Math. Program., Ser. A (2008) 112:403–425
DOI 10.1007/s10107-006-0022-3

FULL LENGTH PAPER

Dual multilevel optimization

Timothy A. Davis · William W. Hager

Received: 17 October 2005 / Accepted: 7 July 2006 / Published online: 19 September 2006

Abstract We study the structure of dual optimization problems associated with linear constraints, bounds on the variables, and separable cost. We show how the separability of the dual cost function is related to the sparsity structure of the linear equations. As a result, techniques for ordering sparse matrices based on nested dissection or graph partitioning can be used to decompose a dual optimization problem into independent subproblems that could be solved in parallel. The performance of a multilevel implementation of the Dual Active Set Algorithm is compared with CPLEX Simplex and Barrier codes using Netlib linear programming test problems.

Keywords Multilevel optimization · Dual optimization · Dual separability · Dual active set algorithm · Parallel algorithms

Mathematics Subject Classification (2000) 90C05 · 90C06 · 65Y20

T. A. Davis
Department of Computer and Information Science and Engineering,
University of Florida, PO Box 116120, Gainesville, FL 32611-6120, USA
e-mail: davis@cise.ufl.edu
URL: http://www.cise.ufl.edu/~davis

W. W. Hager (✉)
Department of Mathematics, University of Florida,
Gainesville, PO Box 118105, FL 32611-8105, USA
e-mail: hager@math.ufl.edu
URL: http://www.math.ufl.edu/~hager

1 Introduction

We consider problems of the form

$$\sup_{\boldsymbol{\lambda}} \ \mathcal{L}(\boldsymbol{\lambda}), \tag{1}$$

where

$$\mathcal{L}(\boldsymbol{\lambda}) = F(\boldsymbol{\lambda}) + \inf_{\mathbf{x}\geq\mathbf{0}} \left(f(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{A}\mathbf{x}\right),$$
$$F(\boldsymbol{\lambda}) = \sum_{i=1}^{m} F_i(\lambda_i),$$
$$f(\mathbf{x}) = \sum_{j=1}^{n} f_j(x_j).$$

Here $F_i : \mathbb{R} \to \mathbb{R}$, $f_j : [0,\infty) \to \mathbb{R}$, and $\mathbf{A}$ is an m by n matrix. If $F_i(\lambda_i) = b_i\lambda_i, i = 1,2,\ldots,m$, for some $\mathbf{b} \in \mathbb{R}^m$, then (1) is the dual of the primal problem

$$\min f(\mathbf{x}) \ \text{ subject to } \ \mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}. \tag{2}$$

We refer to $\mathcal{L}$ as the "dual function" and to (1) as the "dual problem," regardless of how the F_i are chosen. In the case where (1) is obtained by forming the dual of an optimization problem, the f_j should be convex to ensure that the dual problem solves the primal problem. The special case $f_j(x_j) = x_j \log x_j$ corresponds to the (negative) Boltzmann–Shannon entropy function [32].

Because of the separable structure of f, the dual function can be expressed as

$$\mathcal{L}(\boldsymbol{\lambda}) = F(\boldsymbol{\lambda}) + \sum_{j=1}^{n} \ell_j(\boldsymbol{\lambda}), \tag{3}$$

where

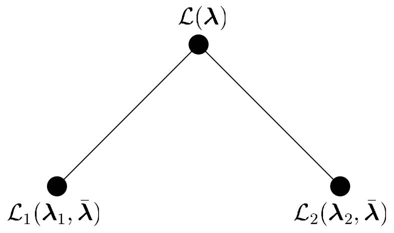
$$\ell_j(\boldsymbol{\lambda}) = \inf_{x_j\geq 0} \left(f_j(x_j) - x_j \sum_{i=1}^{m} \lambda_i a_{ij}\right). \tag{4}$$

Even though both f and F are separable, the dual cost function does not separate, in general, because each of the functions ℓ_j can depend on the entire $\boldsymbol{\lambda}$ vector. In this paper, we examine how the dual problem can be decomposed into relatively independent subproblems by exploiting the sparsity of $\mathbf{A}$.

The basic step in the decomposition process is to write

$$\mathcal{L}(\boldsymbol{\lambda}) = \bar{\mathcal{L}}(\bar{\boldsymbol{\lambda}}) + \mathcal{L}_1(\boldsymbol{\lambda}_1, \bar{\boldsymbol{\lambda}}) + \mathcal{L}_2(\boldsymbol{\lambda}_2, \bar{\boldsymbol{\lambda}}), \tag{5}$$

where $\boldsymbol{\lambda}_1$, $\boldsymbol{\lambda}_2$, and $\bar{\boldsymbol{\lambda}}$ are *disjoint subvectors* of $\boldsymbol{\lambda}$ whose union yields the entire vector $\boldsymbol{\lambda}$. That is, the components of the vectors $\boldsymbol{\lambda}_1$, $\boldsymbol{\lambda}_2$, and $\bar{\boldsymbol{\lambda}}$ are distinct com-



$\mathcal{L}(\boldsymbol{\lambda})$
$\mathcal{L}_1(\boldsymbol{\lambda}_1, \bar{\boldsymbol{\lambda}})$
$\mathcal{L}_2(\boldsymbol{\lambda}_2, \bar{\boldsymbol{\lambda}})$

Fig. 1 The dual optimization tree for the decomposition (5)

ponents of $\boldsymbol{\lambda}$, and the union of the components of $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2$, and $\bar{\boldsymbol{\lambda}}$ yields $\boldsymbol{\lambda}$. When $\mathcal{L}$ can be decomposed in this way, we can visualize the dual optimization problem using the tree shown in Fig. 1. The full dual function $\mathcal{L}(\boldsymbol{\lambda})$ is associated with the root of the tree, while $\mathcal{L}_1$ and $\mathcal{L}_2$ are associated with each leaf of the tree. If $\bar{\boldsymbol{\lambda}}$ is fixed, then the optimization of $\mathcal{L}_1(\boldsymbol{\lambda}_1, \bar{\boldsymbol{\lambda}})$ over $\boldsymbol{\lambda}_1$ is independent of the optimization of $\mathcal{L}_2(\boldsymbol{\lambda}_2, \bar{\boldsymbol{\lambda}})$ over $\boldsymbol{\lambda}_2$.

Figure 1 suggests various approaches for exploiting the structure. For example, if the optimization is done using dual coordinate ascent, we could start at the leaves of the tree and optimize over $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ while holding $\bar{\boldsymbol{\lambda}}$ fixed. The optimization over $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ can be done in parallel. Then we can move to the root of the tree and optimize over $\bar{\boldsymbol{\lambda}}$ while holding $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ fixed. After a new value of $\bar{\boldsymbol{\lambda}}$ is determined, we would drop to the leaves and again optimize over $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ with $\bar{\boldsymbol{\lambda}}$ fixed. Under suitable assumptions, the dual coordinate ascent iteration converges to a stationary point of $\mathcal{L}$ (e.g. [31, p. 228]). The decomposition (5) reveals that two of the dual coordinate ascent iterates could be computed independently.

In this paper, we develop general multilevel techniques for decomposing the dual problem into independent subproblems, with each subproblem associated with a node in a tree. Multilevel techniques have proved to be very effective for solving elliptic partial differential equations [4,16], and for solving NP-hard graph partitioning problems [25–29]. Our multilevel strategy is related to the arrowhead partitioning presented in [13]. However, instead of working towards an arrowhead form, we utilize a nested dissection ordering of $\mathbf{AA}^\mathsf{T}$ (recursive graph partitioning via node separators) [24–29], similar to the block-structured matrices in linear programming and interior point solvers discussed by [14].

We present a version of the dual active set algorithm (DASA) [10,17–22] for maximizing $\mathcal{L}$. The algorithm exploits a multilevel decomposition and moves up and down the multilevel tree in a way loosely related to that of multilevel methods for graph partitioning or multigrid methods for partial differential equations. Although the decoupled problems could be solved in parallel, we find that even for a single processor, it can be quicker to solve some problems using a multilevel approach, as opposed to the single level approach, because the subproblems are solved quickly, and there are relatively few iterations at the root node. Comparisons with CPLEX simplex and barrier codes, using Netlib linear programming (LP) test problems, are given in Sect. 7. Our implementation of LP DASA is based on our recent package CHOLMOD [6,8,9,11] for computing a Cholesky factorization and its modification after small rank changes in the matrix. CHOLMOD is partially incorporated in MATLAB 7.2 as the `chol`, `symbfact`, and `etree` functions, and in `x=A\b` when `A` is symmetric positive definite.

2 Level 1 partitioning

Given vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, define

$$\mathbf{x} \odot \mathbf{y} = \begin{cases} 0 & \text{if } x_j y_j = 0 \text{ for all } j, \\ 1 & \text{if } x_j y_j \neq 0 \text{ for some } j. \end{cases}$$

If $\mathbf{a}_i$ denotes the i-th row of $\mathbf{A}$, then $\mathbf{a}_i \odot \mathbf{a}_j = 0$ if and only if the variables in equations i and j of $\mathbf{Ax} = \mathbf{b}$ are different. In essence, equations i and j are independent of each other.

Lemma 1 *Let $\mathcal{R}_k, k = 1, 2, \ldots, N$, be disjoint subsets of $\{1, 2, \ldots, m\}$ and define*

$$\bar{\mathcal{R}} = (\mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_N)^c,$$

where the superscript "c" denotes complement. If the "orthogonality condition" $\mathbf{a}_p \odot \mathbf{a}_q = 0$ holds for all $p \in \mathcal{R}_k$ and $q \in \mathcal{R}_l$ whenever $1 \leq k < l \leq N$, then $\mathcal{L}$ can be expressed as

$$\mathcal{L}(\boldsymbol{\lambda}) = \bar{\mathcal{L}}(\bar{\boldsymbol{\lambda}}) + \sum_{k=1}^{N} \mathcal{L}_k(\boldsymbol{\lambda}_k, \bar{\boldsymbol{\lambda}}), \tag{6}$$

where $\boldsymbol{\lambda}_k, k = 1, \ldots, N$, and $\bar{\boldsymbol{\lambda}}$ are disjoint subvectors of $\boldsymbol{\lambda}$ associated with indices in $\mathcal{R}_k, k = 1, \ldots, N$, and in $\bar{\mathcal{R}}$ respectively.

Proof Define the sets

$$\mathcal{C}_k = \{j \in [1, n] : a_{ij} \neq 0 \text{ for some } i \in \mathcal{R}_k\}, \quad k = 1, 2, \ldots, N-1, \tag{7}$$

and

$$\mathcal{C}_N = \left(\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_{N-1}\right)^c. \tag{8}$$

Observe that

(a) *$\mathcal{C}_k$ and $\mathcal{C}_l$ are disjoint for $k \neq l$*: Clearly, $\mathcal{C}_N$ is disjoint from $\mathcal{C}_l$ for $l < N$ by the definition of $\mathcal{C}_N$. If $j \in \mathcal{C}_k \cap \mathcal{C}_l$ for $k < l < N$, then there exists $p \in \mathcal{R}_k$ and $q \in \mathcal{R}_j$ such that $a_{pj} \neq 0$ and $a_{qj} \neq 0$. This implies that $\mathbf{a}_p \odot \mathbf{a}_q = 1$, a contradiction.

(b) *For any l, $a_{ij} = 0$ for every $i \in (\mathcal{R}_l \cup \bar{\mathcal{R}})^c$ and $j \in \mathcal{C}_l$*: If $l < N$ and $j \in \mathcal{C}_l$, then by (7) there exists $p \in \mathcal{R}_l$ such that $a_{pj} \neq 0$. By the orthogonality

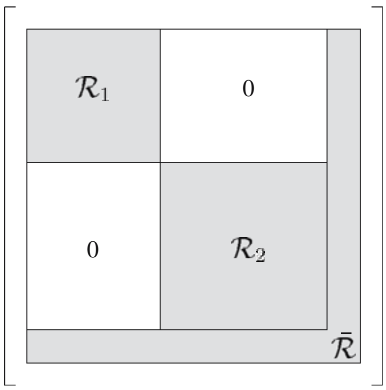


Fig. 2 Structure of the dependency matrix of **PA** in Lemma 1 for an appropriate permutation **P**; equivalently, the sparsity pattern of $(\mathbf{PA})(\mathbf{PA})^{\mathsf{T}}$

condition, $a_{ij} = 0$ whenever $i \in (\mathcal{R}_l \cup \bar{\mathcal{R}})^c$. If $l = N$ and $j \in \mathcal{C}_N$, then $j \notin \mathcal{C}_k$ for $k < N$. Since $j \notin \mathcal{C}_k$, (7) implies that $a_{ij} = 0$ for every $i \in \mathcal{R}_k$. Since

$$(\mathcal{R}_N \cup \bar{\mathcal{R}})^c = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_{N-1},$$

it follows that $a_{ij} = 0$ for every $i \in (\mathcal{R}_N \cup \bar{\mathcal{R}})^c$ and $j \in \mathcal{C}_N$.

By (a) and (8), we can group the terms of (3) in the following way:

$$\mathcal{L}(\boldsymbol{\lambda}) = F(\boldsymbol{\lambda}) + \sum_{k=1}^{N} \left(\sum_{j \in \mathcal{C}_k} \ell_j(\boldsymbol{\lambda}) \right).$$

If $j \in \mathcal{C}_k$, then by (b), the only nonzero terms $\lambda_i a_{ij}$ in (4) correspond to indices $i \in \mathcal{R}_k \cup \bar{\mathcal{R}}$. It follows that $\ell_j(\boldsymbol{\lambda})$ is independent of $\boldsymbol{\lambda}_p$ when $p \neq k$. The decomposition (6) is obtained by taking, for example,

$$\mathcal{L}_k(\boldsymbol{\lambda}_k, \bar{\boldsymbol{\lambda}}) = \sum_{i \in \mathcal{R}_k} F_i(\lambda_i) + \sum_{j \in \mathcal{C}_k} \ell_j(\boldsymbol{\lambda}), \tag{9}$$

and

$$\bar{\mathcal{L}}(\bar{\boldsymbol{\lambda}}) = \sum_{i \in \bar{\mathcal{R}}} F_i(\lambda_i). \tag{10}$$

The dependency matrix $\mathbf{D}$ associated with $\mathbf{A}$ is a symmetric matrix defined by

$$d_{ij} = \mathbf{a}_i \odot \mathbf{a}_j.$$

Under the hypotheses of Lemma 1, if we permute the rows of $\mathbf{A}$ in the order $\mathcal{R}_1$, $\mathcal{R}_2$, and $\bar{\mathcal{R}}$ and shade the nonzero parts of the dependency matrix, we obtain Fig. 2. If $\mathbf{P}$ a matrix which permutes the rows of $\mathbf{A}$ in the order $\mathcal{R}_1$, $\mathcal{R}_2$, and $\bar{\mathcal{R}}$, then the dependency matrix $\mathbf{D}$ depicted in Fig. 2 indicates the sparsity pattern

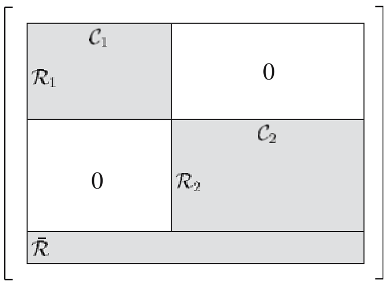


Fig. 3 The permuted matrix **PAQ** associated with Lemma 1

of $(\mathbf{PA})(\mathbf{PA})^{\mathsf{T}}$. Observe that $\mathbf{D}$ is 0 in the region corresponding to (i,j) with $i \in \mathcal{R}_1$ and $j \in \mathcal{R}_2$ or $i \in \mathcal{R}_2$ and $j \in \mathcal{R}_1$. Suppose that in addition the columns of $\mathbf{A}$ are permuted in the order $\mathcal{C}_1$ followed by $\mathcal{C}_2$. If $\mathbf{Q}$ is a column permutation matrix which puts the columns in the order $\mathcal{C}_1$ followed by $\mathcal{C}_2$, then $\mathbf{PAQ}$ has the *block-angular* structure seen in Fig. 3. By the definition of $\mathcal{C}_k$ in (7), a_{ij} is zero for $i \in \mathcal{R}_1$ and $j \in \mathcal{C}_2$ or $i \in \mathcal{R}_2$ and $j \in \mathcal{C}_1$.

As a specific illustration, let us consider the Netlib test problem TRUSS. Although this matrix, in its original form, does not have the structure seen in Fig. 2, it can be put in this form using graph partitioning codes such as CHACO [25,26] or METIS [27–29]. These heuristic methods find a small edge separator of the graph G whose adjacency matrix is $\mathbf{D}$. An edge separator is a subset of the edges of G that, when removed from G, cause it to split into two (or more) disconnected components. Ideally, the separator is a balanced one, where the two components are of equal size. Finding a minimal balanced edge separator is an NP-complete problem. When applied to $\mathbf{D}$, these codes partition the row and column indices into two sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$, chosen to minimize approximately the number of nonzeros d_{ij} with $i \in \mathcal{Q}_1$ and $j \in \mathcal{Q}_2$. The sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are roughly the same size. We then extract elements from $\mathcal{Q}_1$ or $\mathcal{Q}_2$ in order to obtain a set $\bar{\mathcal{R}}$ with the property that $d_{ij} = 0$ whenever $i \in \mathcal{Q}_1 \setminus \bar{\mathcal{R}}$ and $j \in \mathcal{Q}_2 \setminus \bar{\mathcal{R}}$. Finally, $\mathcal{R}_l = \mathcal{Q}_l \setminus \bar{\mathcal{R}}$ for $l = 1$ and 2. This step converts the edge separator found by CHACO or METIS into a vertex separator.

In LP DASA, we use the nested dissection ordering computed in our new software package CHOLMOD [6]. It uses METIS to find the node separators at each level, followed by a constrained minimum degree ordering of the whole graph (CCOLAMD). After a node separator is found, the subgraphs are themselves split, recursively, unless they consist of fewer than 200 nodes. The set of node separators found forms a tree, where each node in the tree represents one block or subgraph in the nested dissection ordering. The parent of a node (other than the root) is the subgraph consisting of the nodes in the separator. This method gives a good fill-reducing ordering, but can lead to an excessive number of blocks for multilevel LP DASA. A post-processing step collapses nodes in this separator tree, if the separator consists of more than 6% of the subgraph being processed, or if the subgraph has less than 500 nodes. The same permutation was used for the single-level LP DASA tests, but the entire tree is always collapsed into a single node. Thus, if the top-level separator consists of

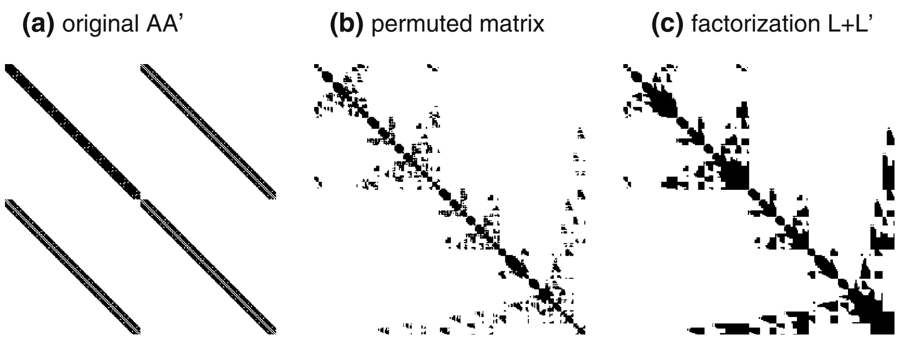


Fig. 4 Netlib test problem TRUSS

more than 6% of the nodes of the graph (or if there are fewer than 500 rows in **A**, the multilevel method collapses the tree into a single block, and the method becomes identical to single-level LP DASA.

The original matrix $\mathbf{AA}^{\mathsf{T}}$ from the TRUSS problem, the permuted matrix $\mathbf{PA}(\mathbf{PA})^{\mathsf{T}}$, and its Cholesky factorization, are shown in Fig. 4. Figure 5 shows the separator tree found by CHOLMOD, followed by the collapsed tree. In this case the collapsed tree consists of just three nodes, but in general it can be a subtree of the original tree.

3 Multilevel partitioning

In Lemma 1 we take the dual function $\mathcal{L}$ and write it as a sum of functions $\mathcal{L}_k$ given in (9). Notice that $\mathcal{L}_k$ has the same structure as $\mathcal{L}$ itself in (3); it is the sum of a separable function of $\boldsymbol{\lambda}$ (the F_l terms in (9)) and terms $\ell_j(\boldsymbol{\lambda})$, $j \in \mathcal{C}_k$. For any $j \in \mathcal{C}_k$, the nonzeros a_{ij} in column of j of **A** correspond to indices $i \in \mathcal{R}_k$ or $i \in \bar{\mathcal{R}}$. Hence, for any $j \in \mathcal{C}_k$, we have

$$\ell_j(\boldsymbol{\lambda}) = \inf_{x_j \geq 0} \left(f_j(x_j) - x_j \sum_{i \in \bar{\mathcal{R}}} \lambda_i a_{ij} - x_j \sum_{i \in \mathcal{R}_k} \lambda_i a_{ij} \right). \tag{11}$$

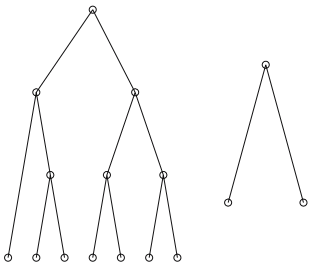


Fig. 5 Separator tree and collapsed separator tree for TRUSS

Let us view $\bar{\boldsymbol{\lambda}}$ as constant. If we identity the first two terms in (11) with the f_j term in (4) and if we identity the $\mathcal{R}_k$ sum in (11) with the sum in (4), we can apply Lemma 1 to $\mathcal{L}_k$ to decompose the dual function further.

We will carry out this recursion in the case $k = 2$. In the initial application of Lemma 1, we have

$$\mathcal{L}(\boldsymbol{\lambda}) = \bar{\mathcal{L}}(\bar{\boldsymbol{\lambda}}) + \mathcal{L}_1(\boldsymbol{\lambda}_1, \bar{\boldsymbol{\lambda}}) + \mathcal{L}_2(\boldsymbol{\lambda}_2, \bar{\boldsymbol{\lambda}}),$$

where $\boldsymbol{\lambda}_k$ denotes the components of $\boldsymbol{\lambda}$ associated with the set $\mathcal{R}_k$, and where $\mathcal{L}_k$ and $\bar{\mathcal{L}}$ are given in (9) and (10) in terms of the sets $\mathcal{C}_k$ in (7). Suppose that $\mathcal{R}_k$ is partitioned further as

$$\mathcal{R}_k = \mathcal{R}_{k1} \cup \mathcal{R}_{k2} \cup \bar{\mathcal{R}}_k,$$

where the sets $\mathcal{R}_{k1}$ and $\mathcal{R}_{k2}$ satisfy the orthogonality condition of Lemma 1. Lemma 1 is applied to (9) in the case $k = 2$. The terms that define $\mathcal{L}_k$ are grouped in the following way:

$$\mathcal{L}_k(\boldsymbol{\lambda}_k, \bar{\boldsymbol{\lambda}}) = \bar{\mathcal{L}}_k(\bar{\boldsymbol{\lambda}}_k) + \mathcal{L}_{k1}(\boldsymbol{\lambda}_{k1}, \bar{\boldsymbol{\lambda}}_k) + \mathcal{L}_{k2}(\boldsymbol{\lambda}_{k2}, \bar{\boldsymbol{\lambda}}_k), \tag{12}$$

where

$$\begin{aligned}
\mathcal{L}_{kl}(\boldsymbol{\lambda}_{kl}, \bar{\boldsymbol{\lambda}}_k) &= \sum_{i \in \mathcal{R}_{kl}} F_i(\lambda_i) + \sum_{j \in \mathcal{C}_{kl}} \ell_j(\boldsymbol{\lambda}),\\
\mathcal{C}_{k1} &= \{j \in \mathcal{C}_k : a_{ij} \neq 0 \text{ for some } i \in \mathcal{R}_{k1}\}, \quad \mathcal{C}_{k2} = \mathcal{C}_k \setminus \mathcal{C}_{k1},\\
\bar{\mathcal{L}}_k(\bar{\boldsymbol{\lambda}}_k) &= \sum_{i \in \bar{\mathcal{R}}_k} F_i(\lambda_i).
\end{aligned}$$

In the recursion (12), some components of $\boldsymbol{\lambda}_k$ form the vector $\boldsymbol{\lambda}_{k1}$ (corresponding to $\mathcal{R}_{k1}$), other components go into the vector $\boldsymbol{\lambda}_{k2}$ (corresponding to $\mathcal{R}_{k2}$), while the remaining components (corresponding to $\bar{\mathcal{R}}_k$) are added to $\bar{\boldsymbol{\lambda}}$ to form $\bar{\boldsymbol{\lambda}}_k$. Thus $\boldsymbol{\lambda}_{k1}$ and $\boldsymbol{\lambda}_{k2}$ are subvectors of $\boldsymbol{\lambda}_k$ and $\bar{\boldsymbol{\lambda}}_k$ is $\bar{\boldsymbol{\lambda}}$ augmented with the remaining components of $\boldsymbol{\lambda}_k$. When $\bar{\boldsymbol{\lambda}}_k$ is fixed in (12), the minimization of $\mathcal{L}_{k1}$ over $\boldsymbol{\lambda}_{k1}$ and the minimization of $\mathcal{L}_{k2}$ over $\boldsymbol{\lambda}_{k2}$ can be done independently.

In Fig. 3 we show the nested block-angular arrangement of nonzeros when **A** is initially partitioned (and permuted) in accordance with Lemma 1. In Fig. 6 we show the further rearrangements of nonzeros arising from the additional partitioning of $\mathcal{R}_1$ and $\mathcal{R}_2$, and $\mathcal{C}_1$ and $\mathcal{C}_2$.

If the sets of Lemma 1 were generated by graph partitioning, as we did for TRUSS in the previous section, then the dependency matrix of **A** used for the initial partitioning would be replaced by the dependency matrix for $\mathbf{A}_k$, the submatrix of **A** associated with the intersection of the rows with indices in $\mathcal{R}_k$ and columns with indices in $\mathcal{C}_k$. The partitioning process writes the initial $\mathcal{R}_k$ as a disjoint union of sets $\mathcal{R}_{kl}$, $l = 1, 2$, and $\bar{\mathcal{R}}_k$. The dependency matrix of the permuted **A** for TRUSS appears in Fig. 7a. To visualize better the structure of the

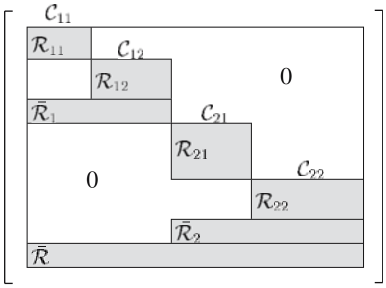


Fig. 6 The matrix in Fig. 3 after applying additional permutations to the rows and columns forming the diagonal blocks

reordered matrix, we darken the portion between the first nonzero in each row or column and the diagonal to obtain Fig. 7b, the profile of the matrix in Fig. 7a.

With these insights, we now formally define a multilevel decomposition of the dual problem. It is described by a tree $\mathcal{T}$ with r nodes labeled $1, 2, \ldots, r$ that satisfy the following four conditions:

M1. Corresponding to each node k of $\mathcal{T}$, there is a set $\mathcal{R}_k$. The root of the tree is r and $\mathcal{R}_r = \{1, 2, \ldots, m\}$.
M2. For each node k of $\mathcal{T}$, $\mathcal{R}_k$ is strictly contained in $\mathcal{R}_{\pi(k)}$, where $\pi(k)$ is the parent of k, the node directly above k in $\mathcal{T}$.
M3. For each node p of $\mathcal{T}$, $\mathcal{R}_k \cap \mathcal{R}_l = \emptyset$ whenever k and l are distinct children of p (that is, $p = \pi(k) = \pi(l)$ and $k \neq l$).
M4. For each node p of $\mathcal{T}$, $\mathbf{a}_{i_1} \odot \mathbf{a}_{i_2} = 0$ whenever $i_1 \in \mathcal{R}_k$ and $i_2 \in \mathcal{R}_l$, where k and l are distinct children of p.

In the tree of Fig. 8 associated with a 2-level decomposition, $\pi(1) = 3$ and $\pi(3) = 7$. The matrix was not ordered with CCOLAMD, so that the original structure is more visible in the decomposition in the figure. By M2 the sets $\mathcal{R}_l$ grow in size as we move from the leaves up to the root of the tree. At any level

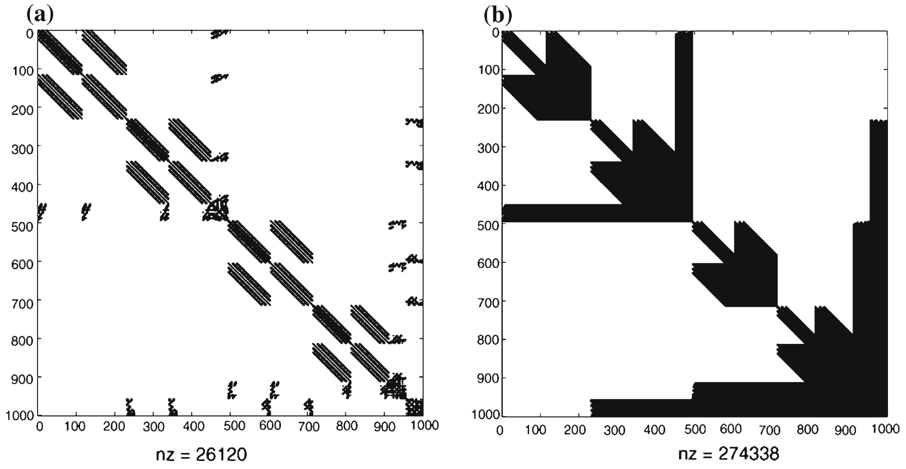


Fig. 7 **a** Sparsity structure of $(\mathbf{P}_2\mathbf{P}_1\mathbf{A})(\mathbf{P}_2\mathbf{P}_1\mathbf{A})^{\mathsf{T}}$ for TRUSS; **b** the corresponding matrix profile

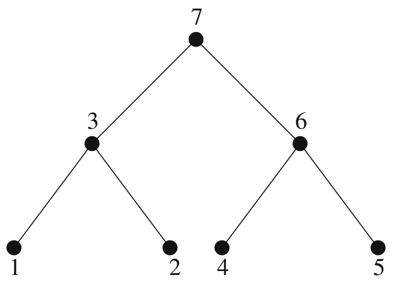


Fig. 8 Tree associated with 2-level decomposition (12)

in the tree, the sets are disjoint according to M3. The orthogonality property M4 leads to the decomposition of the dual function as described in Lemma 1.

Using the sets $\mathcal{R}_k$ associated with each node in the multilevel decomposition, we construct the following additional sets and functions:

(a) the vector $\boldsymbol{\lambda}_k$ formed from components of $\boldsymbol{\lambda}$ associated with elements of $\mathcal{R}_k$,
(b) a complementary set $\bar{\mathcal{R}}_k$ and a complementary vector $\bar{\boldsymbol{\lambda}}_k$ (defined below),
(c) the function $\mathcal{L}_k$ (also defined below).

We now define the entities introduced in (b) and (c). The complementary set $\bar{\mathcal{R}}_k$ is obtained by removing from $\mathcal{R}_k$ the sets $\mathcal{R}_c$ of its children. In other words,

$$\bar{\mathcal{R}}_k = \mathcal{R}_k \setminus \bigcup_{c\in\pi^{-1}(k)} \mathcal{R}_c.$$

In sparse matrix terminology, the complementary set $\bar{\mathcal{R}}_k$ is the separator of the children $\mathcal{R}_c$, $c \in \pi^{-1}(k)$; the equations $(\mathbf{Ax} - \mathbf{b})_i = 0$ associated with $i \in \bar{\mathcal{R}}_k$ couple together the equations associated with each of the children $\mathcal{R}_c$.

The complementary vector $\bar{\boldsymbol{\lambda}}_k$ is defined as follows: At the root, $\bar{\boldsymbol{\lambda}}_r$ is the vector with components λ_i, $i \in \bar{\mathcal{R}}_r$. At any other node $k \neq r$, $\bar{\boldsymbol{\lambda}}_k$ consists of the complementary vector $\bar{\boldsymbol{\lambda}}_{\pi(k)}$ of the parent augmented by those components of $\boldsymbol{\lambda}$ associated with the set $\bar{\mathcal{R}}_k$. Since the parent of the root $\pi(r)$ is undefined, we take $\bar{\boldsymbol{\lambda}}_{\pi(r)}$ to be empty. An equivalent description of the complementary vector $\bar{\boldsymbol{\lambda}}_k$ is the following: It consists of those components of $\boldsymbol{\lambda}$ associated with sets $\bar{\mathcal{R}}_p$ for each p on the path between k and r, the root of the tree.

Next, we write the decomposition rule (6) of Lemma 1 in a recursive fashion. At the root, $\mathcal{L}_r(\boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\lambda})$. For nodes beneath the root, the structure of the recursion is gleaned from (12). The left side of the equation contains a function $\mathcal{L}_k$ associated with node k of the tree. The right side contains functions associated with each of the children of k. The left side involves the variable $\boldsymbol{\lambda}_k$ associated with node k, and the complementary variable $\bar{\boldsymbol{\lambda}}$ associated with the parent of k (and the set $\bar{\mathcal{R}}$). The right side involves the dual variables $\boldsymbol{\lambda}_{kl}$ of the children of k, and the complementary variable $\bar{\boldsymbol{\lambda}}_k$ for node k. Hence, we have

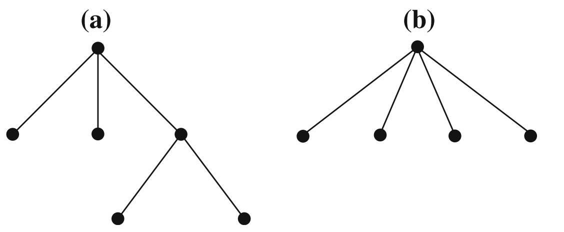


Fig. 9 Tree associated with 2-level decomposition (12); **a** $\bar{\mathcal{R}}_3 = \emptyset$, **b** $\bar{\mathcal{R}}_3 = \bar{\mathcal{R}}_6 = \emptyset$

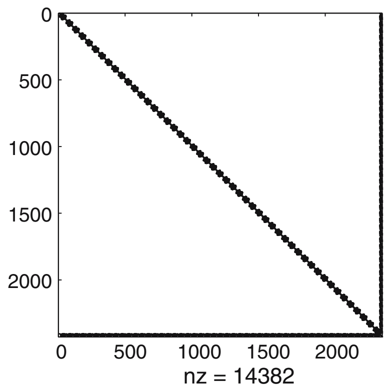


Fig. 10 Dependency matrix for KEN_07 in Netlib/lp

$$\begin{aligned} \mathcal{L}_k(\boldsymbol{\lambda}_k, \bar{\boldsymbol{\lambda}}_{\pi(k)}) &= \bar{\mathcal{L}}_k(\bar{\boldsymbol{\lambda}}_k) + \sum_{c \in \pi^{-1}(k)} \mathcal{L}_c(\boldsymbol{\lambda}_c, \bar{\boldsymbol{\lambda}}_k), \\ &= \bar{\mathcal{L}}_k(\bar{\boldsymbol{\lambda}}_k) + \sum_{c \in \pi^{-1}(k)} \mathcal{L}_c(\boldsymbol{\lambda}_c, \bar{\boldsymbol{\lambda}}_{\pi(c)}), \end{aligned} \tag{13}$$

with the initialization $\mathcal{L}_r(\boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\lambda})$.

If $\mathcal{L}$ is split into two parts as in (5), and if these parts are further subdivided into two parts, then the tree associated with the decomposition process is binary. In the special case where the complementary set is vacuous, the multilevel tree can be simplified. For example, if $\bar{\mathcal{R}}_3 = \emptyset$ in Fig. 8, then the decomposed Lagrangian is represented by the tree in Fig. 9a. If in addition $\bar{\mathcal{R}}_6 = \emptyset$ in Fig. 8, then the decomposed Lagrangian is represented by the tree in Fig. 9b. As an example, the dependency matrix of the Netlib test problem KEN_07 shown in Fig. 10 is associated with a tree that has 49 nodes beneath the root.

4 Dual dependencies

The dual functions $\mathcal{L}_i, i = 1, \ldots, r$, are not all independent of each other. For the decomposition associated with Fig. 8, $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_4$, and $\mathcal{L}_5$ are all independent of each other (assuming the complementary variables are fixed). Also, $\mathcal{L}_1$ and $\mathcal{L}_2$ are independent of $\mathcal{L}_6$ because $\mathcal{L}_1$ and $\mathcal{L}_2$ are obtained by decomposing $\mathcal{L}_3$, which is independent of $\mathcal{L}_6$. On the other hand, $\mathcal{L}_3$ is not independent of $\mathcal{L}_1$

and $\mathcal{L}_2$, even when the complementary variables are fixed, because they share common variables. In this section, we specify the independent functions in the dual decomposition.

Given a subset $\mathcal{N}$ of the nodes of $\mathcal{T}$, the *special subtree* $\mathcal{T}_{\mathcal{N}}$ is the tree obtained by pruning from $\mathcal{T}$ all descendants of the nodes $\mathcal{N}$. Hence, if $\mathcal{N} = \emptyset$, then $\mathcal{T}_{\mathcal{N}} = \mathcal{T}$; if $r \in \mathcal{N}$, then $\mathcal{T}_{\mathcal{N}} = \{r\}$. For any tree $\mathcal{T}$, we let $\partial\mathcal{T}$ denote the leaves of the tree (that is, the childless nodes).

Lemma 2 *Given a multilevel decomposition associated with a tree $\mathcal{T}$ satisfying* (13) *and a special subtree $\mathcal{T}_{\mathcal{N}}$, we have*

$$\mathcal{L}(\boldsymbol{\lambda}) = \sum_{l \in \mathcal{T}_{\mathcal{N}} \setminus \partial\mathcal{T}_{\mathcal{N}}} \bar{\mathcal{L}}_l(\bar{\boldsymbol{\lambda}}_l) + \sum_{l \in \partial\mathcal{T}_{\mathcal{N}}} \mathcal{L}_l(\boldsymbol{\lambda}_l, \bar{\boldsymbol{\lambda}}_{\pi(l)}). \tag{14}$$

Proof Suppose that (14) is violated for some special subtree $\mathcal{T}_{\mathcal{N}_0}$ of $\mathcal{T}$. Let $l_0 \in \partial\mathcal{T}_{\mathcal{N}_0}$, and let $p_0 = \pi(l_0)$ be its parent. Let $\mathcal{T}_{\mathcal{N}_1}$ be the special subtree of $\mathcal{T}$ obtained by pruning from $\mathcal{T}_{\mathcal{N}_0}$ all the children $\pi^{-1}(p_0)$. That is, $\mathcal{N}_1 = \mathcal{N}_0 \cup \{p_0\}$. Either (14) holds for $\mathcal{T}_{\mathcal{N}_1}$ and we stop the pruning process, or it is violated for $\mathcal{T}_{\mathcal{N}_1}$, in which case we choose $l_1 \in \partial\mathcal{T}_{\mathcal{N}_1}$, $p_1 = \pi(l_1)$, and $\mathcal{T}_{\mathcal{N}_2}$ is the special subtree of $\mathcal{T}$ induced by $\mathcal{N}_2 = \mathcal{N}_1 \cup \{p_1\}$. Since the decomposition (14) holds for the special subtree $\mathcal{T}_{\{r\}}$, which is contained in any special subtree, this pruning process must terminate with a special subtree $\mathcal{T}_{\mathcal{N}_t}$ for which (14) is violated, while (14) holds for $\mathcal{T}_{\mathcal{N}_{t+1}}$. Since $\mathcal{T}_{\mathcal{N}_{t+1}}$ is obtained by pruning the children of some node k from $\mathcal{T}_{\mathcal{N}_t}$, it follows from (13) that

$$\mathcal{L}_k(\boldsymbol{\lambda}_k, \bar{\boldsymbol{\lambda}}_{\pi(k)}) = \bar{\mathcal{L}}_k(\bar{\boldsymbol{\lambda}}_k) + \sum_{c \in \pi^{-1}(k)} \mathcal{L}_c(\boldsymbol{\lambda}_c, \bar{\boldsymbol{\lambda}}_{\pi(c)}). \tag{15}$$

Since (14) holds for $\mathcal{T}_{\mathcal{N}_{t+1}}$, we have

$$\mathcal{L}(\boldsymbol{\lambda}) = \sum_{l \in \mathcal{T}_{\mathcal{N}_{t+1}} \setminus \partial\mathcal{T}_{\mathcal{N}_{t+1}}} \bar{\mathcal{L}}_l(\bar{\boldsymbol{\lambda}}_l) + \sum_{l \in \partial\mathcal{T}_{\mathcal{N}_{t+1}}} \mathcal{L}_l(\boldsymbol{\lambda}_l, \bar{\boldsymbol{\lambda}}_{\pi(l)}). \tag{16}$$

Since $k \in \partial\mathcal{T}_{\mathcal{N}_{t+1}}$, it is one of the terms in the last summation in (16). After making the substitution (15) in (16), we obtain (14) in the case $\mathcal{N} = \mathcal{N}_t$. This contradicts the fact that (15) was violated for $\mathcal{N} = \mathcal{N}_t$. Hence, (14) holds for all special subtrees of $\mathcal{T}$. □

Lemma 3 *If $\mathcal{T}_{\mathcal{N}}$ is a special subtree of $\mathcal{T}$ and k and $l \in \partial\mathcal{T}_{\mathcal{N}}$, $k \neq l$, then $\mathcal{R}_k \cap \mathcal{R}_l = \emptyset$. If $\mathcal{S}_k$ is the set of indices of $\boldsymbol{\lambda}$ associated with the complementary variable $\bar{\boldsymbol{\lambda}}_k$ (equivalently, $\mathcal{S}_k$ is the union of the complementary sets $\bar{\mathcal{R}}_q$ over all nodes q on the path between k and the root), then $\mathcal{S}_k \cap \mathcal{R}_l = \emptyset$.*

Proof Consider the sequence of ancestors $\pi^p(k), p = 1, 2, \ldots$, and $\pi^q(l), q = 1, 2, \ldots$. Let $P = \pi^p(k) = \pi^q(l)$ be the first common ancestor of k and l (see

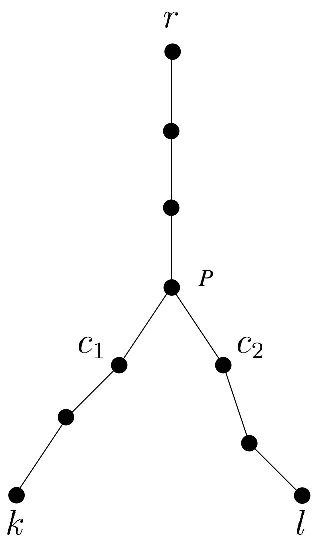


Fig. 11 P is first common ancestor of nodes k and l

Fig. 11). Since P is the first common ancestor, the children $c_1 = \pi^{p-1}(k)$ and $c_2 = \pi^{q-1}(l)$ are distinct. By M3, $\mathcal{R}_{c_1} \cap \mathcal{R}_{c_2} = \emptyset$. By M2, $\mathcal{R}_k \subset \mathcal{R}_{c_1}$ and $\mathcal{R}_l \subset \mathcal{R}_{c_2}$. Hence, $\mathcal{R}_k \cap \mathcal{R}_l = \emptyset$. By the definition of the complementary set $\bar{\mathcal{R}}_P$, it is disjoint from the sets $\mathcal{R}_c$ of each of the children $c \in \pi^{-1}(P)$. That is, $\bar{\mathcal{R}}_P \cap \mathcal{R}_c = \emptyset$ for each $c \in \pi^{-1}(P)$. Applying this result to each node between P and the root r, we conclude that $\mathcal{S}_P \cap \mathcal{R}_c = \emptyset$ for each $c \in \pi^{-1}(P)$. For each node on the path from c_1 to k, the complementary set is contained in $\mathcal{R}_{c_1}$ due to M2. Hence, $\mathcal{S}_k \subset (\mathcal{R}_{c_1} \cup \mathcal{S}_P)$. The relations

$$\mathcal{R}_{c_1} \cap \mathcal{R}_{c_2} = \emptyset, \quad \mathcal{S}_k \cap \mathcal{R}_{c_2} = \emptyset, \quad \mathcal{R}_l \subset \mathcal{R}_{c_2}, \quad \text{and} \quad \mathcal{S}_k \subset (\mathcal{R}_{c_1} \cup \mathcal{S}_P),$$

imply that $\mathcal{S}_k \cap \mathcal{R}_l = \emptyset$. □

By Lemmas 2 and 3, it follows that when $\mathcal{T}_\mathcal{N}$ is a special subtree of $\mathcal{T}$, maximizing the dual function in (14) over $\boldsymbol{\lambda}_l$ for each $l \in \partial\mathcal{T}_\mathcal{N}$, while holding the other components of $\boldsymbol{\lambda}$ fixed, decomposes into independent maximizations:

$$\max_{\boldsymbol{\lambda}_l} \ \mathcal{L}_l(\boldsymbol{\lambda}_l, \bar{\boldsymbol{\lambda}}_{\pi(l)}).$$

5 An example

We give a concrete illustration of a multilevel decomposition using the dual of a linear programming problem

$$\min \quad \mathbf{c}^\mathsf{T}\mathbf{x} \quad \text{subject to} \ \ \mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0},$$

with

$$\mathbf{A} = \begin{bmatrix} 1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1 \\ 1\,0\,1\,0\,1\,0 \end{bmatrix}. \tag{17}$$

For the tree shown in Fig. 12, it can be checked that conditions M1–M4 hold. The first level of the tree corresponds to the decomposition

$$\mathcal{L}(\boldsymbol{\lambda}) = b_5\lambda_5 + \mathcal{L}_3(\lambda_1, \lambda_2, \lambda_3, \lambda_5) + \mathcal{L}_4(\lambda_4, \lambda_5),$$

where the complementary variable is $\bar{\boldsymbol{\lambda}}_5 = \lambda_5$, and the first term $b_5\lambda_5$ corresponds to $\bar{\mathcal{L}}_5(\bar{\boldsymbol{\lambda}}_5)$. The functions $\mathcal{L}_3$ and $\mathcal{L}_4$ are given as follows:

$$\mathcal{L}_3(\lambda_1, \lambda_2, \lambda_3, \lambda_5) = \begin{cases} b_1\lambda_1 + b_2\lambda_2 + b_3\lambda_3 & \text{if } \begin{cases} c_1 \geq \lambda_1 + \lambda_3 + \lambda_5 \\ c_2 \geq \lambda_1 + \lambda_3 \\ c_3 \geq \lambda_2 + \lambda_3 + \lambda_5 \\ c_4 \geq \lambda_3 \end{cases} \\ -\infty & \text{otherwise,} \end{cases}$$

and

$$\mathcal{L}_4(\lambda_4, \lambda_5) = \begin{cases} b_4\lambda_4 & \text{if } \begin{cases} c_5 \geq \lambda_4 + \lambda_5 \\ c_6 \geq \lambda_4 \end{cases} \\ -\infty & \text{otherwise.} \end{cases}$$

$\mathcal{L}_3$ is independent of $\mathcal{L}_4$ when $\bar{\boldsymbol{\lambda}}_5$ is fixed. At nodes 3 and 4, we have $\boldsymbol{\lambda}_3 = (\lambda_1, \lambda_2, \lambda_3)$ and $\boldsymbol{\lambda}_4 = (\lambda_4)$. By the structure of $\mathcal{L}_3$, it can be further decomposed as

$$\mathcal{L}_3(\boldsymbol{\lambda}) = b_3\lambda_3 + \mathcal{L}_1(\lambda_1, \lambda_3, \lambda_5) + \mathcal{L}_2(\lambda_2, \lambda_3, \lambda_5),$$

where

$$\mathcal{L}_1(\lambda_1, \lambda_3, \lambda_5) = \begin{cases} b_1\lambda_1 & \text{if } \begin{cases} c_1 \geq \lambda_1 + \lambda_3 + \lambda_5 \\ c_2 \geq \lambda_1 + \lambda_3 \end{cases} \\ -\infty & \text{otherwise,} \end{cases}$$

and

$$\mathcal{L}_2(\lambda_2, \lambda_3, \lambda_5) = \begin{cases} b_2\lambda_2 & \text{if } \begin{cases} c_3 \geq \lambda_2 + \lambda_3 + \lambda_5 \\ c_4 \geq \lambda_3 \end{cases} \\ -\infty & \text{otherwise.} \end{cases}$$

The complementary variable at node 3 is $\bar{\boldsymbol{\lambda}}_3 = (\lambda_3, \lambda_5)$, the complementary variable at the parent of 3 augmented by the variable λ_3 associated with $\bar{\mathcal{R}}_3 = \{3\}$. $\mathcal{L}_1$

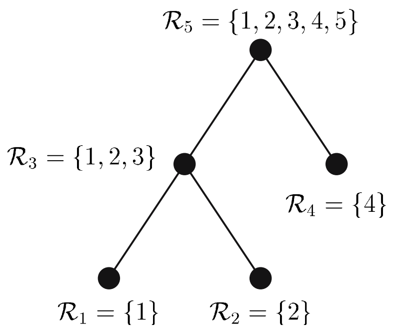


Fig. 12 Multilevel decomposition associated with (17)

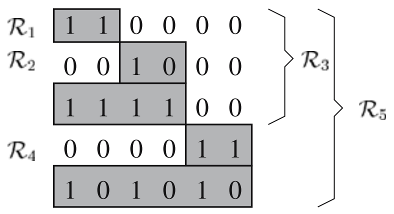


Fig. 13 Nonzero structure of **A** corresponding to multilevel partitioning tree in Fig. 12

and $\mathcal{L}_2$ are independent when $\bar{\lambda}_3$ is fixed. The nonzero structure of **A** associated with the tree of Fig. 12 is shaded in Fig. 13.

6 Multilevel DASA

It was pointed out in Sect. 1 that a multilevel decomposition could be used in various algorithms for solving the dual problem; a specific illustration was dual coordinate ascent. Here we focus on a more efficient use of a multilevel decomposition based on the dual active set algorithm (DASA). By [19, Theorem. 1], DASA solves (1) in a finite number of iterations when f is strongly convex and F is strongly concave. Given a *bound set* $B \subset \{1, 2, \ldots, n\}$, let $\mathbf{x}_B$ be the subvector of $\mathbf{x}$ consisting of those components x_i associated with $i \in B$. We define the functions

$$\mathcal{L}_{B^+}(\boldsymbol{\lambda}) = \min\,\{\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) : \mathbf{x} \in \mathbb{R}^n, \mathbf{x}_B \geq \mathbf{0}\}, \tag{18}$$

and

$$\mathcal{L}_{B^0}(\boldsymbol{\lambda}) = \min\,\{\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) : \mathbf{x} \in \mathbb{R}^n, \mathbf{x}_B = \mathbf{0}\}, \tag{19}$$

where

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) = F(\boldsymbol{\lambda}) + f(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{A}\mathbf{x}.$$

The components of $\mathbf{x}$ in the minimization problems (18) and (19) corresponding to indices in the complement of B are unconstrained. Assuming the f_j are strongly convex, there exists a unique minimizer $\mathbf{x}(\boldsymbol{\lambda}, B)$ in (18) for each choice of $\boldsymbol{\lambda}$ and B. Let $\mathbf{x}(\boldsymbol{\lambda})$ denote $\mathbf{x}(\boldsymbol{\lambda}, B)$ in the case $B = \{1, 2, \ldots, n\}$.

Algorithm 1 Dual active set algorithm

```
$s = 0$
$\boldsymbol{\lambda}_0$ = starting guess
while $\nabla \mathcal{L}(\boldsymbol{\lambda}_s) \neq \mathbf{0}$
    $\boldsymbol{\nu}_0 = \boldsymbol{\lambda}_s$
    $B_0 = \{j \in [1, n] : x_j(\boldsymbol{\lambda}_s) = 0\}$
    for $t = 0, 1, \ldots$
        $\boldsymbol{\omega} = \arg\ \max\{\mathcal{L}_{B_t^0}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \mathbb{R}^m\}$
        $\boldsymbol{\nu}_{t+1} = \arg\ \max\{\mathcal{L}_{B_t^+}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in [\boldsymbol{\nu}_t, \boldsymbol{\omega}]\}$
        $B_{t+1} = \{j \in B_t : x_j(\boldsymbol{\nu}_{t+1}, B_t) = 0\}$
        if $\boldsymbol{\nu}_{t+1} = \boldsymbol{\omega}$ break
    end
    $s = s + 1$
    $\boldsymbol{\lambda}_s = \boldsymbol{\omega}$
end
```

The statement of DASA in Algorithm 1 has an outer loop indexed by s, and an inner loop indexed by t. In the inner loop, indices are deleted from B_t to obtain B_{t+1}; that is, B_{t+1} is a subset of B_t consisting of those indices for which $x_i(\boldsymbol{\nu}_{t+1}, B_t) = 0$. When the inner loop terminates, we reinitialize B_0 by solving the problem

$$\min_{\mathbf{x} \geq \mathbf{0}} \mathcal{L}(\boldsymbol{\lambda}_s, \mathbf{x}) \tag{20}$$

to obtain $\mathbf{x}(\boldsymbol{\lambda}_s)$. Since the optimization over $\mathbf{x}$ in (20) decomposes into independent optimization over x_j as in (4), it follows that the B_0 set at the start of the iteration includes some indices that were missing from the final B_t associated with the inner loop in the previous iteration. Hence, the dual initialization step adds indices to the current bound set, while the formula for B_{t+1} in the subiteration deletes bound indices. The proof [19, Thm. 1] that DASA solves (1) in a finite number of iterations is based on the fact that the subiteration strictly increases the value of the dual function; as a result, the final set B_t generated in the subiteration cannot repeat. Since the sets B_t are chosen from a finite set, convergence occurs in a finite number of steps.

Algorithm 2 is the multilevel version of DASA. Here we replace the maximization over $\boldsymbol{\lambda}$ in the computation of $\boldsymbol{\omega}$ in Algorithm 1 by a maximization over the leaves of a special subtree. The set $\mathcal{N}_t$ in Algorithm 2 is a collection of nodes in the tree $\mathcal{T}$; these nodes essentially correspond to components of the dual variable over which the dual function has been maximized. Initially, $\mathcal{N}_0 = \emptyset$ because we have not maximized over any components of $\boldsymbol{\lambda}$. As the inner iterations progress, the set $\mathcal{N}_t$ grows in size. In each inner iteration, we maximize $\mathcal{L}_{B_t^0}$ over the leaves of the special subtree $\mathcal{T}_{\mathcal{N}_t}$. As seen after Lemmas 2 and 3, maximization over the leaves of a special subtree decomposes into the independent maximization of local dual functions, which could be done in parallel. These maximizers are utilized in local line searches that take us to a new point $\boldsymbol{\nu}_{t+1}$ with a larger value for both $\mathcal{L}_{B_t^+}$ and $\mathcal{L}$. Since the bound set B_t can only decrease in size, the line search on the interval $[\boldsymbol{\nu}, \boldsymbol{\omega}]$ eventually takes a

Algorithm 2 Multilevel dual active set algorithm

```
$s = 0$
$\boldsymbol{\lambda}_0$ = starting guess
while $\nabla\mathcal{L}(\boldsymbol{\lambda}_s) \neq \mathbf{0}$
    initialize $\boldsymbol{\nu}_0 = \boldsymbol{\lambda}_s$ and $\mathcal{N}_0 = \emptyset$
    $B_0 = \{j \in [1,n] : x_j(\boldsymbol{\lambda}_s) = 0\}$
    for $t = 0, 1, \ldots$
        initialize $\boldsymbol{\nu} = \boldsymbol{\nu}_t$ and $\mathcal{N} = \mathcal{N}_t$
        for each $l \in \partial\mathcal{T}_{\mathcal{N}_t}$
            $\boldsymbol{\omega} = \arg\max\{\mathcal{L}_{B_t^0}(\boldsymbol{\lambda}) : \lambda_i = \nu_i,\ \text{for all } i \in \mathcal{R}_l^c\}$
            $\boldsymbol{\nu} \leftarrow \arg\max\{\mathcal{L}_{B_t^+}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in [\boldsymbol{\nu}, \boldsymbol{\omega}]\}$
            if $\boldsymbol{\nu} = \boldsymbol{\omega}$, then $\mathcal{N} \leftarrow \mathcal{N} \cup \{l\}$
        end
        $\boldsymbol{\nu}_{t+1} = \boldsymbol{\nu}$ and $\mathcal{N}_{t+1} = \mathcal{N}$
        if $r \in \mathcal{N}$ break
        $B_{t+1} = \{j \in B_t : x_j(\boldsymbol{\nu}_{t+1}, B_t) = 0\}$
    end
    $s = s + 1$
    $\boldsymbol{\lambda}_s = \boldsymbol{\nu}_t$
end
```

full step to $\boldsymbol{\omega}$, in which case $\mathcal{N}$ grows by the addition of node l. That is, the step stops short of $\boldsymbol{\omega}$ only when the bound set decreases in size. If a full step is taken in the line search, then we add the associated node l to the set of completed nodes $\mathcal{N}$. Eventually, $\mathcal{N}$ includes the root r, in which case the inner iteration terminates.

The proof of finite convergence is the same given previously for DASA (e.g. [17,19,21]). That is, in the multilevel setting, we work up to the root of the tree $\mathcal{T}$ increasing the value of the dual function, and by M1, the set $\mathcal{R}_r$ contains all the indices of $\boldsymbol{\lambda}$. Hence, in the multilevel setting, the final dual subiterate again takes us to a maximizer of $\mathcal{L}_{B_k^0}$ for some set B_k, which can never repeat without contradicting the strict ascent of the dual iterates.

In Algorithms 1 and 2, we maximize $\mathcal{L}_{B_k^0}$ in each subiteration. As an alternative, we could approximately maximize $\mathcal{L}_{B_k^0}$ in each subiteration. Under suitable assumptions (see [20]), convergence is achieved in the limit, as the number of iterations tends to infinity.

7 Numerical comparisons

In this section, we apply both single level and multilevel versions of LP DASA to Netlib linear programming test problems and compare the performance to that of simplex and barrier methods as implemented in the commercial code ILOG CPLEX version 9.1.3 [2]. The test problems were solved on a 3.2 GHz Pentium 4 with 4 GB of RAM, running SUSE Linux, and the Goto BLAS [15]. Since the CPLEX Barrier code automatically applies a presolver [1] to the input problem, we use the CPLEX presolved problem as input to the codes. All

Table 1 Description of the test problems

Problem	Rows	Columns	Nonzeros
PEROLD	503	1,273	5,545
25FV47	682	1,732	10,181
NUG07	473	930	3,321
PILOT_WE	602	2,526	8,414
PILOTNOV	748	1,999	11,703
CRE_C	2,257	5,293	13,078
CRE_A	2,684	6,382	15,739
OSA_07	1,047	24,062	63,037
NESM	598	2,764	12,988
D6CUBE	402	5,467	34,332
TRUSS	1,000	8,806	27,836
FIT2P	3,000	13,525	50,284
PILOT_JA	708	1,684	11,155
NUG08	741	1,631	5,940
FIT2D	25	10,388	127,793
GREENBEB	1,015	3,211	23,042
GREENBEA	1,015	3,220	23,142
80BAU3B	1,789	9,872	20,007
QAP8	741	1,631	5,940
KEN_11	5,511	11,984	26,538
DEGEN3	1,406	2,503	25,204
PDS_06	2,972	18,530	42,304
OSA_14	2,266	52,723	139,136
D2Q06C	1,855	5,380	31,325
MAROS_R7	2,152	6,578	80,167
STOCFOR3	8,388	15,254	53,528
PILOT	1,204	4,124	41,160
OSA_30	4,279	100,396	267,149
PDS_10	4,725	33,270	76,307
KEN_13	10,962	24,818	57,238
CRE_D	3,990	28,489	86,144
CRE_B	5,176	36,222	111,434
PILOT87	1,811	6,065	71,838
OSA_60	10,209	234,334	594,462
PDS_20	10,214	81,224	184,176
KEN_18	39,867	89,439	208,594
NUG12	2,793	8,855	33,526
QAP12	2,793	8,855	33,526
DFL001	3,861	9,607	31,955
NUG15	5,697	22,274	85,468
QAP15	5,697	22,274	85,468
NUG20	14,097	72,599	281,958
NUG30	49,647	376,740	1,486,829

Netlib problems taking more than 0.1 s (based the fastest of CPLEX and LP DASA run times) are listed in Table 1. The table is sorted by this run time. These statistics are for the presolved problems used in the numerical experiments, not for the original problems on the Netlib test site.

The multilevel partitioning is computed via nested dissection of $\mathbf{AA}^{\mathsf{T}}$ using CHOLMOD Version 1.2. The run times we report for LP DASA include the time to compute the ordering, including the separator tree and its post-processing.

For LP DASA, developed in [10], we take

$$f(\mathbf{x}) = \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2,$$

where $\epsilon > 0$ is a regularization parameter and $\mathbf{y}$ is a shift. Once we achieve the maximum in the dual problem, the regularization parameter and shift are updated. The details of their adjustments are given in [10].

In coding the LP DASA multilevel algorithm, we need to factor a matrix of the form $\mathbf{A}_F\mathbf{A}_F^{\mathsf{T}}$, where F is the complement of the set B_k in Algorithm 1, and the subscript "F" denotes the submatrix of $\mathbf{A}$ associated with column indices in F. This factorization is done using a supernodal Cholesky factorization algorithm, in CHOLMOD. In the multilevel case, the rows and columns of $\mathbf{L}$ corresponding to leaves of the separator tree are factorized in a supernodal method. The nodes farther up in the tree are factorized incrementally as they are needed, using a row-oriented sparse factorization [5,30].

We refactor the matrix infrequently, and after a column is freed or a row is dropped, we use the sparse modification techniques developed in [8,9,11] to modify the factorization. Since the Cholesky factorization is computed row-by-row, it can be incrementally extended in the multilevel setting. The submatrix associated with a subtree (nodes 1, 2 and 3 in Fig. 12, for example) can be factored without having to compute the entire factorization. New rows corresponding to separators can be added to the factorization without touching the previously computed partial factorization. The codes for modifying a factorization after column or row updates take advantage of speedup associated with multiple-rank updates. As a result, a rank-16 update of a sparse matrix achieves flop rates comparable to those of a BLAS dot product, and about 1/3 the speed of a BLAS matrix-matrix multiply [12], in the experiments given in [9].

In coding LP DASA, we use a new version of COLAMD [7], which we call constrained COLAMD (CCOLAMD), to reorder the rows of $\mathbf{A}$ to minimize the fill associated with the Cholesky factorization of $\mathbf{A}\mathbf{A}^{\mathsf{T}}$. We constrain the ordering so that rows associated with a set $\mathcal{R}_k$ for a leaf of the partitioning tree remain in $\mathcal{R}_k$, where the sets $\mathcal{R}_k$ are chosen to satisfy the orthogonality condition of Lemma 1; to preserve this orthogonality property, we can only exchange rows within $\mathcal{R}_k$. Our constrained COLAMD, developed in collaboration with S. Rajamanickam, is related to the scheme developed in [3] for a sparse LU factorization.

Our starting guesses for an optimal dual multiplier and primal solution are always $\boldsymbol{\lambda} = \mathbf{0}$ and $\mathbf{x} = \mathbf{0}$ respectively. Our convergence test is based on the LP optimality conditions. The primal approximation $\mathbf{x}(\boldsymbol{\lambda})$ always satisfies the bound constraints $\mathbf{x} \geq \mathbf{0}$. The column indices are expressed as $B \cup F$ where $F = B^c$. For all $j \in B$, $x_j(\boldsymbol{\lambda}) = 0$ and $(\mathbf{c} - \mathbf{A}^{\mathsf{T}}\boldsymbol{\lambda})_j \geq 0$. Hence, the optimality conditions would be satisfied if the primal and dual linear systems, $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{A}_F^{\mathsf{T}}\boldsymbol{\lambda} = \mathbf{c}_F$, were satisfied. Our convergence criterion is expressed in terms of the relative residuals in the primal and dual systems:

$$\frac{\|\mathbf{b}-\mathbf{A}\mathbf{x}\|_\infty}{1+\|\mathbf{x}\|_\infty}+\frac{\|\mathbf{c}_F-\mathbf{A}_F^{\mathsf{T}}\boldsymbol{\lambda}\|_\infty}{1+\|\boldsymbol{\lambda}\|_\infty}\le 10^{-8}. \tag{21}$$

Here $\|\cdot\|_\infty$ denotes the maximum absolute component of a vector. Note that the test problems are given in MPS format which provides, in general, space for about five significant digits when the data is represented in floating point (exponential) format. Hence, the number of significant digits in $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ is five in general, while the relative error in (21) is 10^{-8}. Note that in certain cases, more than five significant digits can be represented in MPS format; for example, any problem where the data consists of small integers, such as a network problem, could be represented exactly using MPS format.

The CPU times for LP DASA, both single level and multilevel versions, and for CPLEX simplex and barrier codes are in Table 2. We also give the number of blocks for multilevel LP DASA. A dash means the problem cannot be solved by that method on our computer. Note that LP DASA was the only code that was able to solve NUG30. With both Barrier and Simplex, there was not enough memory. LP DASA switches to an iterative implementation when there is insufficient memory to factor the matrix. In the iterative approach, developed in [20], a conjugate gradient scheme is used to compute the solution of the quadratic programming problems as opposed to a direct method based on a Cholesky factorization. Observe that the number of blocks in the TRUSS problem is 3, which is the number of nodes in the collapsed tree in Fig. 5.

Even for a single processor, the multilevel implementation is typically at least as fast as the single level implementation. The small subproblems in the multilevel approach are solved quickly and the overall flop count (not listed) is much smaller than that of a single-level implementation. Of course in a parallel computing environment where the 324 independent subproblems associated with KEN_18 (for example) can be assigned to separate computers, significant speedup is possible.

In Table 2, we see the following: multilevel LP DASA is faster than CPLEX simplex for 20 out of 43 problems, and it is faster than CPLEX Barrier for 12 problems.

8 Summary

Sparse optimization problems with separable cost and with linear constraints and bounds can be decomposed into a multilevel structure by applying codes such as CHACO or METIS to the dependency matrix. The multilevel structure is described by a tree in which each node i is associated with a function $\mathcal{L}_i$. Those $\mathcal{L}_i$ associated with the leaves of a special subtree can be optimized independently, when the complementary variables are fixed.

In an implementation of the dual active set algorithm, the computation proceeds from the leaves of the tree up to the root, optimizing over the leaves of special subtrees along the way. Even on a single processor, the multilevel

Table 2 LP DASA (single and multilevel), CPLEX (Simplex and Barrier)

Problem	One level	Multilevel	Blocks	Simplex	Barrier
PEROLD	1.16	1.16	1	0.15	0.12
25FV47	0.18	0.18	1	0.32	0.12
NUG07	0.30	0.29	1	0.29	0.12
PILOT_WE	1.53	1.52	1	0.54	0.16
PILOTNOV	1.21	1.20	1	0.23	0.18
CRE_C	0.44	0.40	93	0.18	0.20
CRE_A	0.61	0.60	133	0.18	0.25
OSA_07	0.22	0.19	628	0.23	0.43
NESM	0.39	0.39	1	0.35	0.19
D6CUBE	3.53	3.54	1	0.20	0.31
TRUSS	1.08	0.94	3	11.62	0.21
FIT2P	0.23	0.23	1	4.46	0.43
PILOT_JA	3.72	3.70	1	0.33	0.25
NUG08	0.71	0.70	1	1.86	0.25
FIT2D	0.26	0.26	1	0.59	0.71
GREENBEB	0.59	0.57	11	1.46	0.27
GREENBEA	1.82	2.19	3	1.03	0.28
80BAU3B	1.60	1.14	49	0.28	0.41
QAP8	0.84	0.83	1	1.40	0.28
KEN_11	0.53	0.32	122	0.38	0.55
DEGEN3	0.65	0.84	3	0.57	0.43
PDS_06	1.93	1.69	23	0.50	1.81
OSA_14	0.76	0.68	1837	0.59	1.03
D2Q06C	2.70	2.70	1	2.81	0.62
MAROS_R7	0.83	0.83	1	3.29	1.37
STOCFOR3	5.24	4.33	26	1.75	1.09
PILOT	5.30	5.29	1	3.22	1.19
OSA_30	1.74	1.51	4070	1.41	2.56
PDS_10	5.23	4.65	96	1.43	6.29
KEN_13	5.57	2.72	170	1.50	1.65
CRE_D	8.37	8.33	1	1.86	2.54
CRE_B	8.63	8.34	576	4.24	3.03
PILOT87	17.69	17.73	1	22.49	3.68
OSA_60	5.10	3.76	9766	4.08	6.58
PDS_20	34.91	36.34	149	9.16	29.07
KEN_18	58.53	23.49	325	11.74	11.23
NUG12	52.86	52.99	1	202.72	11.33
QAP12	64.22	64.37	1	197.93	12.71
DFL001	36.94	37.08	1	22.51	16.04
NUG15	438.60	443.34	1	2787.11	78.99
QAP15	339.76	342.70	1	3143.31	85.83
NUG20	6094.90	6175.99	1	–	13755.03
NUG30	32206.34	32206.34	1	–	–

implementation was faster than the single level implementation. Competitive performance, as compared to CPLEX, was achieved for the test set of Table 1. Further speedup in LP DASA could be achieved by exploiting hypersparsity of the solution to the linear systems that arise during the solution process (see [23]), and by developing a recursive version of the equation dropping techniques introduced in [10].

Acknowledgements Constructive comments by the referee leading to an improved presentation are gratefully acknowledged. This material is based upon work supported by the National Science Foundation under Grant No. 0203270.

References

1. Andersen, E.D., Andersen, K.D.: Presolving in linear programming. Math. Program. **71**, 221–245 (1995)
2. Bixby, R.E.: Progress in linear programming. ORSA J. Comput. **6**, 15–22 (1994)
3. Brainman, I., Toledo, S.: Nested-dissection orderings for sparse LU with partial pivoting. SIAM J. Matrix Anal. Appl. **23**, 998–1012 (2002)
4. Bramble, J.H.: Multigrid Methods. Wiley, New York (1993)
5. Davis, T.A.: Algorithm 849: a concise sparse Cholesky factorization package. ACM Trans. Math. Softw. **31**, 587–591 (2005)
6. Davis, T.A.: CHOLMOD users' guide. University of Florida (2005). http: www.cise.ufl.edu/∼davis
7. Davis, T.A., Gilbert, J.R., Larimore, S.I., Ng, E.G.: A column approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**, 353–376 (2004)
8. Davis, T.A., Hager, W.W.: Modifying a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **20**, 606–627 (1999)
9. Davis, T.A., Hager, W.W.: Multiple-rank modifications of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **22**, 997–1013 (2001)
10. Davis, T.A., Hager, W.W.: A sparse proximal implementation of the LP dual active set algorithm. Math. Program (in press) (2006). DOI 10.1007/s10107-006-0017-0
11. Davis, T.A., Hager, W.W.: Row modifications of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **26**, 621–639 (2005)
12. Dongarra, J., Du Croz, J., Hammarling, S., Duff, I.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**, 1–17 (1990)
13. Ferris, M.C., Horn, J.D.: Partitioning mathematical programs for parallel solution. Math. Program. **80**, 35–62 (1998)
14. Gondzio, J., Sarkissian, R.: Parallel interior-point solver for structured linear programs. Math. Prog. **96**, 561–584 (2003)
15. Goto, K., van de Geijn, R.: On reducing TLB misses in matrix multiplication. TR-2002-55, University of Texas at Austin, Departmentn of Computer Sciences (2002)
16. Hackbusch, W.: Multigrid Methods and Applications. Springer, Berlin Heidelberg New York (1985)
17. Hager, W.W.: The dual active set algorithm. In: Pardalos, P.M. (ed.) Advances in Optimization and Parallel Computing 137–142. North Holland, Amsterdam (1992)
18. Hager, W.W.: The LP dual active set algorithm. In: Leone, R.D., Murli, A., Pardalos, P.M., Toraldo, G. (eds.) High Performance Algorithms and Software in Nonlinear Optimization pp. 243–254. Kluwer, Dordrecht (1998)
19. Hager, W.W.: The dual active set algorithm and its application to linear programming. Comput. Optim. Appl. **21**, 263–275 (2002)
20. Hager, W.W.: The dual active set algorithm and the iterative solution of linear programs. In: Pardalos, P.M., Wolkowicz, H. (eds.) Novel Approaches to Hard Discrete Optimization pp. 95–107, vol. 37. Fields Institute Communications (2003)
21. Hager, W.W., Hearn, D.W.: Application of the dual active set algorithm to quadratic network optimization. Comput. Optim. Appl. **1**, 349–373 (1993)
22. Hager, W.W., Shi, C.-L., Lundin, E.O.: Active set strategies in the LP dual active set algorithm, Tech. Report, University of Florida, http://www.math.ufl.edu/∼hager/LPDASA (1996)
23. Hall, J.A.J., McKinnon, K.I.M.: Hyper-sparsity in the revised simplex method and how to exploit it. Comput. Optim. Appl. **32**, 259–283 (2005)
24. Heath, M.T., Raghavan, P.: A Cartesian parallel nested dissection algorithm. SIAM J. Matrix Anal. Appl. **16**, 235–253 (1995)
25. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proc. Supercomputing '95, ACM (1995)

26. Hendrickson, B., Rothberg, E.: Improving the runtime and quality of nested dissection ordering. SIAM J. Sci. Comput. **20**, 468–489 (1999)
27. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1998)
28. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**, 96–129 (1999)
29. Karypis, G., Kumar, V.: Parallel multilevel k-way partitioning scheme for irregular graphs. SIAM Rev. **41**, 278–300 (1999)
30. Liu, J.W.H.: A generalized envelope method for sparse factorization by rows. ACM Trans. Math. Softw. **17**, 112–129 (1991)
31. Luenberger, D.G.: Linear and Nonlinear Programming. Addison Wesley, Reading (1984)
32. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 623–653 (1948)