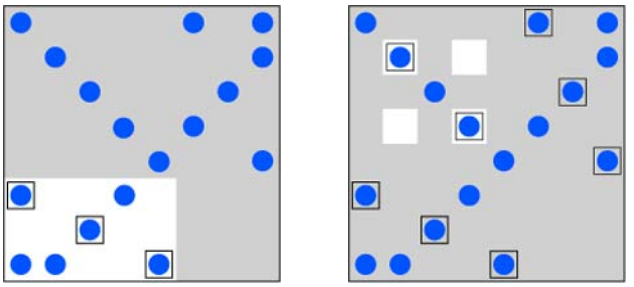
If we focus on finding favorable 2×2 diagonal pivots and assume that only diagonal entries are matched in H, we can approximate a global symmetric maximum weight matching by determining a *row-perfect* maximum weight matching in the constraint block A^T only. Expanded to the whole matrix, the corresponding entries in A are matched as well, and the matching is completed with diagonal entries from H. See Fig. 5 for an illustration. With the notation $\mathcal{M}(i) = j$, if $(i,j) \in \mathcal{M}$, and using $\mathcal{M}_A$ to denote the maximum product matching in A^T, we can write

$$\mathcal{M}s(i) := \begin{cases} j+n & \text{if } (j,i) \in \mathcal{M}_A, \\ j & \text{if } (i-n,j) \in \mathcal{M}_A, \\ i & \text{otherwise.} \end{cases}$$

$P_{\mathcal{M}}$ is then constructed as in the previous section. This approach yields a good selection of 2×2 pivots and nicely scaled blocks A and A^T (as in (4.3)), but the scaling of the entries in H is as yet not accounted for. This is achieved by scaling A^T before the matching.

In the following we denote the scaling vectors acquired by the matching of A^T as $u_A \in \mathbb{R}^m$ for the row scaling, and $v_A \in \mathbb{R}^n$ for the column scaling for A^T. The global scaling is denoted as D_s as in (1.4). The individual entries of these scaling vectors are identified in parentheses. Before we determine a maximum product matching in A^T, we scale its columns by the reciprocals of the infinity-norms of the respective columns in H, which are larger than one:

$$\hat{v}_A(i) = 1/\max\left(1, \|H(:,i)\|_\infty\right), \quad i = 1, \ldots, n.$$



(a) Match rows in scaled constraint block A^T

(b) Expand matching in A and unmatched rows and columns of H

Fig. 5 Steps for constraint matching. The gray areas are not referenced in the respective step

This ensures that all values in the scaled matrix have a magnitude of at most one. Since all matched entries in A^T are scaled to one, too, and the column scalings v_A of unmatched columns are not decreased by the matching algorithm, this also ensures that all matched entries in H are scaled to one (see condition (4.3)), if we define

$$D_s(i) = \begin{cases} v_A(i)\hat{v}_A(i) & \text{for } i \le n, \\ u_A(i-n) & \text{for } i > n. \end{cases}$$

In the actual implementation, v_A and $\hat{v}_A$ are the same vector, and v_A is changed during the matching process.

By matching in A^T it is possible to detect certain structural rank deficiencies, which can, for example, stem from redundant constraints in the optimization problem formulation. This information could potentially be helpful for the determination of δ_c, or for the complete elimination of the respective constraint conditions, but as yet we have not investigated these possibilities.

5 Numerical experiments

In this section we present a number of numerical results to explore the robustness and efficiency of different direct linear solvers within the interior point code IPOPT. The solvers used for the comparisons is the PARDISO solver with the different ordering strategies, as well as the Harwell routines MA27 and MA57 (Version 3.0) [10]. The Harwell solvers implement the threshold Duff–Reid factorization [14] for indefinite symmetric matrices, and provide the inertia of the factorized matrix within the possible numerical accuracy exactly.

We used the default parameters for the Harwell routines, except that we choose the pivot tolerance to be $\epsilon_{\text{piv}} = 10^{-8}$ to reduce the fill-in otherwise generated by the default value $\epsilon_{\text{piv}} = 10^{-2}$. With this small tolerance, the provided solution is usually sufficiently accurate. However, if, during the iterative refinement applied to the non-symmetric primal–dual system (2.2), IPOPT detects that the solution is not accurate enough, the pivot tolerance is step-wise increased (up to at most 10^{-4}) and used until the end of the optimization (for details, see Sect. 3.10 in [41]). No prior scaling of the

Table 1 Overview and abbreviations for the tested solvers

MA27	Initial $\epsilon_{\text{piv}} = 10^{-8}$, default parameters
MA57	Initial $\epsilon_{\text{piv}} = 10^{-8}$, default parameters, Version 3.0
PARDISO1	SBK pivoting, complete matching
PARDISO2	SBK pivoting, preselected 2×2 pivots, complete matching
PARDISO3	SBK pivoting, preselected 2×2 pivots, constraint matching

linear systems (e.g., using an equilibration method such as those implemented in the Harwell routines MC19 or MC29) is performed.

The PARDISO code is used in IPOPT as follows: At the beginning of the optimization the scaling matrix D_s and the reorderings P and $P_{\mathcal{M}}$ are computed based on the symmetric matching. At a later point, whenever PARDISO perturbs pivots during the Supernodal-Bunch–Kaufman factorization, the reorderings and scalings are recomputed using the new matrix values, and the matrix is factorized again. This is a heuristic to keep the inertia estimates accurate.

In order to compare the effects of the proposed approaches, we examine the solvers and solver variants listed in Table 1.

5.1 Standard NLP test sets

For the first set of experiments we ran the IPOPT algorithm[1] with the linear solvers MA27, MA57, and with the factorization algorithm in PARDISO Version 2.2, with default options, on two standard test sets for nonlinear optimization.

The first test set consists of 721 problems from the CUTE [5] collection, as provided by Benson [1] in the AMPL modeling language [20]. Here, we omitted 16 problems which are unbounded, infeasible, or have too few degrees of freedom. The size of the problems varies between 1 and 50000 variables (including slack variables for reformulated inequality constraints) and between 0 and 14000 constraints. The second test consists of 65 COPS [4] problems (Version 3.0).[2] Those problems have 150 to 20496 variables, and 0 to 20098 constraints. We note here that the AMPL preprocessor was disabled for the CUTE problems, and enabled for the COPS problems.

The results were obtained on a AMD dual-Opteron 2.2 GHz PC running Linux. All codes were compiled using GCC and GFortran version 4.0.0. The main goal of these experiments is to assess the robustness of the different PARDISO options; in particular, we wanted to explore whether the inertia information provided by the Supernode-Bunch–Kaufman factorization in PARDISO is sufficiently exact and can be used in nonconvex optimization.

The performance in terms of robustness for the CUTE problems is summarized in Table 2. Detailed inspection of the results shows that the number of iterations is in most cases identical for the different linear solvers, and that PARDISO2 and PARDISO3 in general provide very similar inertia information compared to the Harwell

[1] Development version of the C++ implementation, as of November 9, 2005.

[2] We excluded the first instance of the `tetra` example since the gradient of the objective function could not be evaluated at the starting point provided by AMPL.

Table 2 Number of problems solved using different linear solvers for CUTE problems

Solver	Solved	Exceeded 30 min time limit	Failed
MA27	694	5	22
MA57	696	1	24
PARDISO1	689	1	31
PARDISO2	696	1	24
PARDISO3	693	1	27

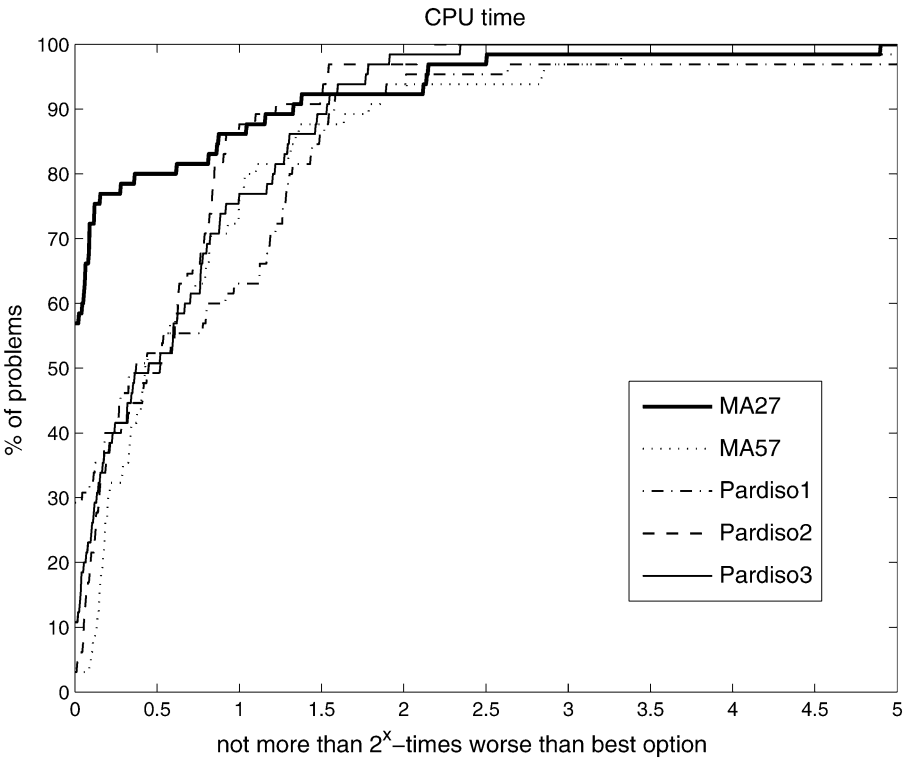


Fig. 6 Performance plot comparing runtimes on COPS problems

codes. As expected, the quality of the required inertia information of PARDISO is increased, when the "all preselected 2×2" pivots are enforced. The constraint matching (PARDISO3) appears to yield almost as good results as the full matching approach. With the exception of the PARDISO1 approach, all solvers seem to be comparably robust. Since more than 60% of the CUTE problems are solved in less than 0.1 CPU seconds, we do not present a comparison of the runtimes. Instead, we assess the runtime behavior using the COPS problems.

All of the COPS problems could be solved within the time limit of 30 CPU minutes by MA27, PARDISO2 and PARDISO3. The comparison of the runtimes is presented in Fig. 6, using Dolan–Moré performance profiles [8]. Those profiles compare the relative performance of the individual options for each problem to the option that did

best for this problem. For example, the left-most position of the curve tells us in what percentage of problems the considered option had the best runtime (e.g., MA27 was best in about 57% of the problems). Further to the right we can see how far behind an option is (e.g., in about 55% of the problems, PARDISO1 was not more than $2^1 = 2$ times worse than the best solver for each individual instance). Finally, the right-most position of the curve indicates the robustness of an option (e.g., PARDISO1 could solve about 97% of the problems).

As we can see, using the Supernodal-Bunch–Kaufman factorization from Sect. 3 combined with weighted matchings orderings from Sect. 4, the PARDISO code (options 2 and 3) works slightly more efficiently than MA57, and the older MA27 subroutine performs best. We want to stress the following observations from the CUTE and COPS experiments. Firstly, MA27 is still surprisingly effective on both test sets. The primarily reason is that the saddle-point matrices from these test sets are relatively small and the overwhelming majority of the matrices can be solved within a second of factorization time. Secondly, even though the PARDISO code has no mathematical guarantee that it is able to compute the inertia of the factorized matrix exactly, the implemented heuristics using the options 2 and 3 usually seems to provide very good information.

However, closer examination of the results shows that NLPs, which are degenerate so that the iteration matrix is singular in every iteration, are not always handled well, because the heuristics implemented in IPOPT to detect such structural degeneracy rely on the detection of singularity of the matrix by the linear solver. Since PARDISO always perturbs zero and small pivots, it is not able to detect this situation.

5.2 Optimal control problems with discretized PDEs

In order to assess the efficiency of the linear solvers for large-scale optimization problems, we applied the IPOPT algorithms to optimal control problems based on partial differential equations, which were discretized to obtain an NLP formulation of the form (1.1). The size of the resulting optimization problem can be varied by changing the number of grid points in the discretized domain.

The problems we considered are the eight boundary control examples from [28] (`Bndry1–Bndry8`) and the six distributed control examples from [29] (`Dist1–Dist6`). All involve a two-dimensional elliptic PDE as the constraint. We also considered the four examples from [31] that involve a two-dimensional parabolic PDE (`Para1–Para4`);[3] see the references for a detailed description of the problems and the discretized NLP formulation. The problems were originally implemented in AMPL by Mittelmann [30]; however, since we wanted to solve problems in size beyond the capabilities of AMPL, we reimplemented the NLP formulation in C++. In contrast to the original AMPL formulation, the objective functions are multiplied by $1/h^2$, where $h = 1/(N + 1)$ is the mesh size; without this scaling, the components of the objective function gradient are $\mathcal{O}(h^2)$, which leads to a badly scaled NLP formulation for large N. Table 3 shows the number of variables and constraints as a function of the number N of discretization points (per dimension). The size of the

[3] Those correspond to examples 5.1, 5.2.I, 5.2.II, 5.2.III in [31].

Table 3 Size of NLP formulation for PDE control problems as function of discretization parameter N

Problem name	Number of variables with upper bounds	Number of variables with lower and upper bounds	Number of equality constraints
`Bndry1–4`	N^2	$4N$	N^2
`Bndry5–8`	N^2+4N	$4N$	N^2+4N
`Dist1–3`	N^2	N^2	N^2
`Dist4–5`	N^2+4N	N^2	N^2+4N
`Dist6`	0	$2N^2+4N$	N^2+4N
`Para1–3`	0	N	N^2+N
`Para4`	0	N^2+2N	N^2+N

matrix in the linear system (2.6) for a problem is the sum of all numbers in a row of the table.[4]

These discretized control problems were solved with IPOPT, using the adaptive barrier parameter strategy, which chooses a different value for μ in every iteration, based on a quality-function (see [34] for details). As demonstrated in [34], this option is usually able to reduce the number of iterations of the optimization algorithm compared to the default monotone Fiacco–McCormick approach, at the price of some additional computational work per iteration. Since in the considered problems the factorizations of the linear system constitute the main part of the computation, this strategy reduced the overall computation time.

In the first set of experiments we compare the performance of the Harwell subroutines with PARDISO, using the method PARDISO1, PARDISO2 and PARDISO3. Here, we use the pivot tolerance $\epsilon_{\text{piv}} = 10^{-4}$ for MA27 and MA57 in all runs, since the IPOPT default of 10^{-8} turned out to be too small, and lead to an increased runtime due to inaccurate steps.

We solved all 18 problems, each for the sizes $N = 100, 200, 300, 400$, using the same computer as for the previous results. The comparison in terms of CPU time is presented in form of performance profiles in Fig. 7. More detailed information about the CPU times is given in Table 4, where the average CPU times for each problem class and size is listed.

For most of these problems, PARDISO is clearly the fastest linear solver. Even for these large problems, the PARDISO2 and PARDISO3 options perform very similarly, though. This is due to the fact that these problems have very regular structure and even the maximum weighted matching on the largest matrices is typically completed in a few seconds. Furthermore, in its current implementation, the PARDISO3 option incurs an additional transpose operation on A. In the future, we plan to evaluate these approaches on more irregular problems.

[4]Note that for some of the optimal control problems the Hessian matrix W_k in (2.6) has very few non-zero elements, and therefore the saddle-point matrix (2.6) can become very ill-conditioned at the end of the optimization process, similarly as for interior-point methods for linear optimization. For example, in `Para4`, the number of non-zero elements in W_k is only $3\sqrt{n}$.

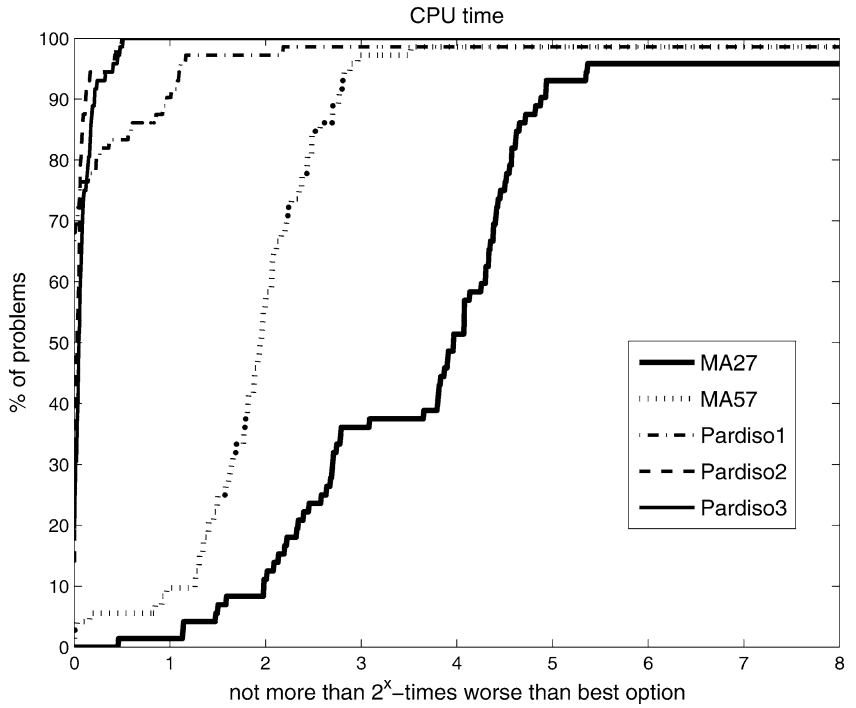


Fig. 7 Performance plot comparing runtimes on PDE control problems

Table 4 Average CPU seconds for each problem class and size

Problem	N	MA27	MA57	PARDISO1	PARDISO2	PARDISO3
Bndry	100	13.46	6.39	2.23	2.29	2.42
Bndry	200	213.59	59.72	14.22	14.72	15.46
Bndry	300	956.40	174.85	44.57	46.69	47.82
Bndry	400	2803.61	465.82	103.92	105.00	108.15
Dist	100	14.49	8.72	4.12	4.09	4.11
Dist	200	342.09	82.24	26.87	26.41	26.48
Dist	300	1322.3	363.55	79.05	78.20	78.24
Dist	400	3578.52	1024.55	185.40	179.37	177.12
Para	100	11.11	9.29	6.65	4.70	4.70
Para	200	171.44	144.05	42.63	33.25	33.42
Para	300	1915.97	593.77	138.39	112.77	114.00
Para	400	8666.20[a]	4096.36[a]	401.69[a]	259.94	261.67

[a]One problem was not solved within the time limit of 4 hours

5.2.1 Parallel solution

In our final experiments we explore the parallel scale-up of the PARDISO code (only using the PARDISO2 method), solving the discretized PDE optimal control problems for instances with several million variables. The reader is referred to [37, 39] for a detailed discussion of the parallel numerical factorization and solution algorithms in PARDISO. All results were obtained on an 8-way IBM computer with 1.45 GHz Power 4+ processors and 32 GB of memory, running AIX 5.1. From each problem class we picked the first instance, and used the same options for IPOPT as before. The results are shown in Table 5. The table shows the number of grip points N for each problem, the fraction of runtime for the linear solver PARDISO compared to IPOPT, and the numbers of processors available to the direct solver. The one entry in the table marked with a * has unexpectedly large wall clock time, because the memory requirement were so large that swapping occurred.

We see that the speed-up for 2 CPUs is on average 1.67, we have 2.65 for 4 CPUs, and 3.53 for 8 CPUs. The `Para1` example scales best, with a speed-up of up to 5.2 for 8 CPUs.

We note that the largest problem instance solved had $n = 12,500,000$ variables and $m = 6,250,000$ constraints. It was solved in about 2.7 hours with 4 processors.

Table 5 Wall clock time (in minutes) and speedup (in brackets) for parallel performance

Problem	N	% time in PARDISO	Number of CPUs			
			1	2	4	8
`Bndry1`	500	94.24	2.98	2.03 (1.5)	1.54 (1.9)	1.34 (2.2)
`Bndry1`	750	95.49	9.12	5.82 (1.6)	4.10 (2.2)	3.53 (2.6)
`Bndry1`	1000	96.01	20.50	12.63 (1.6)	8.61 (2.4)	7.08 (2.9)
`Bndry1`	1500	96.97	67.02	40.19 (1.7)	26.71 (2.5)	20.98 (3.2)
`Bndry1`	2000	97.62	149.03	85.03 (1.8)	52.98 (2.8)	40.48 (3.7)
`Bndry1`	2500	97.77	284.05	163.02 (1.7)	100.07 (2.8)	71.90 (4.0)
`Dist1`	500	87.58	4.51	3.10 (1.5)	2.33 (1.9)	2.06 (2.2)
`Dist1`	750	89.75	13.12	8.56 (1.5)	6.08 (2.2)	5.16 (2.5)
`Dist1`	1000	91.94	29.87	18.50 (1.6)	12.59 (2.4)	10.30 (2.8)
`Dist1`	1500	93.95	93.00	55.17 (1.7)	35.81 (2.6)	28.11 (3.3)
`Dist1`	2000	95.55	238.00	139.08 (1.7)	89.07 (2.7)	69.00 (3.4)
`Dist1`	2500	96.13	428.05	252.02 (1.7)	161.00 (2.7)	237.05* (1.8)
`Para1`	500	97.14	4.61	2.72 (1.7)	1.67 (2.8)	1.25 (3.7)
`Para1`	750	97.73	14.93	8.43 (1.8)	5.03 (3.0)	3.61 (4.1)
`Para1`	1000	97.89	30.21	18.27 (1.7)	10.12 (3.0)	7.00 (4.3)
`Para1`	1500	98.26	104.02	56.80 (1.8)	32.41 (3.1)	22.01 (4.7)
`Para1`	2000	98.69	267.05	147.05 (1.8)	80.08 (3.3)	53.03 (5.0)
`Para1`	2500	98.76	474.03	255.08 (1.9)	140.05 (3.4)	92.03 (5.2)

6 Conclusion

We presented an analysis of direct solution methods based on restricted pivoting and combinatorial preprocessing for the solution of large-scale saddle-point problems stemming from interior-point optimization methods. We integrated the direct solver PARDISO into the IPOPT optimization package and investigated several preprocessing and pivoting strategies.

The results indicate that the combination of the preprocessing step based on weighted symmetric matchings, and the static factorization with Supernodal-Bunch–Kaufman pivoting provides sufficient accuracy both in terms of inertia information (an important requirement for solving nonconvex nonlinear optimization problems), as well as accuracy of the solution. By combining these approaches, however, the performance could be increased by an order of magnitude. Furthermore, the static scheduling allows for an efficient parallelization of the method, yielding speed-ups of up to a factor 5.2 on 8 CPUs.

The combinatorial preprocessing turns out to be very cost effective, and by leveraging the inherent block structure of the problem, we could further reduce the worst case complexity of the matching algorithms, without compromising the accuracy.

Let us briefly recall the main ingredients necessary for this progress: At the heart of the approach lies the use of symmetric matchings in the factorization stage of the Supernodal-Bunch–Kaufman method. The method itself is complementary to the often used threshold pivoting strategies. Furthermore, the approximate constraint matching approach is also of importance for the computation of orderings and scalings at a manageable cost. And last, we emphasize that these results, of course, reflect our selected problem class: to compute the factorization and the inertia for highly indefinite symmetric matrices defined by an interior point method for nonconvex nonlinear programming.

Acknowledgements The authors thank Andrew Conn for valuable suggestions and Iain Duff for a temporary license from HSL. In addition, we would also like to thank the referees for providing constructive criticism which improved both presentation and content of the paper.

References

1. Benson, H.Y.: AMPL formulation of CUTE models. See http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/cute/
2. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**, 1–137 (2005)
3. Bergamaschi, L., Gondzio, J., Zilli, G.: Preconditioning indefinite systems in interior point methods for optimization. Comput. Optim. Appl. **28**, 149–171 (2004)
4. Bondarenko, A.S., Bortz, D.M., Moré, J.J.: COPS: Large-scale nonlinearly constrained optimization problems. Technical Report ANL/MCS-TM-237, Argonne National Laboratory, Argonne, USA (1998, revised October 1999)
5. Bongartz, I., Conn, A.R., Gould, N.I.M., Toint, P.L.: CUTE: Constrained and unconstrained testing environment. ACM Trans. Math. Software **21**, 123–160 (1995)
6. Bunch, J.R., Kaufman, L.: Some stable methods for calculating inertia and solving symmetric linear systems. Math. Comput. **31**, 163–179 (1977)
7. Byrd, R.H., Hribar, M.E., Nocedal, J.: An interior point algorithm for large-scale nonlinear programming. SIAM J. Optim. **9**, 877–900 (1999)

8. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
9. Duff, I.S.: Algorithm 575: permutations for a zero-free diagonal [F1]. ACM Trans. Math. Software **7**, 387–390 (1981)
10. Duff, I.S.: MA57—A new code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Software **30**(2), 118–144 (2004)
11. Duff, I.S., Gilbert, J.R.: Maximum-weighted matching and block pivoting for symmetric indefinite systems. In: Abstract book of Householder Symposium XV, pp. 73–75 (17–21 June 2002)
12. Duff, I.S., Koster, J.: The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM J. Matrix Anal. Appl. **20**, 889–901 (1999)
13. Duff, I.S., Pralet, S.: Strategies for scaling and pivoting for sparse symmetric indefinite problems. SIAM J. Matrix Anal. Appl. **27**(2), 313–340 (2005)
14. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. Math. Software **9**, 302–325 (1983)
15. El-Bakry, A.S., Tapia, R.A., Tsuchiya, T., Zhang, Y.: On the formulation and theory of the Newton interior-point method for nonlinear programming. J. Optim. Theory Appl. **89**, 507–541 (1996)
16. Fiacco, A.V., McCormick, G.P.: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. Wiley, New York (1968). Reprinted by SIAM (1990)
17. Forsgren, A., Gill, P.E.: Primal-dual interior methods for nonconvex nonlinear programming. SIAM J. Optim. **8**, 1132–1152 (1998)
18. Forsgren, A., Gill, P.E., Griffin, J.D.: Iterative solution of augmented systems arising in interior methods. Technical Report NA-05-03, University of California, San Diego (2005)
19. Forsgren, A., Gill, P.E., Wright, M.H.: Interior methods for nonlinear optimization. SIAM Rev. **44**, 525–597 (2002)
20. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. Thomson, Danvers (1993)
21. Gould, H.S.D.N.I.M., Schilders, W.H.A., Wathen, A.J.: On iterative methods and implicit-factorization preconditioners for regularized saddle-point systems. Technical Report RAL-TR-2005-011, Rutherford Appleton Laboratory (2005). SIMAX (to appear)
22. Gould, N.I.M., Hu, Y., Scott, J.A.: A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. Technical Report RAL-TR-2005-005, Rutherford Appleton Laboratory (2005, to appear)
23. Gould, N.I.M., Orban, D., Sartenaer, A., Toint, P.L.: Superlinear convergence of primal–dual interior point algorithms for nonlinear programming. SIAM J. Optim. **11**, 974–1002 (2001)
24. Gupta, A., Ying, L.: On algorithms for finding maximum matchings in bipartite graphs. Technical Report RC 21576 (97320), IBM T.J. Watson Research Center, Yorktown Heights (25 October 1999)
25. Hagemann, M., Schenk, O.: Weighted matchings for preconditioning symmetric indefinite linear systems. SIAM J. Sci. Comput. **28**, 403–420 (2006)
26. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1998)
27. Liegmann, A.: Efficient solution of large sparse linear systems. Ph.D. thesis, ETH Zürich (1995)
28. Maurer, H., Mittelmann, H.D.: Optimization techniques for solving elliptic control problems with control and state constraints. part 1: Boundary control. Comput. Optim. Appl. **16**, 29–55 (2000)
29. Maurer, H., Mittelmann, H.D.: Optimization techniques for solving elliptic control problems with control and state constraints. part 2: Distributed control. Comput. Optim. Appl. **18**, 141–160 (2001)
30. Mittelmann, H.D.: AMPL models. See ftp://plato.la.asu.edu/pub/ampl_files/
31. Mittelmann, H.D.: Sufficient optimality for discretized parabolic and elliptic control problems. In: Hoffmann, K.-H., Hoppe, R., Schulz, V. (eds.) Fast Solution of Discretized Optimization Problems. Birkhäuser, Basel (2001)
32. Neumaier, A.: Scaling and structural condition numbers. Linear Algebra Appl. **263**, 157–165 (1997)
33. Ng, E., Peyton, B.: Block sparse Cholesky algorithms on advanced uniprocessor computers. SIAM J. Sci. Comput. **14**, 1034–1056 (1993)
34. Nocedal, J., Wächter, A., Waltz, R.A.: Adaptive barrier strategies for nonlinear interior methods. Technical Report RC 23563, IBM T.J. Watson Research Center, Yorktown Heights, USA (March 2005)
35. Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York (1999)
36. Olschowka, M., Neumaier, A.: A new pivoting strategy for Gaussian elimination. Linear Algebra Appl. **240**, 131–151 (1996)
37. Schenk, O., Gärtner, K.: Two-level scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. Parallel Comput. **28**, 400–441 (2002)

38. Schenk, O., Gärtner, K.: On fast factorization pivoting methods for symmetric indefinite systems. Electr. Trans. Numer. Anal. **23**, 158–179 (2006)
39. Schenk, O., Gärtner, K., Fichtner, W.: Efficient sparse LU factorization with left–right looking strategy on shared memory multiprocessors. BIT **40**, 158–176 (2000)
40. Ulbrich, M., Ulbrich, S., Vicente, L.N.: A globally convergent primal–dual interior-point filter method for nonlinear programming. Math. Program. **100**, 379–410 (2004)
41. Wächter, A., Biegler, L.T.: On the implementation of a primal–dual interior-point filter line search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
42. Wächter, A., Biegler, L.T.: Line search filter methods for nonlinear programming: Motivation and global convergence. SIAM J. Optim. **16**, 1–31 (2005)