Math. Program., Ser. B 100: 95–132 (2004)

Digital Object Identifier (DOI) 10.1007/s10107-003-0487-2

Nick Gould · Philippe L. Toint

Preprocessing for quadratic programming

To Roger Fletcher, friend and optimization wizard, on his 65th birthday

Received: November 1, 2002 / Accepted: April 14, 2003
Published online: January 9, 2004 – © Springer-Verlag 2004

Abstract. Techniques for the preprocessing of (not-necessarily convex) quadratic programs are discussed. Most of the procedures extend known ones from the linear to quadratic cases, but a few new preprocessing techniques are introduced. The implementation aspects are also discussed. Numerical results are finally presented to indicate the potential of the resulting code, both for linear and quadratic problems. The impact of insisting that bounds of the variables in the reduced problem be as tight as possible rather than allowing some slack in these bounds is also shown to be numerically significant.

1. Introduction

We investigate presolving techniques for quadratic programs, that is nonlinear optimization problems of the form

$$QP\left\{\begin{array}{ll} \text{minimize} & q(x) = f + g^T x + \frac{1}{2}x^T H x \\ \text{subject to} & \underline{c} \le Ax \le \overline{c} \\ & \underline{x} \le x \le \overline{x} \end{array}\right. \tag{1.1}$$

where the vector of variables x as well as the vectors g, $\underline{x}$ and $\overline{x}$ belong to $\Re^n$, where H is a symmetric $n \times n$ matrix, A an $m \times n$ matrix, and where the vectors $\underline{c}$ and $\overline{c}$ belong to $\Re^m$. Infinite components are allowed in $\underline{x}$, $\overline{x}$, $\underline{c}$ or $\overline{c}$, and fixed variables and equality constraints may be specified by choosing, for any component i, $\underline{x}_i = \overline{x}_i$ or $\underline{c}_i = \overline{c}_i$, respectively. The nonzero entries of A will be denoted by a_{ij} and those of H by h_{ij}. In many cases of practical interest, H and/or A are large and sparse. The components of vectors in the problem will be denoted by a simple subscript.

Presolving (or preprocessing) techniques have a long history in the domain of linear programming (that is for the case where $H = 0$ in (1.1)), where they are considered as an integral part of high quality software for this kind of problem. The idea of *presolving* is to exploit simple logical relations between constraints and variables to simplify the problem at hand at low cost as much as possible, before passing the resulting transformed problem to an effective solver. Once the transformed problem has been solved, the inverse operation is performed to restore its solution in terms of the original formulation, an operation often called *postsolving* (or postprocessing). The reader will find a clear and

N. Gould: Rutherford Appleton Laboratory, Computational Science and Engineering Department, Chilton, Oxfordshire England e-mail: `gould@rl.ac.uk`

Ph.L. Toint: Department of Mathematics, University of Namur, 61, rue de Bruxelles, B-5000, Namur, Belgium e-mail: `philippe.toint@fundp.ac.be`

Mathamatics Subject Classification(2000): 20E28, 20G40, 20C20

recent description of presolving techniques for linear programming in Gondzio (1997) and Andersen and Andersen (1995), while older references include Brearley, Mitra and Williams (1975), Bradley, Brown and Graves (1983) and Tomlin and Welch (1983a, 1983b). Ioslovich (2001) presents interesting developments for linear programs with box constraints and non-negative coefficients. General primal presolve techniques are also described in Fourer and Fourer and Gay (1994). Presolving for the linear complementarity problem is discussed in Ferris and Munson (2001) .

In this paper, we are principally interested in the case where H is nonzero. We do not make any convexity assumption on (1.1), which is to say that H is allowed to be indefinite.

The first order optimality conditions for (1.1) at x are given by the conditions

$$g + Hx - A^T y - z = 0, \tag{1.2}$$

$$\underline{c} \le Ax \le \overline{c}, \tag{1.3}$$

$$\underline{x} \le x \le \overline{x}, \tag{1.4}$$

$$y_i \begin{cases} \ge 0 & \text{when } \underline{c}_i = [Ax]_i, \\ = 0 & \text{when } \underline{c}_i < [Ax]_i < \overline{c}_i, \\ \le 0 & \text{when } \qquad [Ax]_i = \overline{c}_i, \end{cases} \tag{1.5}$$

and

$$z_j \begin{cases} \ge 0 & \text{when } \underline{x}_j = x_j, \\ = 0 & \text{when } \underline{x}_j < x_j < \overline{x}_j, \\ \le 0 & \text{when } \qquad x_j = \overline{x}_j, \end{cases} \tag{1.6}$$

where y and z are the optimal Lagrange multipliers associated with the general linear constraints (1.3) and the bound constraints (1.4), respectively, and where the notation $[u]_k$ denotes the k-th component of the vector u. As we shall see, we may be able to augment problem (1.1) with constraints on y and z of the form

$$\underline{y} \le y \le \overline{y} \tag{1.7}$$

and

$$\underline{z} \le z \le \overline{z}, \tag{1.8}$$

where $\underline{y}, \overline{y} \in \Re^m$ and $\underline{z}, \overline{z} \in \Re^n$ may contain infinite components and where equalities may be specified by choosing lower and upper bounds equal, as above. Combining such bounds with (1.5) and (1.6) may enable us to deduce which of the constraints are active at optimality. In the absence of a priori information, we assume that

$$\underline{y}_i = -\infty, \quad \overline{y}_i = +\infty, \quad \underline{z}_j = -\infty, \quad \overline{z}_j = +\infty,$$

for $i = 1, \dots, m$ and $j = 1, \dots, n$. When a constraint bound is satisfied as an equality, we say that this bound is *active*.

2. Simple transformations

Most problem transformations that we discuss in the following are based on conditions (1.2)–(1.8). We have deliberately restricted the class of transformations that we consider to those that do not create fill-in, i.e. to tranformations that do not cause the number of nonzero entries in H or A to increase, since then the original data structure may be reused to hold that of the transformed problem. They include a few simple operations that we now outline.

- *Fixing a variable.* If we fix the j-th variable to the value x_j, say, this operation requires that we also adapt the other problem quantities to reflect the transformation. In particular, the need to update the constraint bounds, gradient and objective function independent term as follows. If variable j occurs in the i-th constraint, then

$$\underline{c}_i^+ = \underline{c}_i - a_{ij}x_j \qquad \text{and} \qquad \overline{c}_i^+ = \overline{c}_i - a_{ij}x_j,$$

  where the superscript $^+$ denotes quantities associated with the transformed problem. The independent term of the quadratic objective function is updated to

$$f^+ = f + g_j x_j + \tfrac{1}{2} h_{jj} x_j^2$$

  and the gradient becomes

$$g_k^+ = g_k + h_{jk} x_j \quad (k \neq j).$$

  In some cases, a variable may be *ignored*. This happens if variable j does not occur explicitly in the constraints nor in the objective function. Its value has therefore no impact of their values, and it may be set to any value x_j satisfying $\underline{x}_j \leq x_j \leq \overline{x}_j$ without performing the updating of constraint bounds, objective or gradient values.
- *Tightening bounds.* If we are able to deduce that $x_j \leq b_j$, say, then we may revise the upper bound on x_j. We may then distinguish the following subcases:
  - ◇ $b_j < \underline{x}_j$: the new and old bounds are incompatible, and the problem must therefore be infeasible;
  - ◇ $b_j = \underline{x}_j$: the new bound is forcing and we may fix the j-th variable to b_j, using the technique that we have just described;
  - ◇ $\underline{x}_j < b_j < \overline{x}_j$: the new bound can be tightened, which is to say that $\overline{x}_j^+ = b_j$;
  - ◇ $b_j \geq \overline{x}_j$: the new bound is redundant and no information is gained.

  In the case where the bound may be tightened, we note that the original value of $\overline{x}_j$ has become irrelevant, and we may just assume that it is equal to $+\infty$: variable j is said to be *implied upper bounded*, meaning that the bound that can be deduced from the rest of the problem data is stronger than $\overline{x}_j$.

  Similar cases can be distinguished for the case where a new bound of the form $b_j \leq x_j$ is deduced. A variable that is both implied lower bounded and implied upper bounded is known as *implied free*.

  Obviously, the same type of reasoning applies for the possible tightening of the bounds on the dual variables z_j, on the constraint values $[Ax]_i$ and on the multipliers y_i. In our approach, we do not take into account the fact that dual variables or multipliers may become implied free, but we immediately exploit any strictly positive or strictly negative value of z_j or y_i to fix x_j or c_i to one of its bounds.

- *Eliminating a constraint.* If a constraint is either implied free or if it is known to be always satisfied for other reasons, we simply remove it from the problem formulation without altering its feasible domain. The number of rows in A then decreases by one, which is obtained by declaring the eliminated row *inactive*. Since we need these inactive rows in order to reconstruct the data of the problem after the transformed problem has been solved, they are not completely discarded but are kept in the problem data structure at the end of A, without playing any role in the transformed problem.
- *Adding a multiple of a constraint to another.* As we will see below, it may sometimes be advantageous to add a multiple of an equality constraint to another constraint, typically to create a new zero in the updated version of the latter, but sometimes even to zero it entirely, as might happen when combining (and thus identifying) dependent constraints.

3. Problem reduction

We now review the problem simplications techniques, in which the above transformations may be used. A large number of the techniques used for the simplification of QPs apply as well to the case of linear programming, and are essentially described in the references cited above. We also mention them for completeness, and indicate the new transformations explicitly.

3.1. Bound compatibility

Given problem (1.1), we may immediately verify the simple compatibility conditions

$$\underline{x}_j \le \overline{x}_j \quad (j = 1, \dots, n) \text{ and } \underline{c}_i \le \overline{c}_i \quad (i = 1, \dots, m). \tag{3.1}$$

If one of these conditions fails, (1.1) is clearly incompatible and attempting to solve it is doomed to fail. The diagnostic is then passed to the user and preprocessing stopped. We also immediately obtain the simple conclusions:

$$\overline{z}_j < 0 \Rightarrow z_j < 0 \Rightarrow x_j = \overline{x}_j \Rightarrow \overline{x}_j < +\infty$$

and

$$\underline{z}_j > 0 \Rightarrow z_j > 0 \Rightarrow x_j = \underline{x}_j \Rightarrow \underline{x}_j > -\infty$$

and, conversely,

$$\overline{x}_j = +\infty \Rightarrow z_j \ge 0 \Rightarrow \underline{z}_j \ge 0 \quad \text{and} \quad \underline{x}_j = -\infty \Rightarrow z_j \le 0, \Rightarrow \overline{z}_j \le 0,$$

for $j = 1, \dots, n$. If we now turn to the constraints and their multiplers, we obtain that, for $i = 1, \dots, m$,

$$\overline{y}_i < 0 \Rightarrow y_i < 0 \Rightarrow c_i = \overline{c}_i \Rightarrow \overline{c}_i < +\infty$$

and

$$\underline{y}_i > 0 \Rightarrow y_i > 0 \Rightarrow c_i = \underline{c}_i \Rightarrow \underline{c}_i > -\infty$$

and, conversely, that

$$\overline{c}_i = +\infty \Rightarrow y_i \geq 0 \Rightarrow \underline{y}_i \geq 0 \quad \text{and} \quad \underline{c}_i = -\infty \Rightarrow y_i \leq 0 \Rightarrow \overline{y}_i \leq 0.$$

3.2. Reductions of the constraints

We next consider the problem simplifications that can be deduced from the primal constraints (1.3) or (1.4) independently of each other and irrespective of the values of g or H.

3.2.1. Empty rows The first (and easy) case is when row i of the matrix A is identically zero. Then, either

$$\underline{c}_i \leq 0 \leq \overline{c}_i, \tag{3.2}$$

in which case the problem is compatible and the i-th linear constraint may be removed, or (3.2) fails and the problem is again incompatible.

3.2.2. Singleton rows If a general constraint i has the form $\underline{c}_i \leq a_{ij}x_j \leq \overline{c}_i$ for some j between 1 and n, then it can be recast as a set of simple bounds on the variable x_j. More precisely, this constraint is equivalent to

$$\frac{\underline{c}_i}{a_{ij}} \leq x_j \leq \frac{\overline{c}_i}{a_{ij}} \quad (\text{if} \quad a_{ij} > 0) \quad \text{or} \quad \frac{\overline{c}_i}{a_{ij}} \leq x_j \leq \frac{\underline{c}_i}{a_{ij}} \quad (\text{if} \quad a_{ij} < 0).$$

Once the current bounds on x_j have been possibly tightened using these relations, the i-th general constraint may be removed from the problem.

3.2.3. Doubleton rows and split equalities Another useful case is when the i constraint takes the form $a_{ik}x_k + a_{ij}x_j = c_i$ (meaning that $\underline{c}_i = \overline{c}_i = c_i$). In this case, x_j may be eliminated from the equality, giving that

$$\underline{x}_j \leq x_j = \frac{c_i - a_{ik}x_k}{a_{ij}} \leq \overline{x}_j,$$

which in turns is equivalent to the bounds

$$\frac{c_i - a_{ij}\overline{x}_j}{a_{ik}} \leq x_k \leq \frac{c_i - a_{ij}\underline{x}_j}{a_{ik}} \tag{3.3}$$

if a_{ij} and a_{ik} have the same sign, or

$$\frac{c_i - a_{ij}\underline{x}_j}{a_{ik}} \leq x_k \leq \frac{c_i - a_{ij}\overline{x}_j}{a_{ik}} \tag{3.4}$$

otherwise. This shows that the bounds on x_j may be "transferred" to x_k (or vice-versa), so that x_j may be considered as a free variable. In other words, we obtain that $\underline{x}_j^+ = -\infty$, $\overline{x}_j^+ = +\infty$, and

$$\underline{x}_k^+ = \max\left[\underline{x}_k, \frac{c_i - a_{ij}\overline{x}_j}{a_{ik}}\right] \quad \text{and} \quad \overline{x}_k^+ = \min\left[\overline{x}_k, \frac{c_i - a_{ij}\underline{x}_j}{a_{ik}}\right]$$

if a_{ij} and a_{ik} have the same sign, or

$$\underline{x}_k^+ = \max\left[\underline{x}_k, \frac{c_i - a_{ij}\underline{x}_j}{a_{ik}}\right] \quad \text{and} \quad \overline{x}_k^+ = \min\left[\overline{x}_k, \frac{c_i - a_{ij}\overline{x}_j}{a_{ik}}\right]$$

otherwise. As will be seen in Sections 3.3.1 and 3.3.2, this transfer is sometimes advantageous.

A new extension of this idea is to free a variable by "splitting" an equality constraint into two inequalities while removing the bounds on the considered variable. More precisely, the constraint

$$a_{ij}x_j + \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k = c_i \quad \text{and} \quad \underline{x}_j \le x_j \le \overline{x}_j$$

is transformed into

$$a_{ij}x_j + \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k = c_i \quad \text{and} \quad c_i - a_{ij}\overline{x}_j \le \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k \le c_i - a_{ij}\underline{x}_j \tag{3.5}$$

if $a_{ij} > 0$, and

$$a_{ij}x_j + \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k = c_i \quad \text{and} \quad c_i - a_{ij}\underline{x}_j \le \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k \le c_i - a_{ij}\overline{x}_j \tag{3.6}$$

if $a_{ij} < 0$. Of course, this transformation is only useful as a problem reduction technique if the original equality may subsequently be eliminated from the problem, using the fact that x_j is freed in the process (see Sections 3.3.1 and 3.3.2). It also has the drawback of reducing the number of equality constraints, therefore decreasing their potential use in other transformations such as those of Section 3.2.6. For this reason, we found best not to use this technique in the first stages of the presolving process[1], where the transformations of Section 3.2.6 are attempted first.

[1] I.e. the first two of the presolving loops described in Section 7.

3.2.4. Forcing, redundant and infeasible primal constraints If we now consider a general linear constraint, corresponding to the i-th row of A, say, then a simple calculation shows that

$$[Ax]_i = \sum_{j=1}^{n} a_{ij}x_j \le \sum_{\substack{j=1\\ a_{ij}>0}} a_{ij}\overline{x}_j + \sum_{\substack{j=1\\ a_{ij}<0}} a_{ij}\underline{x}_j \overset{\text{def}}{=} u_i, \tag{3.7}$$

and, similarly, that

$$[Ax]_i \ge \sum_{\substack{j=1\\ a_{ij}>0}}^{n} a_{ij}\underline{x}_j + \sum_{\substack{j=1\\ a_{ij}<0}}^{n} a_{ij}\overline{x}_j \overset{\text{def}}{=} \ell_i. \tag{3.8}$$

The quantities ℓ_i and u_i are known as lower and upper *implied bounds* on the i-th constraint. Several cases may now occur. The first is when $u_i < \underline{c}_i$ or $\ell_i > \overline{c}_i$. This implies that the i-th constraint cannot be satisfied for any value of the variables between the variable bounds, and the problem is therefore *infeasible*. The second is when $u_i = \underline{c}_i$ or $\ell_i = \overline{c}_i$. The i-th constraint is then said to be *forcing* in the sense that all variables occuring in this constraint must be fixed to their respective bounds for the constraint to be satisfied, which implies that

$$u_i = \underline{c}_i \quad \Longrightarrow \quad x_j = \overline{x}_j \;\text{ for all }\; j \mid a_{ij} > 0 \;\text{ and }\; x_j = \underline{x}_j \;\text{ for all }\; j \mid a_{ij} < 0$$

or

$$\ell_i = \overline{c}_i \quad \Longrightarrow \quad x_j = \underline{x}_j \;\text{ for all }\; j \mid a_{ij} > 0 \;\text{ and }\; x_j = \overline{x}_j \;\text{ for all }\; j \mid a_{ij} < 0.$$

Once these variables are fixed, the i-th constraint is automatically satisfied and may be eliminated from the problem. It may also happen that $\underline{c}_i < \ell_i$ or $u_i < \overline{c}_i$. In this case, the corresponding inequality constraint is *redundant* and may be ignored.

3.2.5. Further use of implied variable bounds on primal constraints If a primal constraint is not forcing, redundant or inconsistent, we may still use its expression and the values of ℓ_i and u_i to deduce useful bounds on the variables that occur in the constraint. Consider a variable k occuring the i-th constraint with $a_{ik} > 0$. Then

$$\ell_i + a_{ik}(x_k - \underline{x}_k) \le \sum_{j=1}^{n} a_{ij}x_j \le \overline{c}_i \tag{3.9}$$

which then implies that

$$x_k \le \underline{x}_k + \frac{\overline{c}_i - \ell_i}{a_{ik}}.$$

Similarly, if $a_{ik} < 0$, then

$$\ell_i + a_{ik}(x_k - \overline{x}_k) \le \sum_{j=1}^{n} a_{ij}x_j \le \overline{c}_i \tag{3.10}$$

implies that

$$x_k \geq \overline{x}_k + \frac{\overline{c}_i - \ell_i}{a_{ik}}.$$

We may apply the same reasoning using the implied upper bound u_i and obtain that

$$a_{ik} > 0 \quad \Longrightarrow \quad x_k \geq \overline{x}_k + \frac{\underline{c}_i - u_i}{a_{ik}}, \qquad \text{and} \qquad a_{ik} < 0 \quad \Longrightarrow \quad x_k \leq \underline{x}_k + \frac{\underline{c}_i - u_i}{a_{ik}}.$$

Of course, these bounds are only useful if ℓ_i or u_i are finite, which is to say when the bounds of the variables occuring in their definitions (3.7) and (3.8) are finite. We now follow Gondzio (1997) are derive further implications in the special case where only one of the bounds on the variables is infinite in (3.7) or (3.8). Assuming that $\underline{x}_k = -\infty$ for some k such that $a_{ik} > 0$, then, considering the right part (upper bound) of the i-th constraint, we have that

$$a_{ik}x_k + \sum_{\substack{j=1, j\neq k\\ a_{ij}>0}}^{n} a_{ij}\underline{x}_j + \sum_{\substack{j=1\\ a_{ij}<0}}^{n} a_{ij}\overline{x}_j \leq \sum_{j=1}^{n} a_{ij}x_j \leq \overline{c}_i,$$

which then yields that

$$x_k \leq \frac{1}{a_{ik}}\left(\overline{c}_i - \sum_{\substack{j=1, j\neq k\\ a_{ij}>0}}^{n} a_{ij}\underline{x}_j - \sum_{\substack{j=1\\ a_{ij}<0}}^{n} a_{ij}\overline{x}_j\right).$$

Symmetrically, for $a_{ik} < 0$ and $\overline{x}_k = +\infty$, the inequality

$$a_{ik}x_k + \sum_{\substack{j=1\\ a_{ij}>0}}^{n} a_{ij}\underline{x}_j + \sum_{\substack{j=1, j\neq k\\ a_{ij}<0}}^{n} a_{ij}\overline{x}_j \leq \sum_{j=1}^{n} a_{ij}x_j \leq \overline{c}_i,$$

gives that

$$x_k \geq \frac{1}{a_{ik}}\left(\overline{c}_i - \sum_{\substack{j=1\\ a_{ij}>0}}^{n} a_{ij}\underline{x}_j - \sum_{\substack{j=1, j\neq k\\ a_{ij}<0}}^{n} a_{ij}\overline{x}_j\right).$$

If we now turn to the left part (lower bound) of the i-th constraint, we deduce in the same manner that

$$a_{ik} > 0 \;\text{ and }\; \overline{x}_k = +\infty \quad \Longrightarrow \quad x_k \geq \frac{1}{a_{ik}}\left(\underline{c}_i - \sum_{\substack{j=1, j\neq k\\ a_{ij}>0}}^{n} a_{ij}\overline{x}_j - \sum_{\substack{j=1\\ a_{ij}<0}}^{n} a_{ij}\underline{x}_j\right)$$

$$a_{ik} < 0 \;\text{ and }\; \underline{x}_k = -\infty \quad \Longrightarrow \quad x_k \leq \frac{1}{a_{ik}}\left(\underline{c}_i - \sum_{\substack{j=1\\ a_{ij}>0}}^{n} a_{ij}\overline{x}_j - \sum_{\substack{j=1, j\neq k\\ a_{ij}<0}}^{n} a_{ij}\underline{x}_j\right).$$

This technique is especially useful because it allows the derivation of bounds on a variable that is free in the original problem, provided it occurs as the only free variable in a constraint.

3.2.6. Making A sparser A final possible reduction using primal constraints is to create new zero entries in A by suitably combining two constraints. More specifically, we consider the case where both constraints i and k involve variable j ($a_{ij} \neq 0 \neq a_{kj}$) and where the i-th constraint is an equality constraint ($\underline{c}_i = \overline{c}_i$). It is then possible to perform the transformation

$$[\text{constraint } k]^+ = [\text{constraint } k] - \frac{a_{kj}}{a_{ij}} [\text{constraint } i], \tag{3.11}$$

together with

$$\underline{c}_k^+ = \underline{c}_k - \frac{a_{kj}\underline{c}_i}{a_{ij}}, \qquad \overline{c}_k^+ = \overline{c}_k - \frac{a_{kj}\overline{c}_i}{a_{ij}}.$$

The multiplier bounds then become $\underline{y}_k^+ = \underline{y}_k$ and $\overline{y}_k^+ = \overline{y}_k$, together with

$$\underline{y}_i^+ = \underline{y}_i + \frac{a_{kj}\underline{y}_k}{a_{ij}} \qquad \overline{y}_i^+ = \overline{y}_k + \frac{a_{kj}\overline{y}_k}{a_{ij}}$$

if the sign of a_{kj} is the same as that of a_{ij}, or, otherwise,

$$\underline{y}_i^+ = \underline{y}_i + \frac{a_{kj}\overline{y}_k}{a_{ij}} \qquad \overline{y}_i^+ = \overline{y}_k + \frac{a_{kj}\underline{y}_k}{a_{ij}}.$$

Constraint i is then called the *pivot constraint* and a_{ij} the *pivot*. This transformation does not alter the feasible set of the problem, but introduces a zero in position (k, j) of A. If we wish to remain consistent with our strategy of avoiding fill-in in this matrix, we must restrict such transformations to the case where the sparsity pattern of row k contains that of row i, that is when

$$\{j \mid 1 \leq j \leq n \ \text{ and } \ a_{ij} \neq 0\} \subseteq \{j \mid 1 \leq j \leq n \ \text{ and } \ a_{kj} \neq 0\}.$$

In practice, we maintain a list of the equality constraints which are potential candidates for being pivot constraints. Within this list, we follow the procedure suggested in Gondzio (1997): for each constraint i taken by increasing number of nonzeros, we first search the column of A that contain the least number of nonzeros. We then consider combining constraint i and each of the constraints that have a nonzero in this column of minimum size: the transformation is applied if the sparsity pattern of the considered constraint k is a superset of that of the pivot constraint i. Note that constraint k may itself be an equality, which implies that the list of equality constraints (used as candidate for pivot constraints) must be managed dynamically.

If constraint k turns out to be a multiple of constraint i, then the transformation not only "zeroes" a_{kj}, but the entire k-th row through cancellation. Monitoring cancellation is useful here, as it may lead to the conclusion that constraint k may be eliminated from the problem.

The attentive reader will have noticed that this transformation is a form Gaussian elimination. Since the purpose of the presolving procedure is not to replace the linear or quadratic solvers that are potentially much more efficient in terms of linear algebra, we limit the number of "passes" in the procedure, that is the number of times one goes through the list of candidate equality constraints to see if they can be used as pivot constraints. Also note that care must be taken to avoid dividing by a pivot value which is too small, which could cause severe loss of information, and thus some kind of threshold pivoting (see Duff, Erisman and Reid, 1986, for instance) is advisable.

3.3. Reductions on the variables

We next consider the problem reductions that depend on all constraints, or on the values of g or H.

3.3.1. Free linear singleton columns Free (or implied free) variables that occur only linearly in the objective function and explicitly occuring in a single constraint may be handled using equation (1.2), which reduces in this case to $g_j - a_{ij} y_i = 0$. Thus, we must have that

$$y_i = \frac{g_j}{a_{ij}}. \tag{3.12}$$

Two cases are then possible. The first (and most common) is when $g_j \neq 0$. In this situation, y_i is also nonzero and its sign determines which of the lower or upper bound is active for constraint i. This constraint may therefore be interpreted as an equality constraint, setting

$$\underline{c}_i^+ = \overline{c}_i^+ = \overline{c}_i \ \text{ if } \ y_i < 0, \qquad \text{or} \qquad \overline{c}_i^+ = \underline{c}_i^+ = \underline{c}_i \ \text{ if } \ y_i > 0.$$

Furthermore, since variable j only appears in the i-th constraint which is now transformed into an equality, it can be substituted out from the constraint, giving that

$$x_j = \frac{1}{a_{ij}} \left(\overline{c}_i^+ - \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} a_{i\ell} x_\ell \right). \tag{3.13}$$

When a_{ij} is not too small[2], we then remove this variable from the problem, while remembering that its value can be computed from (3.13), once the optimal values of the other variables are known. Of course, the objective function and the gradient of the transformed problem must be updated to reflect this substitution. Using (3.12) and (3.13), we obtain that these updates are given by

$$f^+ = f + \overline{c}_i y_i \qquad \text{and} \qquad g_\ell^+ = g_\ell - a_{i\ell} y_i \quad (\ell \neq j, a_{i\ell} \neq 0). \tag{3.14}$$

The second case is when $g_j = y_i = 0$. In this case, the value of the objective function is independent from that of the j-th variable. It can therefore be fixed to any value that

[2] In practice, when its abolute value is above some user-specified threshold.

makes the problem feasible, without affecting optimality. We may thus, for instance, choose (3.13) or

$$x_j = \frac{1}{a_{ij}} \left(\frac{\underline{c}_i + \overline{c}_i}{2} - \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} a_{i\ell} x_\ell \right), \tag{3.15}$$

depending on which of a boundary or interior solution is prefered. Note that, since $y_i = 0$, no update of the objective function or gradient is necessary. In both cases, constraint i may then be eliminated from the problem, as it is implicitly taken into account by (3.13) or (3.15).

This reduction is quite advantageous since it allows both the numbers of variables and constraints to decrease by one. Note that the technique of Section 3.2.3 (transferring bounds on the variables in a doubleton row) may sometimes be used to make a linear singleton variable free or implied free.

Finally, linear singleton columns that are not free can nevertheless be used to deduce bounds on the associated multiplier if one of the bounds corresponding to the column is infinite. Indeed, in this situation, we have that $g_j - a_{ij} y_i = z_j$. Thus, we obtain that

$$y_i^+ \leq \frac{g_j}{a_{ij}} \quad \text{if } \overline{x}_j = +\infty \text{ and } a_{ij} > 0, \text{ or if } \underline{x}_j = -\infty \text{ and } a_{ij} < 0,$$

and

$$y_i^+ \geq \frac{g_j}{a_{ij}} \quad \text{if } \overline{x}_j = +\infty \text{ and } a_{ij} < 0, \text{ or if } \underline{x}_j = -\infty \text{ and } a_{ij} > 0.$$

3.3.2. Doubleton columns We now introduce a new transformation that applies to the case where column j of A only contains two nonzeros, a_{ij} and a_{kj}, variable j is linear (meaning that $h_{ij} = 0$ for $i = 1, \ldots, n$) and free or implied free, and constraint i is an equality constraint. Then, as above, variable j may substituted from this constraint, yielding (3.13). However, the situation is now slightly more complicated in that not only the objective function and gradient must be updated using (3.14), but (3.13) must also be substituted into constraint k. This transformation may be expressed, provided a_{ij} is not too small, as

$$a_{k\ell}^+ = a_{k\ell} - \frac{a_{kj} a_{i\ell}}{a_{ij}} \tag{3.16}$$

for $\ell = 1, \ldots, n$, $\ell \neq j$, and

$$\underline{c}_k^+ = \underline{c}_k - \frac{a_{kj} \underline{c}_i}{a_{ij}} \quad \text{and} \quad \overline{c}_k^+ = \overline{c}_k - \frac{a_{kj} \overline{c}_i}{a_{ij}}$$

(note that $\underline{c}_i = \overline{c}_i$). Moreover, the j-th component of (1.2) gives that

$$g_j - a_{ij} y_i - a_{kj} y_k = z_j = 0 \tag{3.17}$$

since variable j must be linear and (implied) free. We therefore deduce the bounds

$$\frac{g_j - a_{kj}\overline{y}_k}{a_{ij}} \leq y_i \leq \frac{g_j - a_{kj}\underline{y}_k}{a_{ij}}$$

if a_{ij} and a_{kj} have the same sign, and

$$\frac{g_j - a_{kj}\underline{y}_k}{a_{ij}} \leq y_i \leq \frac{g_j - a_{kj}\overline{y}_k}{a_{ij}}$$

otherwise. Similarly, if a_{kj} is not too small,

$$\frac{g_j - a_{ij}\overline{y}_i}{a_{kj}} \leq y_k \leq \frac{g_j - a_{ij}\underline{y}_i}{a_{kj}}$$

if a_{ij} and a_{kj} have the same sign, and

$$\frac{g_j - a_{ij}\underline{y}_i}{a_{kj}} \leq y_k \leq \frac{g_j - a_{ij}\overline{y}_i}{a_{kj}}$$

otherwise. One could argue that (3.16) might create fill-in in A, when $a_{k\ell} = 0 \neq a_{i\ell}$ for some $\ell \neq j$. This is formally correct, but, fortunately, of no consequence since we may use the storage required for the i-th pivot contraint (to be eliminated) to store these possible fill-ins, as their number cannot exceed the number of nonzeros in constraint i. Of course this requires some care in the associated data structure management (see Section 4). In our implementation, we use an additional integer array s of size m to allow the concatenation of two rows of A into a single "merged" row. More precisely, $s(k)$ contains the index of the row to be concatenated to row k, if there is such a row, or a conventional marker if there is no further row to concatenate to row k. This allows an easy exploration of merged rows.

This kind of reduction is especially useful in A (or part of it) is associated with linear network constraints, where intermediate nodes may often be eliminated. On the other hand, there is a danger of creating constraints that are too dense (although the total number of nonzero entries in A can only decrease), which can be a disadvantage to certain interior-point methods, and safeguards against this effect may be built in the procedure. For instance, row merging can be allowed so long as the size of the merged row does not exceed that of row k in the original problem formulation by more than a user-specified percentage when A is sparse. Observe that one may again attempt to make a linear doubleton variable free or implied free by applying the technique of Section 3.2.3. Finally note that our decision to use the storage originally allocated for the pivot constraint for the possible fill-in in row k prevents us from substituting (3.13) into more than one other constraint.

3.3.3. Weakly forcing, forcing and infeasible dual constraints In the case of general dual constraints, we see that the j-th constraint in (1.2) may be rewritten as

$$\sum_{i=1}^{m} a_{ij} y_i - \sum_{\ell=1}^{n} h_{\ell j} x_\ell = g_j - z_j. \tag{3.18}$$

If some components of $\underline{y}$ or $\overline{y}$ are finite, this equation may be exploited using the notion of implied bounds, in a manner similar to used for primal constraints in Section 3.2.4. We first define

$$v_j \stackrel{\text{def}}{=} \sum_{\substack{i=1\\a_{ij}>0}}^{m} a_{ij}\underline{y}_i + \sum_{\substack{i=1\\a_{ij}<0}}^{m} a_{ij}\overline{y}_i - \sum_{\substack{\ell=1\\h_{\ell j}>0}}^{n} h_{\ell j}\overline{x}_\ell - \sum_{\substack{\ell=1\\h_{\ell j}<0}}^{n} h_{\ell j}\underline{x}_\ell \tag{3.19}$$

and

$$w_j \stackrel{\text{def}}{=} \sum_{\substack{i=1\\a_{ij}>0}}^{m} a_{ij}\overline{y}_i + \sum_{\substack{i=1\\a_{ij}<0}}^{m} a_{ij}\underline{y}_i - \sum_{\substack{\ell=1\\h_{\ell j}>0}}^{n} h_{\ell j}\underline{x}_\ell - \sum_{\substack{\ell=1\\h_{\ell j}<0}}^{n} h_{\ell j}\overline{x}_\ell. \tag{3.20}$$

From these definitions, it is clear that

$$v_j \le \sum_{i=1}^{m} a_{ij}y_i - \sum_{\ell=1}^{n} h_{\ell j}x_\ell \le w_j,$$

and therefore, from (3.18), that

$$v_j \le g_j - z_j \le w_j. \tag{3.21}$$

Unsurprisingly, v_j and w_j are called *implied bounds* on the j-th dual constraint. We then obtain that

$$g_j - w_j \le z_j \le g_j - v_j.$$

Let us consider first the case where $\overline{z}_j \le 0$, which must occur if $\underline{x}_j = -\infty$, for instance. A first possibility is then that $g_j > w_j$. If this happens, then we deduce from (3.21) that

$$z_j \ge g_j - w_j > 0, \tag{3.22}$$

which is inconsistent with our assumption that $\overline{z}_j \le 0$. Hence the problem is dual infeasible, and the presolving procedure may be terminated. A second possibility is that $g_j < v_j$. In this case, we obtain from (3.21) that

$$z_j \le g_j - v_j < 0, \tag{3.23}$$

which in turn implies that the upper bound on the jth variable must be active: we may then fix x_j to $\overline{x}_j$. The j-th dual constraint is said to be *weakly forcing* and the j-th variable is said to be *dominated.*

On the other hand, it may happen that $\underline{z}_j \ge 0$ (for instance if $\overline{x}_j = +\infty$). Then, if $g_j < v_j$, we deduce from (3.21) that (3.23) holds, which is again incompatible with our assumption that $\underline{z}_j \ge 0$. The problem is also dual infeasible in this case, and our preprocessor exits with this information. If $g_j > w_j$, then it follows as above that (3.22) holds. The j-th dual constraint is therefore weakly forcing and we may then fix x_j to its lower bound $\underline{x}_j$.

Finally, it may happen that the j-th dual variable is known, i.e. that $\underline{z}_j = \overline{z}_j$. For instance, if variable j is free or implied free, then $\underline{z}_j = 0 = \overline{z}_j$. If, additionally, we also have that $v_j = g_j - \underline{z}_j$ or $w_j = g_j - \overline{z}_j$, then the j-th dual constraint is *forcing*, and, as in Section 3.2.4, all variables and multipliers occuring in this constraint must be fixed to their appropriate bound for the constraint to be satisfied.

3.3.4. Further use of the implied bounds on dual constraints If (3.18) is neither infeasible or (weakly) forcing, we may, as in Section 3.2.5, make further use of v_j and w_j, yielding new bounds on the variables or multipliers that may then be used for possibly tightening the best bounds known for these quantities.

Consider first the case where $\overline{z}_j \leq 0$. We may then derive from the definition of w_j and (3.21) that, for $k = 1, \dots, m$,

$$w_j + a_{kj}(y_k - \overline{y}_k) \geq g_j - z_j \geq g_j$$

when $a_{kj} > 0$, and

$$w_j + a_{kj}(y_k - \underline{y}_k) \geq g_j - z_j \geq g_j$$

when $a_{kj} < 0$. From these two inequalities, we may then deduce that

$$y_k \geq \overline{y}_k - \frac{w_j - g_j}{a_{kj}} \text{ if } a_{kj} > 0, \qquad \text{and} \qquad y_k \leq \underline{y}_k - \frac{w_j - g_j}{a_{kj}} \text{ if } a_{kj} < 0.$$

Similarly, we also obtain from the definition of w_j and (3.21) that, for $k = 1, \dots, n$,

$$w_j - h_{kj}(x_k - \underline{x}_k) \geq g_j - z_j \geq g_j$$

when $h_{kj} > 0$, and

$$w_j - h_{kj}(x_k - \overline{x}_k) \geq g_j - z_j \geq g_j$$

when $h_{kj} < 0$, from which we derive that

$$x_k \leq \underline{x}_k + \frac{w_j - g_j}{h_{kj}} \text{ if } h_{kj} > 0, \qquad \text{and} \qquad x_k \geq \overline{x}_k + \frac{w_j - g_j}{a_{kj}} \text{ if } h_{kj} < 0.$$

The same reasoning applied to the case where $\underline{z}_j \geq 0$ (and thus, in particular, if $\overline{x}_j = +\infty$) yields that

$$y_k \leq \underline{y}_k - \frac{v_j - g_j}{a_{kj}} \text{ if } a_{kj} > 0, \qquad \text{and} \qquad y_k \geq \overline{y}_k - \frac{v_j - g_j}{a_{kj}} \text{ if } a_{kj} < 0.$$

and

$$x_k \geq \overline{x}_k + \frac{v_j - g_j}{h_{kj}} \text{ if } h_{kj} > 0, \qquad \text{and} \qquad x_k \leq \underline{x}_k + \frac{v_j - g_j}{a_{kj}} \text{ if } a_{kj} < 0.$$

This last set of bounds involving h_{kj} is new and specific to quadratic programs.

As for primal constraints, the above bounds are potentially useful when the involved implied bounds are finite, that is when $v_j > -\infty$ or $w_j < +\infty$. However, we may also apply the technique of Section 3.2.5 and deduce further bounds on the variables or multipliers in the case where the implied bounds on the j-th dual constraint contain a single infinite contribution (that is a single infinite term in the sums of (3.19) or (3.20)). We now detail these bounds for the sake of completeness.

The unique infinite contribution may occur because one bound on the multipliers is infinite, or because one bound on the variables is infinite. Assume first that it corresponds to the k-th multiplier (that is $\underline{y}_k = -\infty$ or $\overline{y}_k = +\infty$), and define

$$v_j^{[y_k]} \overset{\text{def}}{=} \sum_{\substack{i=1, i\neq k\\ a_{ij}>0}}^{m} a_{ij}\underline{y}_i + \sum_{\substack{i=1, i\neq k\\ a_{ij}<0}}^{m} a_{ij}\overline{y}_i - \sum_{\substack{\ell=1\\ h_{\ell j}>0}}^{n} h_{\ell j}\overline{x}_\ell - \sum_{\substack{\ell=1\\ h_{\ell j}<0}}^{n} h_{\ell j}\underline{x}_\ell,$$

and

$$w_j^{[y_k]} \overset{\text{def}}{=} \sum_{\substack{i=1, i\neq k\\ a_{ij}>0}}^{m} a_{ij}\overline{y}_i + \sum_{\substack{i=1, i\neq k\\ a_{ij}<0}}^{m} a_{ij}\underline{y}_i - \sum_{\substack{\ell=1\\ h_{\ell j}>0}}^{n} h_{\ell j}\underline{x}_\ell - \sum_{\substack{\ell=1\\ h_{\ell j}<0}}^{n} h_{\ell j}\overline{x}_\ell.$$

We thus have that

$$a_{kj}y_k + v_j^{[y_k]} \leq \sum_{i=1}^{m} a_{ij}y_i - \sum_{\ell=1}^{n} h_{\ell j}x_\ell \leq a_{kj}y_k + w_j^{[y_k]}$$

and therefore that

$$a_{kj}y_k + v_j^{[y_k]} \leq g_j - z_j \leq a_{kj}y_k + w_j^{[y_k]}.$$

If we also have that $\overline{z}_j \leq 0$, we then deduce that

$$y_k \geq \frac{g_j - w_j^{[y_k]}}{a_{kj}} \qquad \text{if } a_{kj} > 0 \text{ and } \overline{y}_k = +\infty,$$

and that

$$y_k \leq \frac{g_j - w_j^{[y_k]}}{a_{kj}} \qquad \text{if } a_{kj} < 0 \text{ and } \underline{y}_k = -\infty.$$

If, on the other hand, $\underline{z}_j \geq 0$, then we obtain that

$$y_k \leq \frac{g_j - v_j^{[y_k]}}{a_{kj}} \qquad \text{if } a_{kj} > 0 \text{ and } \underline{y}_k = -\infty,$$

and

$$y_k \geq \frac{g_j - v_j^{[y_k]}}{a_{kj}} \qquad \text{if } a_{kj} < 0 \text{ and } \overline{y} = +\infty.$$

Symmetrically, if the only infinite bound corresponds to the k-th variable (that is $\underline{x}_k = -\infty$ or $\overline{x}_k = +\infty$), we may define

$$v_j^{[x_k]} \overset{\text{def}}{=} \sum_{\substack{i=1\\ a_{ij}>0}}^{m} a_{ij}\underline{y}_i + \sum_{\substack{i=1\\ a_{ij}<0}}^{m} a_{ij}\overline{y}_i - \sum_{\substack{\ell=1, \ell\neq k\\ h_{\ell j}>0}}^{n} h_{\ell j}\overline{x}_\ell - \sum_{\substack{\ell=1, \ell\neq k\\ h_{\ell j}<0}}^{n} h_{\ell j}\underline{x}_\ell,$$

and

$$w_j^{[x_k]} \overset{\text{def}}{=} \sum_{\substack{i=1\\a_{ij}>0}}^{m} a_{ij}\overline{y}_i + \sum_{\substack{i=1\\a_{ij}<0}}^{m} a_{ij}\underline{y}_i - \sum_{\substack{\ell=1,\ell\neq k\\h_{\ell j}>0}}^{n} h_{\ell j}\underline{x}_\ell - \sum_{\substack{\ell=1,\ell\neq k\\h_{\ell j}<0}}^{n} h_{\ell j}\overline{x}_\ell.$$

If, as above, we also have that $\overline{z}_j \leq 0$, we may then deduce that

$$x_k \leq -\frac{g_j - w_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} > 0 \text{ and } \underline{x}_k = -\infty, \tag{3.24}$$

and that

$$x_k \geq -\frac{g_j - w_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} < 0 \text{ and } \overline{x}_k = +\infty, \tag{3.25}$$

while, if $\underline{z}_j \geq 0$, we obtain that

$$x_k \geq -\frac{g_j - v_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} > 0 \text{ and } \overline{x}_k = +\infty, \tag{3.26}$$

and

$$x_k \leq -\frac{g_j - v_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} < 0 \text{ and } \underline{x}_k = -\infty. \tag{3.27}$$

3.3.5. Removing dependent variables The removal of dependent variables is reminiscent of the procedure of Section 3.2.6, applied to columns and now possibly involving the matrix H. The difference is that we restrict ourselves here to an attempt to identify all pairs of variables k and j such that

$$\alpha h_{\ell k} = h_{\ell j} \quad (\ell = 1, \ldots, n) \qquad \text{and} \qquad \alpha a_{ik} = a_{ij} \quad (i = 1, \ldots, m) \tag{3.28}$$

for some constant $\alpha \neq 0$ independent of i and ℓ. In order to verify this condition, we start by maintaining a list of variables that are potentially dependent on another and by rejecting variables k from this list whose associated column is diagonal in H. Indeed, if $h_{kk} \neq 0$, then (3.28) implies that $h_{kj} = h_{jk} \neq 0$, in which case column k of H cannot be diagonal. We next identify a row i of A such that $a_{ik} \neq 0$ which is of minimal size, and attempt to verify that (3.28) holds for each column $j \neq k$ such that $a_{ij} \neq 0$. Since this verification is potentially expensive, we first reject all columns j of A that have a number of nonzeros different from that of column k. We also eliminate all j such that the size of the j-th row of H is different from that of its k-th row. We also terminate the verification of (3.28) as soon as a row index is found such that the second part of (3.28) fails, thereby limiting the computational effort as much as possible. If the second part of the condition (concerning A) is verified, we then verify its first part (on H), stopping, as above, as soon as possible. This procedure is new, but inspired by Tomlin and Welch (1983a), Andersen and Andersen (1995) and Gondzio (1997), with adaptations for the presence of the Hessian H. If (3.28) holds, two cases may arise.

The first is when, additionally, $\alpha g_k = g_j$. In this case, we may interpret variable j as being redundant in the problem, since the dependence of the objective function and constraint value on this variable is exactly α times their dependence on variable k. We then remove variable j from the problem, and perform the replacement

$$[\text{variable } k]^+ = [\text{variable } k] + \alpha[\text{variable } j],$$

such that

$$h_{\ell k} x_k^+ = h_{\ell k} x_k + \alpha h_{\ell k} x_j = h_{\ell k} x_k + h_{\ell j} x_j \quad (\ell = 1, \ldots, n),$$

and

$$a_{ik} x_k^+ = a_{ik} x_k + \alpha a_{ik} x_j = a_{ik} x_k + a_{ij} x_j \quad (i = 1, \ldots, m).$$

This implies that the bounds of the new variable k are updated by

$$\underline{x}_k^+ = \underline{x}_k + \alpha \underline{x}_j \quad \text{and} \quad \overline{x}_k^+ = \overline{x}_k + \alpha \overline{x}_j \quad \text{if} \quad \alpha > 0,$$

or

$$\underline{x}_k^+ = \underline{x}_k + \alpha \overline{x}_j \quad \text{and} \quad \overline{x}_k^+ = \overline{x}_k + \alpha \underline{x}_j \quad \text{if} \quad \alpha < 0,$$

while the bounds on the associated dual variable are now given by

$$\underline{z}_k^+ = \underline{z}_k + \alpha \underline{z}_j \quad \text{and} \quad \overline{z}_k^+ = \overline{z}_k + \alpha \overline{z}_j \quad \text{if} \quad \alpha > 0,$$

or

$$\underline{z}_k^+ = \underline{z}_k + \alpha \overline{z}_j \quad \text{and} \quad \overline{z}_k^+ = \overline{z}_k + \alpha \underline{z}_j \quad \text{if} \quad \alpha < 0.$$

No updating of A, H or g is necessary. This type of combination is especially useful when the updated lower or upper bound become infinite.

If, on the other hand, $\alpha g_k \neq g_j$, then it may happen that one of the variables under consideration *dominates* the other. If $\alpha g_k > g_j$, more reduction in the objective function can be achieved by reducing variable k as much as possible than would be possible by reducing variable j. However, the dual constraints have to remain feasible, which is always possible if variable j is free or if it is suitably unbounded. More formally, using (3.28), the symmetry of H and our assumption on g_k, we have that

$$\alpha z_k = \alpha g_k + \alpha \sum_{\ell=1}^{n} h_{k\ell} x_\ell - \alpha \sum_{i=1}^{m} a_{ik} y_i > g_j + \sum_{\ell=1}^{n} h_{j\ell} x_\ell - \sum_{i=1}^{m} a_{ij} y_i = z_j, \quad (3.29)$$

and therefore, if $z_j \geq 0$ (which is the case when $\overline{x}_j = +\infty$) and $\alpha > 0$, we deduce that $z_k > 0$. As a consequence, variable k may be fixed to its lower bound in this case, unless it is equal to $-\infty$, implying that the problem is dual infeasible. The same deduction can of course be made from (3.29) if $z_j \leq 0$ (for instance when $\underline{x}_j = -\infty$) and $\alpha < 0$. Symmetrically, if $\alpha g_k < g_j$, we obtain that

$$\alpha z_k = \alpha g_k + \alpha \sum_{\ell=1}^{n} h_{k\ell} x_\ell - \alpha \sum_{i=1}^{m} a_{ik} y_i < g_j + \sum_{\ell=1}^{n} h_{j\ell} x_\ell - \sum a_{ij} y_i = z_j, \quad (3.30)$$

and we deduce that variable k may be fixed at its upper bound if $z_j \leq 0$ (e.g. if $\underline{x}_j = -\infty$) and $\alpha > 0$, or if $z_j \geq 0$ (e.g. if $\overline{x}_j = +\infty$) and $\alpha < 0$. The inequalities (3.29) and (3.30) may also be used symmetrically to deduce bounds on z_j from the bounds on z_k, and thus possibly to fix x_j at one of its bounds.

3.4. Unconstrained reductions

A last case of interest occurs when a variable does not appear in the linear constraints: we then say that it is *linearly unconstrained*. However, note that such a variable usually remains subject to bound constraints.

If a linearly unconstrained variable j additionally occurs only linearly in the objective function, then it may simply be fixed to its lower or upper bound, depending on whether the corresponding gradient component g_j is positive or negative. If the associated bound is infinite, then the problem is dual infeasible. If g_j is identically zero, then variable j has no impact on the value of the objective function, and it is fixed to an optimal value that is arbitrarily chosen between its lower and upper bounds.

If such a variable does not occur linearly in the objective function, but its contribution is separable from that of other variables (which is the case when the corresponding column of H is diagonal), then the global minimum of the one-dimensional associated quadratic within the variable's bounds can be computed unless this problem is dual infeasible. The considered variable may then be fixed to the computed minimizer and its dual variable chosen accordingly. This last technique is also new and specific to quadratic programs.

4. Data structures and presolving modes

We assume that the matrices A and H for problem (1.1) have been input using a row-wise storage scheme (see Duff, Erisman and Reid, 1986, §2.7), and that only the lower triangular part of the symmetric H is given – our implementation actually supports, in addition, both dense and co-ordinate input, but these are converted internally into a row-wise scheme.

All the problem reductions described above can then be carried out reasonably simply and efficiently, provided we maintain adequate additional information and suitably manage the corresponding data structure. In our implementation, we have chosen to maintain a record of the number of nonzeros in each row and column of A and H, as well as an indicator of the diagonal nature of each column of H. This informations is stored in three integer vectors that are built at the beginning of the problem analysis and maintained throughout the problem reduction process.

Since a number of problem reduction techniques (see Section 3.2) require access to rows of A while others (see Section 3.3) need access to its columns, it is useful to supplement the structure of A by a linked lists superstructure describing its columns (see Duff et al., 1986, §.8). Similarly, it is advantageous to be able to access to a complete row or column of H including its part beyond the diagonal. For this purpose, we also build a pointer superstructure that provides a linked list of all nonzero elements of the upper triangular part of a Hessian row. Both these superstructures (for A and H, whenever appropriate) are computed at the start of the analysis.

As suggested in Section 3.3.2, we also maintain an array of length m indicating, in its i-th component, whether or not another row structure of A is to be concatenated with row i. The presence of this array slightly complicates all program loops on the nonzero entries of a row in this case, but the additional complexity remains marginal.

We also maintain an history of the successive transformations, as it is necessary to invert them once the transformed problem has been solved. This history takes the form of two integer and one real vectors. Each of these arrays has a length equal to the number of transformations actually applied to the problem. The coding of these transformations is slightly more complicated than used by Gondzio (1997), but remains comparable in the amount of memory used. As the number of problem transformations may sometimes be substantial, we also provide a mechanism to save their history to disk files. The user is asked to specify how many transformations are to be kept in memory, and the associated memory storage is then successively filled and written to disk whenever necessary, in a totally transparent manner.

Two status vectors are also maintained for the variables and constraints, in order to distinguish active variables or constraints (that is variables or constraints that haven't been removed from the problem formulation) from inactive ones. The status vector associated with the variables is also used to keep track of implied free variables, whenever possible. Note that the user may specify in these vectors if a variable or a constraint is to be considered as part of the problem or not. This is potentially useful in the case where one solves a set of "neighbouring QPs" that differ by the presence/absence of a few constraints or variables, as can be the case in the exploration of the combinatorial tree for a nonlinear integer programming problem. We further maintain a list of rows and columns of A whose number of nonzeros have been modified, in order to restrict the application of the heuristics of Section 3.2.6 and 3.3 to these rows and columns, respectively. This technique allows us to use these more costly procedures systematically at a very modest cost.

We also followed a suggestion made by Fourer and Gay (1994) in order to avoid, to some extent, the presence of redundant constraints in the reduced problem, which arise naturally as a result of the elimination process. While performing the problem transformations, we store in memory the best bounds on the variables that are known not to be redundant for the (reduced) linear constraints, and allow the user the option to pass this set of bounds to the solver, instead of the tighter but possibly more degenerate ones that result from the problem reduction process. This option is referred to as the "medium" mode, at variance with the "tightest" mode where the tightest known bounds on the variables and constraints are included in the reduced problem. We also extended this idea by allowing the user to require the loosest bounds that are known to guarantee the equivalence of the reduced problem an the original one, an option that we refer to as the "loosest" mode. We comment in Section 7 on the impact of using these options.

5. The final permuted format

When the problem analysis is finally complete, we permute the resulting transformed problem into a form that is designed to make the computation on this transformed problem efficient. We have chosen a form where the variables are ordered so that free variables occur first, and so that the bounds on the remaining variables appear in the order

$$\begin{array}{lllll}
\text{non-negative variables:} & 0 & \leq x_j & & (j = 1, \ldots, n_1),\\
\text{lower bounded variables:} & \underline{x}_j & \leq x_j & & (j = n_1 + 1, \ldots, n_2),\\
\text{range constrained variables:} & \underline{x}_j & \leq x_j & \leq \overline{x}_j & (j = n_2 + 1, \ldots, n_3),\\
\text{upper bounded variables:} & & x_j & \leq \overline{x}_j & (j = n_3 + 1, \ldots, n_4),\\
\text{non-positive variables:} & & x_j & \leq 0 & (j = n_4 + 1, \ldots, n_5),
\end{array}$$

where the unspecified bounds are infinite and where $n_5 \leq n$ is the total number of variables remaining in the transformed problem. Within each category, the variables are further ordered so that those with non-zero diagonal Hessian entries occur before the remainder. The constraints are also ordered so that their bounds appear in the order

$$\begin{array}{lllll}
\text{equality constraints:} & \underline{c}_i & = [Ax]_i & & (i = 1, \ldots, m_1),\\
\text{non-negativity constraints:} & 0 & \leq [Ax]_i & & (i = m_1 + 1, \ldots, m_2),\\
\text{lower bounded constraints:} & \underline{c}_i & \leq [Ax]_i & & (i = m_2 + 1, \ldots, m_3),\\
\text{range constraints:} & \underline{c}_i & \leq [Ax]_i & \leq \overline{c}_i & (i = m_3 + 1, \ldots, m_4),\\
\text{upper bounded constraints:} & & [Ax]_i & \leq \overline{c}_i & (i = m_4 + 1, \ldots, m_5),\\
\text{non-positivity constraints:} & & [Ax]_i & \leq 0 & (i = m_5 + 1, \ldots, m_6),
\end{array}$$

where $m_6 \leq m$ is the total number of constraints remaining in the transformed problem. The data g, H and A is reordered to conform with the above scheme; the diagonal terms of H (if present) are stored last within each of their given (subdiagonal) rows, so that unnecessary checking for diagonal entries does not occur when forming Hessian-vector products.

Of course, the data associated with the disregarded variables or constraints is not lost. In practice, we permute it to the end of the original data structures, beyond the reordered data for the transformed problem. This ensures that it does not interfere at all in the processing of the transformed problem.

6. Problem restoration

Once the transformed problem has been processed (solved, hopefully), it is of course important to translate the results obtained back to the original QP formulation, reintroducing variables and constraints that were eliminated in the presolving process. This is called *postsolving*, and is simply performed by applying the inverse of the problem transformations in the reverse order. In terms of the notation used above, this means that, for each transformation, we have to deduce the quantities without the $^+$ superscript from those with a $^+$ superscript. While this is rather straightforward in most cases, several points are nevertheless worth noticing.

The first is that the restoration of certain problem values might require that of others. For instance, the restoration of the original coefficients of the objective function's gradient requires the restoration of the coefficients of A as well, whenever a variable has been fixed in the presolving process. A simple strategy would be to always restore the complete problem data, but it might be advantageous to avoid the irrelevant part of this computation when restoration of the complete problem is not necessary. We must therefore keep track of the successive dependencies between the various problem quantities that must be restored while transformations are determined and applied, in order to reconstruct all necessary problem components, but only those, and only as far as necessary within the postsolve process. In our implementation, we explicitly store these dynamic dependencies during the problem reduction phase for reuse when postsolving.

The second point of interest is that it is usually desirable to recover the values of the dual variables and multipliers at the solution of the original problem from those at the solution of the transformed problem. For each problem transformation, we therefore have to deduce z and y from z^+, y^+ and, possibly, other data of the transformed problem. We briefly consider how this can be done for some problem transformations.

6.1. Fixing a variable

Fixing a variable x_j does not always permits us to simultaneously fix the associated dual variable z_j. For instance, if we happen to deduce that $z_j > 0$, then we may fix x_j to its lower bound, but the value of z_j is given by the equation $z_j = g_j + [Hx]_j - [A^T y]_j$ and both x and y are still unknown at the presolving stage. Fortunately, this equation may be used when postsolving to deduce z_j from the optimal x^+, H^+, A^+ and y^+, i.e.,

$$z_j = g_j + [H^+ x^+]_j - [A^{+T} y^+]_j.$$

6.2. Tightening a bound on the variables

If a bound on x_k, say, is tightened during the analysis, it may happen that the solution of the reduced problem has a nonzero dual variable z_k^+ associated with this constraint. Since it is purely artificial, z_k must be set to zero in the solution of the initial problem, while maintaining both dual feasibility and complementarity. This typically requires modifying the multipliers associated to the constraints involving x_k and, as a consequence, the duals z_j of the other variables appearing in these constraints.

The simpler case is when an equality doubleton constraint is used to transfer the bounds on x_j to x_k, as explained in Section 3.2.3. In this situation, we may distinguish two cases. If none of the bounds in (3.3) or (3.4) is active, this means that the original bounds on x_j are not active at the solution: they are therefore irrelevant and could have been forgotten from the problem. Thus

$$z_j = z_j^+ (= 0), \quad z_k = z_k^+ \quad \text{and} \quad y_i = y_i^+.$$

If, on the other hand, one of the bounds in (3.3) or (3.4) is active at the solution of the transformed problem, we first deduce that z_k must be zero. Moreover, (1.2) and the fact $z_j^+ = 0$ by design yield that

$$\begin{aligned} a_{ij} y_i + z_j &= a_{ij} y_i^+, \\ a_{ik} y_i &= a_{ik} y_i^+ + z_k^+. \end{aligned}$$

We then immediately deduce that

$$y_i = y_i^+ + \frac{1}{a_{ik}} z_k^+ \quad \text{and} \quad z_j = -\frac{a_{ij}}{a_{ik}} z_k^+. \tag{6.1}$$

The same type of reasoning applies to the slightly more complex case where bounds on x_j are tightened as the result of the analysis of the i-th primal constraint (see Section 3.2.5). In this case, the i-th constraint may involve more than two variables, which means that we have to replace the second part of (6.1) by

$$z_j = -\frac{a_{ij}}{a_{ik}} z_k^+ \text{ for all } j \mid a_{ij} \neq 0. \tag{6.2}$$

The most intricate case is when a bound on x_k is imposed as the result of the analysis of dual constraints, that is via (3.24)–(3.27). In this case, the multipliers of all constraints i such that $a_{ik} \neq 0$ must be considered, together with the duals of all variables involved in these constraints. Fortunately, this exploration and the associated updating of the multipliers and dual variables is typically very fast.

6.3. Freeing a variable by splitting an equality constraint

Transfering the bounds on a variable into bounds on a split equality constraint (as described at the end of Section 3.2.3) is another case where recovering the dual variables and multipliers requires some care. If $\hat{y}_i^+$ denotes the multiplier associated with the inequality constraint of (3.5) and y_i^+ the multiplier associated with the equality constraint, these values satisfy the dual equations

$$g_j + \sum_{\ell=1}^{n} h_{\ell j} x_\ell - \sum_{\substack{p=1 \\ p\neq i}}^{m} a_{pj} y_p^+ - a_{ij} y_i^+ = 0$$

and

$$g_k + \sum_{\ell=1}^{n} h_{\ell k} x_\ell - \sum_{\substack{p=1 \\ p\neq i}}^{m} a_{pk} y_p^+ - a_{ik} y_i^+ - a_{ik} \hat{y}_i^+ = z_k^+$$

for $k \neq j$, while we also have, for the problem before the transformation, that

$$g_k + \sum_{\ell=1}^{n} h_{\ell k} x_\ell - \sum_{p=1}^{m} a_{pk} y_p = z_k$$

for $k = 1, \dots, n$. It can then be verified that we may recover the value of z_j and y_i from the relations

$$z_j = -a_{ij} \hat{y}_i^+ \quad \text{and} \quad y_i = y_i^+ + \hat{y}_i^+,$$

the dual variables of the variables different from j and the other multipliers being unmodified.

6.4. Forcing primal constraints

Consider now the problem transformation that removes a forcing primal constraint and fixes the values of each of the variables occuring in this constraint to its lower or upper bound (as discussed in Section 3.2.4). We then use the following strategy, proposed by Fourer and Gay (1994), to recover the associated dual variables and multiplier. Let J be the index set of the variables fixed when removing the forcing constraint i. If u_i (as defined in (3.7)) is equal to $\underline{c}_i$, we know that $y_i \geq 0$ and choose it as the smallest number which is sufficiently positive to ensure that

$$z_j \leq 0 \text{ for all } j \mid a_{ij} > 0 \qquad \text{and} \qquad z_j \geq 0 \text{ for all } j \mid a_{ij} < 0,$$

while we choose y_i to be the largest non-positive number ensuring that

$$z_j \geq 0 \text{ for all } j \mid a_{ij} > 0 \qquad \text{and} \qquad z_j \leq 0 \text{ for all } j \mid a_{ij} < 0$$

if ℓ_i (as defined in (3.8)) is equal to $\overline{c}_i$.

6.5. Linear combination of constraints

If we consider the transformation that add a multiple of an (equality) constraint to another constraint (such as in Section 3.2.6, where this preocedure is used to make A sparser), the formulae for the corresponding multipliers are easy to find. Specifically, if (3.11) describes such a transformation, we have that

$$y_k = y_k^+ \qquad \text{and} \qquad y_i = y_i^+ - \frac{a_{kj}}{a_{ij}} y_k^+.$$

6.6. Constraint elimination using linear doubleton columns

Another case of interest is the undoing of a transformation where a linear (implied) free doubleton column is used to eliminate one of the variable from an equality constraint (see Section 3.3.2). If variable j is eliminated by substitution from constraint i into constraint k, the j-th component of (1.2) gives (3.17) since variable j must be linear and (implied) free. On the other hand, the same condition on the transformed problem gives that

$$g^+ + H^+ x^+ - [A^+]^T y^+ = z^+.$$

We may then write the ℓ-th component of this identity (where we assume that x_ℓ occurs in the transformed problem) as

$$g_\ell^+ - a_{k\ell}^+ y_k^+ = g_\ell - \frac{a_{i\ell}}{a_{ij}} g_l - a_{k\ell} y_k^+ + \frac{a_{ij}}{a_{kj}} a_{i\ell} y_k^+ = \sum_{\substack{p=1 \\ p\neq k}}^{m} a_{p\ell}^+ y_p^+ + [z^+]_\ell - [H^+ x^+]_\ell \tag{6.3}$$

where we used the second part of (3.14) with (3.12) and (3.16). Now observe that, since x_j is linear, $H = H^+$ and $[H^+ x^+]_\ell = [Hx]_\ell$. Moreover, $a_{p\ell} = a^+_{p\ell}$ for $p \neq k$, $p \neq i$ and $\ell \neq i$ since the transformation only alters rows k and i in A. It is now easy to verify that the choices

$$[z_k]_\ell = [z_k^+]_\ell \quad (\ell \neq j), \qquad \text{and} \qquad z_j = 0,$$

together with

$$y_k = y_k^+, \qquad \text{and} \qquad y_i = \frac{1}{a_{ij}} \left(g_j - a_{kj} y_k \right)$$

(the last equality resulting from (3.17)) translate the optimality conditions of the transformed problem into optimality conditions for the original one. In particular, (6.3) reduces to the ℓ-th component of (1.2) for the untransformed problem.

7. Preliminary numerical experience

We now report some numerical experience gained with our Fortran 90 implementation of the ideas discussed above. The resulting package, PRESOLVE is an integral component of the GALAHAD optimization library (see Gould, Orban and Toint, 2003b). In this implementation, the status and bounds on the problems quantities is first verified according to Section 3.1 and the heuristics discussed above are then applied in a loop, until no further reduction in the problem dimensions (that is n, m and the numbers of nonzeros in A and H) is obtained. Within each loop, the heuristics are applied in the following order:

1. remove empty and singletons rows, as indicated in Sections 3.2.1 and (3.2.2),
2. try to eliminate variables that are linearly unconstrained, as outlined in Section 3.4;
3. attempt to exploit the presence of linear singleton columns, as discussed in Section 3.3.1;
4. attempt to exploit the presence of linear doubleton columns, as explained in Section 3.3.2;
5. complete the analysis of the dual constraints, using the techniques of Section 3.3.3 and 3.3.4;
6. remove empty and singletons rows, as indicated in Sections 3.2.1 and (3.2.2),
7. possibly remove dependent variables, as described in Section 3.3.5;
8. analyze the primal constraints, using the techniques described in Sections 3.2, except the sparsification procedure of Section 3.2.6;
9. try to make A sparser as explained in Section 3.2.6;
10. check the current status of the variables, dual variables and multipliers, using the various implications described in Section 3.1.

The resulting problem is finally permuted to comply with the format described in Section 5.

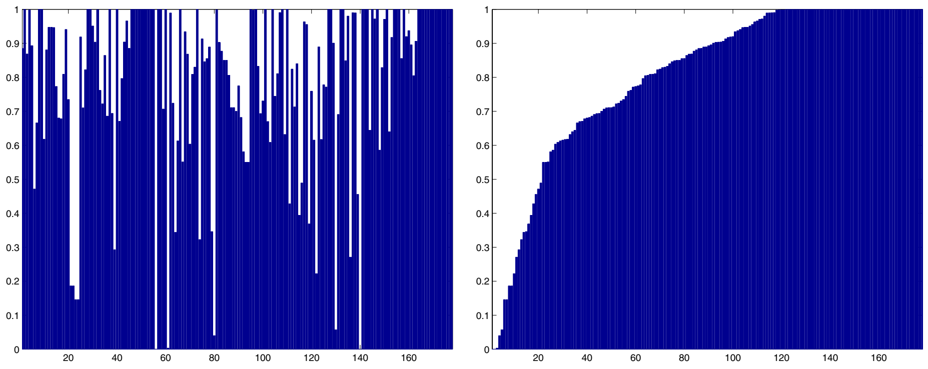

Fig. 1. The reduction obtained by presolving (for 178 problems) on the number of problem variables, sorted by decreasing original problem size (left) and by decreasing reduction (right)

7.1. Problem reduction

We start by considering the efficiency of the presolving process in terms of reduction of the problem sizes. Figures 1–4 report the effect of presolving on a collection of 178 linear and quadratic programs, of which 7 only involve only simple bounds, from the CUTEr collection of test problems (see Bongartz, Conn, Gould and Toint, 1995 and Gould, Orban and Toint, 2003a) which includes the Netlib linear programs and 68 problems with a non-trivial quadratic term. Results were obtained on a Pentium (biprocessor) running Linux Red Hat 7.2 with 1024 MBytes of memory, using the frt Fujitsu Fortran compiler with full optimization.

Figure 1, report, for each problem, the ratio of the number of variables in the reduced problem to that in the original problem, sorted by decreasing size of the original problem (in the left part of the figure) and by decreasing obtained reduction (in the right part). The left picture shows that there is little relation between the original size of a problem and its potential for a substantial reduction in its number of variables. The right picture shows the whole range of reduction, from the spectacular total reduction cases on the left to cases (approximately one third) were no reduction in n is obtained. The number of problem variables is reduced by 19% on average across all problems. This average reduction is of 20% if one considers linear problems only, and of 18% if one considers quadratic ones (that is problems with $|H| > 0$) only. Thus, despite the fact that quadratic programs may typically produce more coupling between the problem's variables than linear ones (and thus potentially less opportunity for applying reduction heuristics), they nevertheless also seem amenable to problem reduction.

Figures 2–4 show the corresponding reduction ratios for the number of constraints m, the number of nonzero entries in A and in H. The first of these figures indicates that the potential in reducing m (on average by 20%) is equivalent to that of reducing n, the amount of reduction being again uncorrelated to the original value of m. Figures 3 and 4 show that the reduction of the number of nonzero entries in A and H follow the same general pattern. The difference between linear and quadratic problems is marginal for these two latter statistics. We also observe, that the average reduction in the size of H (25%) exceeds that in the size of A (20%), which indicates again that the potential of

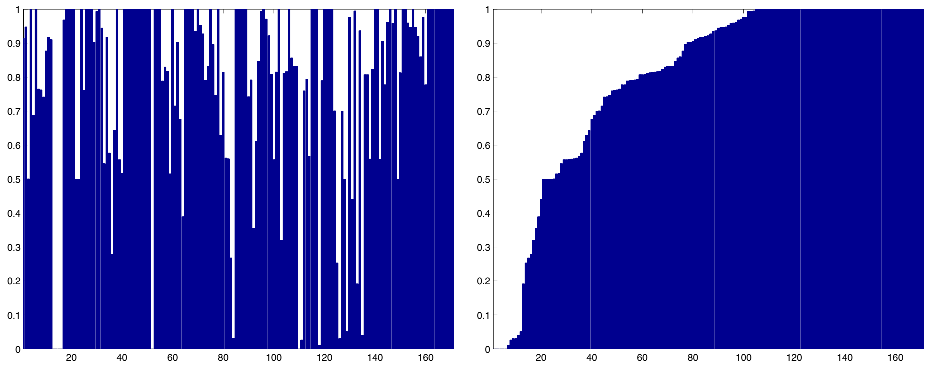

Fig. 2. The reduction obtained by presolving (for 171 problems with $A \neq 0$) on the number of constraints, sorted by decreasing number of constraints in the original problem (left) and by decreasing reduction (right).

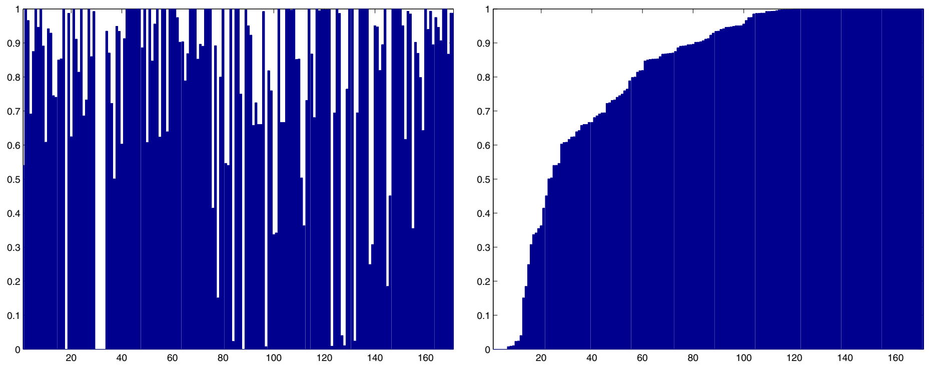

Fig. 3. The reduction obtained by presolving (for 171 problems with $A \neq 0$) on the number of nonzero entries in A, sorted by decreasing number of such entries in the original problem (left) and by decreasing reduction (right).

presolving quadratic programming problems is comparable to that of linear programs, at least in terms of problem size reduction.

Tables A.1–A.2 (in Appendix A) provide details of these results. In these (and later) tables, the CPU-times are given in seconds. Examination of these detailed statistics first shows that the problem reduction obtained for linear programs compares well with that obtained with specialized LP presolving tools. If we compare them with the results reported by Gondzio (1997), we obtain smaller n for 65 of the 93 comparable problems, smaller m for 47 and smaller number of nonzeros in A for 43, Gondzio's results showing smaller n for 17 problems, smaller m for 33 problems and smaller number of nonzeros in A for 41 problems, the rest being ties. The difference with Gondzio's results is not surprising, as, on one hand, we have not implemented the dense column splitting technique that he describes and have added a few new technique such as equality constraint splitting or doubleton column handling. Furthermore, slight differences in way transformations are sequenced do lead to different results. A second point of interest is that comparison of the presolve times with numerical solution times for the tested

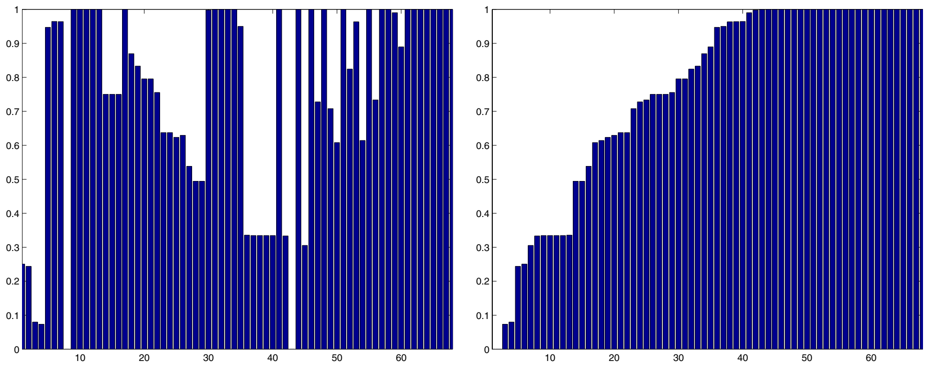

Fig. 4. The reduction obtained by presolving (for 68 problems with $H \neq 0$) on the number of nonzero entries in H, sorted by decreasing number of such entries in the original problem (left) and by decreasing reduction (right).

problems (see below) also shows that presolve time remain marginal, as desired, both for linear and quadratic examples. While it is not surprising that problems QPCBLEND, QPCBOIE1, QPCBOEI2 and QPCSTAIR are very comparable in this respect with their linear counterparts BLEND, BOEING1, BOEING2 and STAIR since their Hessians are diagonal, we notice the full range of possibilities for problems with denser H: from little or no reduction for BLOCKQP1, BLOCKQP5, DTOC3, GMNCASE1, GMNCASE3 or YAO to more interesting cases like AUG2D, DUALC8, MOSARQP1, NCVXQP4, PRIMAL2, STATIC3 or UBH1, up to the very successful examples such as GMNCASE4 and SOS-QP1 in which the presolve removes all the variables and constraints, and thus reveals the complete solution to the QP under consideration. We finally consider the space needed to perform the necessary factorizations of the problem matrices with and without presolving. Tables A.1–A.2 report, under the headings "int ratio" and "real ratio", the ratios of the integer and real workspace needed using the presolver to the corresponding number without presolve. We note that presolving reduces this space in nearly all cases, and that ratios above 1.00 are only marginally so for linear problems. This is not always the case for quadratic programs: the extreme case of CVXQP2 arises, for instance, because the initial (simple) preconditioner is ineffective and a switch to a more sophisticated (and expensive but effective) one is made by the solver for the reformulated problem, while the simple (cheaper but less effective) preconditioner suffices for the original. This also explains why the presolved version nevertheless solves much faster.

7.2. Impact on solution time

We now turn to the impact of presolving on the total solution time for linear and quadratic programs. As for problem reduction, we start by presenting a graphical comparison of the three (tightest, medium and loosest, see Section 4) presolving modes with the case where no presolving is applied. The quadratic solver used is QPB, run in default mode, also from GALAHAD, an interior point algorithm for nonconvex quadratic programs described in Conn, Gould, Orban and Toint (2000), Gould, Orban, Sartenaer and Toint (2002) or

Table 1. Failure causes for problem solution.

Presolving mode	apparently infeasible	conditioning too large	total
none	9	7	16
tightest	3	3	6
medium	3	4	7
loosest	3	2	5

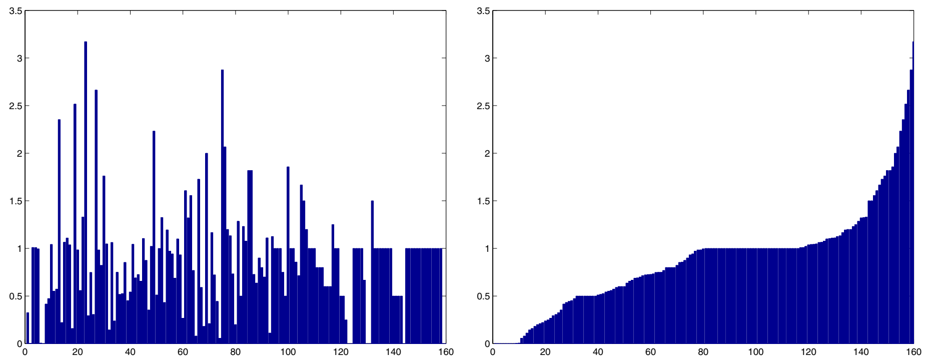

Fig. 5. The reduction (or increase) of solution time with presolving in tightest mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

Gould and Toint (2002). Note that we excluded three problems from the comparison: `STATIC3`, which is unbounded below, `NCVXQP3` for which different local solutions were found for the presolved and original formulations, and `DFL001` for which our compiler generated a swapping error.

We first report in Table 1 the causes of failure to solve the (possibly reduced) problem for all presolving modes (including no presolving at all).

Figure 5 shows the ratio of the solution time required by the solver with presolving (in tightest mode) to that required without presolving, for each of the 160 problems where both solves were successful. As expected, the impact of presolving on solution times is variable. This impact is very favourable[3] in 39 of the 160 compared problems, favourable[4] for 35 problems, relatively neutral[5] for 53, unfavourable[6] for 25 and very unfavorable[7] for 8. The average gain is 11% (10% for linear problems and 13% for quadratic ones).

Figures 6 and 7 show the same ratios for the medium and loosest presolving modes, respectively. For the medium mode, the impact is very favourable for 37 problems, favourable for 32, neutral for 54, unfavourable for 26 and very unfavourable for 12, with an average gain of 2% (9% on linear problems and 0% on quadratic ones). For the loosest mode, the impact is very favourable for 41 problems, favourable for 36, neutral for 52,

[3] Better by a factor at least 2.

[4] Between 10% and 100% better.

[5] Difference smaller than 10%.

[6] Between 10% and 100% worse.

[7] Worse by a factor at least 2.

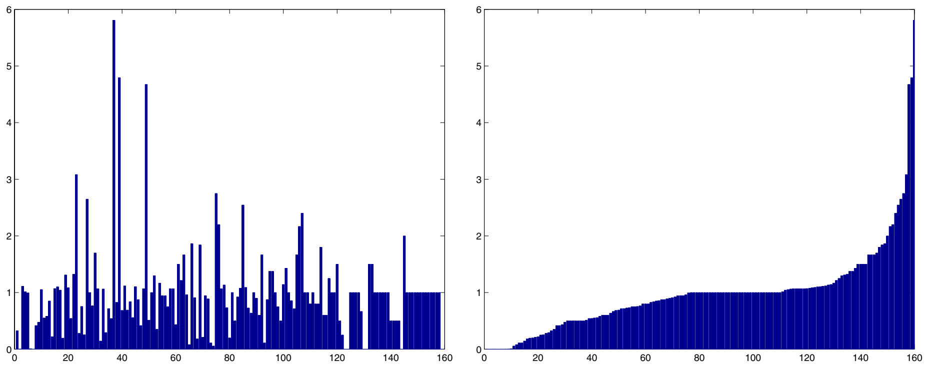

Fig. 6. The reduction (or increase) of solution time with presolving in medium mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

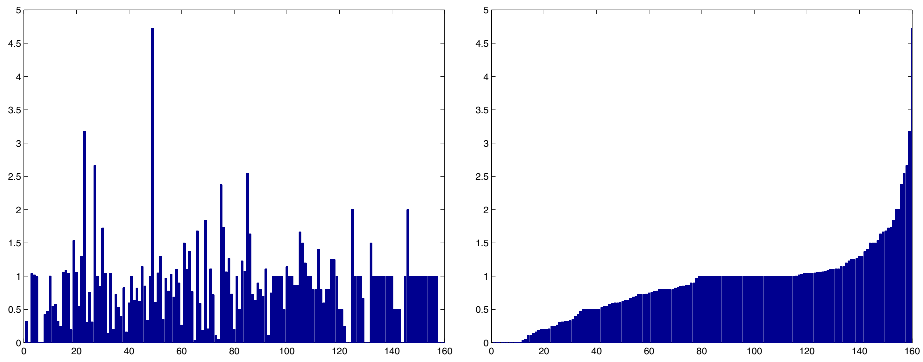

Fig. 7. The reduction (or increase) of solution time with presolving in loosest mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

unfavourable for 25 and very unfavourable for 7, with an average gain of 12% (14% on linear problems and 8% on quadratic ones). The loosest mode therefore appears to be of real interest from the efficiency point of view, especially for linear problems. Note also that the choice between the different presolving modes could well depend on the solver used.

Tables B.1 and B.2 (in Appendix B) report the detailed solution times for each the problem and for each presolving mode ("(t)" for tightest, "(m)" for medium and "(l)" for loosest). Failures are indicated in these tables by a "-". In each case, the "c" column gives an indication of the final status reported by the solver, except in unquestionably successful cases. The symbol "cond." means that the solution returned is the best that could be found given the high conditioning of the problem. The solution time is nevertheless reported for those problems where the found solution coincides with the correct solution by at least five digits of accuracy in the objective function value. The symbol "stp." indicates that the solver was stopped because the step had become so small that further progress was impossible. Note that none of these two cases preclude a satisfactory solution. The symbol "inf." indicates that the solver erroneously concluded that the problem is primal infeasible, "mem." indicates that the necessary memory space could not be allocated, “swap” that a swapping error was generated, and “unb.” that the problem is unbounded below.

Globally speaking, the effect of presolving on solution time therefore seems to be relatively positive.

8. Conclusion and perspectives

We have described a set of presolving techniques that can be applied on linear and quadratic optimization problems. For the latter class, the problem reduction exploits the fact that both variables and multipliers appear together in the dual feasibility condition, leading to transformations that are specific to the quadratic case. Numerical experience with the resulting code indicate that, despite their stronger inner coupling, quadratic problems appear to be as amenable to presolving as linear ones, both in terms of reduced problem size and reduced solution time. The resulting thread-safe Fortran 90 package `PRESOLVE` is available without cost as part of the GALAHAD optimization library (see Gould et al., 2003b).

Acknowledgements. The authors wish to thank J. Gondzio for his encouraging comments while this work was in progress and to P. Y. Bernard and P. Lambrecht for proofreading an early version of the manuscript and helping to debug the code. Thanks are also due to the associate editor and the referees for their helpful suggestions.

Appendix A: The effect of presolving on problem size

Table A.1. The effect of presolving on LP dimensions

Problem	Before presolve			After presolve			int	real	time
	m	n	$\|A\|$	m	n	$\|A\|$	ratio	ratio	
`25FV47`	821	1571	10400	741	1468	9870	0.88	0.84	0.1
`80BAU3B`	2262	9799	21002	1984	8752	19139	5.44	1.64	0.5
`ADLITTLE`	56	97	383	53	89	360	0.96	0.95	0.0
`AFIRO`	27	32	83	21	29	72	0.67	0.80	0.0
`AGG2`	516	302	4284	289	233	2315	0.53	0.37	0.0
`AGG3`	516	302	4300	290	235	2349	0.51	0.38	0.0
`AGG`	488	163	2410	173	105	875	0.37	0.31	0.0
`BANDM`	305	472	2494	173	202	1256	0.46	0.51	0.0
`BEACONFD`	173	262	3375	7	15	26	0.02	0.00	0.0
`BLEND`	74	83	491	70	71	427	0.92	0.98	0.0
`BNL1`	643	1175	5121	535	1073	4560	0.88	0.94	0.0
`BNL2`	2324	3489	13999	1724	2869	12040	0.82	0.87	0.2
`BOEING1`	351	384	3485	292	367	2303	0.26	0.39	0.0
`BOEING2`	166	143	1196	134	139	1133	0.99	1.00	0.0
`BORE3D`	233	315	1429	59	70	356	0.21	0.23	0.0
`BRANDY`	220	249	2148	110	172	1492	0.57	0.57	0.0
`CAPRI`	271	353	1767	214	268	1351	0.39	0.40	0.0
`CYCLE`	1903	2857	20720	1448	2063	16873	0.27	0.45	0.1
`CZPROB`	929	3523	10669	479	2502	5343	0.43	0.41	0.4
`D2Q06C`	2171	5167	32417	1990	4181	30120	0.93	0.91	1.1
`D6CUBE`	415	6184	37704	403	5447	33601	0.89	0.97	0.7

Table A.1. Continued

Problem	Before presolve			After presolve			int	real	time
	m	n	$\lvert A\rvert$	m	n	$\lvert A\rvert$	ratio	ratio	
DEGEN2	444	534	3978	441	529	3968	1.02	0.96	0.0
DEGEN3	1503	1818	24646	1493	1799	24334	0.99	0.96	0.1
DEGENLPA	15	20	82	15	20	81	1.00	1.00	0.0
DEGENLPB	15	20	82	15	20	81	1.00	1.00	0.0
DFL001	6071	12230	35632	5754	10626	33559	–	–	1.4
E226	223	282	2578	156	254	2199	0.67	0.80	0.0
ETAMACRO	400	688	2409	326	461	1762	0.77	0.84	0.0
FFFFF800	524	854	6227	427	624	4912			0.0
FINNIS	497	614	2310	369	457	1573	0.79	0.72	0.0
FIT1D	24	1026	13404	24	1025	13308	1.00	1.00	0.1
FIT1P	627	1677	9868	627	1028	9219	0.99	1.00	0.1
FIT2D	25	10500	129018	25	10485	128882	1.00	1.00	4.8
FIT2P	3000	13525	50284	3000	13525	50284	1.00	1.00	3.7
FORPLAN	161	421	4563	90	206	1892	0.60	0.55	0.0
GANGES	1309	1681	6912	714	578	4418	0.60	0.32	0.1
GFRD-PNC	616	1092	2377	460	934	2063	0.86	0.79	0.0
GREENBEA	2392	5405	30877	1830	3678	23011	0.70	0.68	0.3
GREENBEB	2392	5405	30877	1824	3665	22864	0.32	0.18	0.3
GROW15	300	645	5620	300	645	5620	1.00	1.00	0.0
GROW22	440	946	8252	440	946	8252	1.00	1.00	0.0
GROW7	140	301	2612	140	301	2612	1.00	1.00	0.0
ISRAEL	174	142	2269	163	142	2258	0.71	0.95	0.0
KB2	43	41	286	42	33	256	0.97	0.90	0.0
LOTFI	153	308	1078	126	274	965	0.83	0.86	0.0
MAROS	846	1443	9614	605	871	5797	0.61	0.54	0.1
MAROS-R7	3136	9408	144848	2156	4435	78295	0.40	0.75	1.5
MODEL	339	1831	1893	9	6	20	0.06	0.04	0.0
MODSZK1	687	1620	3168	654	893	2407	0.88	0.92	0.1
NESM	662	2923	13288	614	2227	12421	0.95	0.96	0.2
OET1	1002	3	3004	1002	3	3004	1.00	1.00	0.0
PEROLD	625	1376	6018	560	1113	5228	0.85	0.65	0.0
PILOT	1441	3652	43167	1361	3356	40855	0.98	0.98	0.3
PILOT4	410	1000	5141	378	775	4609	0.56	0.60	0.0
PILOT87	2030	4883	73152	1966	4595	70683	0.97	1.08	0.5
PILOT-JA	940	1988	14698	768	1405	10768	0.92	0.73	0.1
PILOTNOV	975	2172	13057	809	1730	11369	–	–	0.1
PILOT-WE	722	2789	9126	675	2411	8332	0.18	0.32	0.1
PT	501	2	1002	501	2	1002	1.00	1.00	0.0
QAP12	3192	8856	38304	3192	8856	38304	1.00	1.00	1.7
QAP8	912	1632	7296	912	1632	7296	1.00	1.00	0.1
READING2	200	303	800	88	187	361	0.11	0.14	0.0
RECIPELP	91	180	663	74	82	409	0.81	0.85	0.0
SC105	105	103	280	101	100	273	0.99	0.97	0.0
SC205	205	203	551	200	199	543	0.97	1.00	0.0
SC50A	50	48	130	46	45	123	0.95	0.99	0.0
SC50B	50	48	118	43	43	107	0.98	0.85	0.0
SCAGR25	471	500	1554	288	316	1080	0.65	0.94	0.0
SCAGR7	129	140	420	72	82	270	0.52	0.80	0.0
SCFXM1	330	457	2589	262	384	2205	0.28	0.33	0.0
SCFXM2	660	914	5183	522	761	4420	0.79	0.86	0.1
SCFXM3	990	1371	7777	781	1138	6589	0.79	0.84	0.1
SCORPION	388	358	1426	124	132	439	0.39	0.42	0.0
SCRS8	490	1169	3182	388	989	2605	0.81	0.69	0.1
SCSD1	77	760	2388	77	760	2388	1.00	1.00	0.0
SCSD6	147	1350	4316	147	1350	4316	1.00	1.00	0.0
SCSD8	397	2750	8584	397	2750	8584	1.00	1.00	0.0

Table A.1. Continued

Problem	Before presolve			After presolve			int	real	time
	m	n	$\|A\|$	m	n	$\|A\|$	ratio	ratio	
SCTAP1	300	480	1692	300	480	1692	1.00	1.00	0.0
SCTAP2	1090	1880	6714	1090	1880	6714	1.00	1.00	0.0
SCTAP3	1480	2480	8874	1480	2480	8874	1.00	1.00	0.1
SEBA	515	1028	4352	138	41	658	0.06	0.52	0.1
SHARE1B	117	225	1151	106	191	943	0.98	0.78	0.0
SHARE2B	96	79	694	92	79	660	0.98	0.94	0.0
SHELL	536	1775	3556	337	1285	2576	0.72	0.71	0.1
SHIP04L	402	2118	6332	325	1915	5723	0.90	0.91	0.1
SHIP04S	402	1458	4352	224	1266	3481	0.83	0.77	0.1
SHIP08L	778	4283	12802	526	3147	9249	0.72	0.74	0.2
SHIP08S	778	2387	7114	303	1601	4436	0.65	0.62	0.1
SHIP12L	1151	5427	16170	664	4196	11092	0.71	0.68	0.4
SHIP12S	1151	2763	8178	321	1895	4969	0.54	0.57	0.1
SIERRA	1227	2036	7302	1126	1967	6980	0.95	0.94	0.1
SIPOW1	2000	2	4000	2000	2	4000	1.00	1.00	0.0
SIPOW1M	2000	2	4000	2000	2	4000	1.00	1.00	0.0
SIPOW2	2000	2	3000	1000	2	2000	0.55	0.58	0.0
SIPOW2M	2000	2	3000	1000	2	2000	0.55	0.58	0.0
SIPOW3	2000	4	5992	1998	4	5990	1.00	1.00	0.0
SIPOW4	2000	4	7000	2000	4	7000	1.00	1.00	0.0
SSEBLIN	72	194	312	72	192	310	1.00	1.00	0.0
STAIR	356	467	3856	305	333	3560	0.66	0.79	0.0
STANDATA	359	1075	3031	293	372	1037	0.84	0.76	0.0
STANDGUB	361	1184	3139	293	382	1057	0.84	0.76	0.0
STANDMPS	467	1075	3679	395	956	2421	0.79	0.80	0.1
STOCFOR1	117	111	447	91	92	357	0.51	0.30	0.0
STOCFOR2	2157	2031	8343	1964	1798	7392	0.95	0.82	0.1
STOCFOR3	16675	15695	64875	15236	13892	56774	0.65	0.54	3.4
TRUSS	1000	8806	27836	1000	8806	27836	1.00	1.00	0.2
TUFF	333	587	4520	253	476	4032	0.28	0.32	0.0
VTP-BASE	198	203	908	38	55	168	0.03	0.06	0.0
WOOD1P	244	2594	70215	171	1800	48552	0.87	0.48	0.6
WOODW	1098	8405	37474	706	5194	22800	0.39	0.66	0.4

Table A.2. The effect of presolving on QP dimensions

Problem	Before presolve				After presolve				int	real	time
	m	n	$\|A\|$	$\|H\|$	m	n	$\|A\|$	$\|H\|$	ratio	ratio	
AUG2D	1600	3280	6400	3120	1444	2964	5776	2964	0.93	0.89	0.0
AUG2DC	1600	3280	6400	3280	1600	3280	6400	3280	1.00	1.00	0.1
AUG2DCQP	1600	3280	6400	3280	1600	3280	6400	3280	1.00	1.00	0.0
AUG2DQP	1600	3280	6400	3120	1599	3120	6238	3120	1.03	1.00	0.1
BLOCKQP1	1001	2005	9005	3005	1001	2005	8998	1005	1.00	1.00	0.1
BLOCKQP2	1001	2005	9005	3005	1001	2005	8999	1005	1.00	1.00	0.1
BLOCKQP3	1001	2005	9005	3005	1001	2005	8998	1005	1.00	1.00	0.1
BLOCKQP4	1001	2005	9005	3005	1001	2005	8999	1005	1.00	1.00	0.1
BLOCKQP5	1001	2010	14010	3010	1001	2010	13998	1010	1.00	1.00	0.1
BLOWEYA	1002	2002	5003	4003	1002	2002	5003	3002	0.09	0.05	0.1
BLOWEYB	1002	2002	5003	4003	1002	2002	5003	3002	0.14	0.06	0.1
BLOWEYC	1002	2002	5003	4003	1002	2002	5003	3002	0.59	0.44	0.1
BQPGABIM	0	50	0	172	0	46	0	153	1.00	1.00	0.0

Table A.2. Continued

Problem	Before presolve				After presolve				int	real	time
	m	n	$\|A\|$	$\|H\|$	m	n	$\|A\|$	$\|H\|$	ratio	ratio	
BQPGAUSS	0	2003	0	9298	0	2003	0	9298	1.00	1.00	0.0
CVXQP1	500	1000	1498	3984	500	700	1498	2484	0.80	0.62	0.0
CVXQP2	250	1000	749	3984	250	550	749	1969	1.21	12.59	0.0
CVXQP3	750	1000	2247	3984	750	850	2243	3169	0.90	0.78	0.0
DTOC3	1998	2999	6993	2997	1997	2996	6986	2996	0.99	1.00	0.0
DUAL1	1	85	85	3558	1	85	85	3558	1.00	1.00	0.0
DUAL2	1	96	96	4508	1	96	96	4508	1.00	1.00	0.0
DUALC1	215	9	1935	45	11	9	78	45	0.02	0.05	0.0
DUALC2	229	7	1603	28	7	7	39	28	0.00	0.01	0.0
DUALC5	278	8	2224	36	3	8	20	36	0.00	0.00	0.0
DUALC8	503	8	4024	36	16	8	94	36	0.00	0.01	0.0
GMNCASE1	300	175	23940	11803	300	175	23940	11802	0.92	0.52	0.0
GMNCASE2	1050	175	28546	11803	543	175	24361	11803	0.30	0.81	0.1
GMNCASE3	1050	175	28546	11803	585	175	24266	11803	0.32	0.82	0.1
GMNCASE4	350	175	27510	15330	0	0	0	0	0.00	0.00	0.1
GOULDQP2	999	1999	2997	1997	999	1999	2997	1997	1.00	1.00	0.0
GOULDQP3	999	1999	2997	3995	999	1999	2997	3995	1.00	1.00	0.0
HUESTIS	2	1000	2000	1000	2	1000	1974	1000	1.00	1.00	0.0
JNLBRNG1	0	5625	0	16725	0	5329	0	15841	1.00	1.00	0.4
KSIP	1001	20	20001	20	1000	20	20000	20	1.00	1.00	0.1
LISWET8	100	103	400	103	100	103	400	103	0.98	0.94	0.0
MOSARQP1	700	2500	3422	2545	700	732	3397	777	0.62	0.95	0.0
MOSARQP2	600	900	2930	945	600	624	2921	669	0.90	0.99	0.0
NCVXBQP1	0	1000	0	3984	0	998	0	3976	1.00	1.00	0.0
NCVXQP1	500	1000	1498	3984	500	711	1498	2539	0.32	0.18	0.0
NCVXQP2	500	1000	1498	3984	500	711	1498	2539	0.74	0.53	0.0
NCVXQP3	500	1000	1498	3984	500	806	1498	3010	1.80	9.40	0.0
NCVXQP4	250	1000	749	3984	250	550	749	1969	0.21	0.20	0.0
NCVXQP5	250	1000	749	3984	250	581	749	2144	0.21	0.20	0.0
NCVXQP6	250	1000	749	3984	250	682	749	2507	0.21	0.20	0.0
NCVXQP7	750	1000	2247	3984	750	850	2243	3169	1.01	0.89	0.0
NCVXQP8	750	1000	2247	3984	750	877	2243	3319	0.99	0.94	0.0
NCVXQP9	750	1000	2247	3984	750	903	2243	3464	0.93	0.96	0.0
NOBNDTOR	0	100	0	240	0	64	0	176	1.00	1.00	0.0
OBSTCLAE	0	5625	0	16425	0	5329	0	15841	1.00	1.00	0.3
PRIMAL1	85	325	5815	324	85	200	5815	199	0.79	0.89	0.0
PRIMAL2	96	649	8042	648	96	395	8042	394	0.20	5.84	0.0
PRIMALC1	9	230	2070	229	9	230	2070	229	1.00	1.00	0.0
PRIMALC2	7	231	1617	230	7	231	1617	230	1.00	1.00	0.0
PRIMALC5	8	287	2296	286	8	287	2296	286	1.00	1.00	0.0
PRIMALC8	8	520	4160	519	8	520	4160	519	1.00	0.98	0.0
QPCBLEND	74	83	491	83	71	83	443	83	0.95	0.99	0.0
QPCBOEI1	351	384	3485	384	292	370	2303	370	0.62	0.68	0.0
QPCBOEI2	166	143	1196	143	134	143	1137	143	0.90	0.94	0.0
QPCSTAIR	356	467	3856	467	356	385	3666	385	0.70	1.00	0.0
SOSQP1	1001	2000	4000	3000	0	0	0	0	0.00	0.00	0.0
SOSQP2	1001	2000	4000	3000	1001	2000	3000	1000	1.00	1.00	0.2
STATIC3	96	434	496	1014	48	171	176	738	0.46	0.45	0.0
STCQP1	2052	4097	13338	26603	0	598	0	2123			0.1
STCQP2	2052	4097	13338	26603	0	763	0	6661	0.00	0.00	1.4
STNQP1	2052	4097	13338	26603	0	598	0	1952			0.1
STNQP2	2052	4097	13338	26603	0	763	0	6488	0.00	0.00	0.1
TORSION1	0	5476	0	15984	0	5184	0	15408	1.00	1.00	0.3
UBH1	6000	9009	24000	3003	3003	6000	14991	3003	0.30	0.06	1.5
YAO	200	202	600	202	199	200	596	200	0.98	0.92	0.0

Appendix B: The effect of presolving on solution time

Table B.1. The effect of presolving on LP solution time

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
25FV47	12.7		12.5		12.7		12.7	
80BAU3B	23.8		26.4		26.2		26.0	
ADLITTLE	0.0		0.1		0.1		0.7	
AFIRO	0.0		0.7		0.0		0.0	
AGG	0.4		0.1		0.1		0.1	
AGG2	1.9		0.4		0.4		0.4	
AGG3	2.2		0.4		0.4		0.4	
BANDM	0.5		0.4		0.4		0.4	
BEACONFD	0.2		0.0		0.0		0.0	
BLEND	0.1		0.1		0.1		0.1	
BNL1	2.8		3.7		3.4		3.1	
BNL2	18.8		10.5		10.2		10.2	
BOEING1	0.9		1.0		1.5		1.0	
BOEING2	0.3		0.3		0.3		0.3	
BORE3D	0.2		0.1		0.1		0.1	
BRANDY	0.4		0.4		0.4		0.4	
CAPRI	1.1		0.8		0.8		0.8	
CYCLE	–	cond.	11.3		11.6		10.9	
CZPROB	2.8		4.5		4.2		4.2	
D2Q06C	92.6		53.0		53.9		53.0	
D6CUBE	23.4		24.3		24.4		24.5	
DEGEN2	1.5		3.1		3.3		2.6	
DEGEN3	12.8		34.1		33.9		34.1	
DEGENLPA	0.0		0.0		0.0		0.0	
DEGENLPB	–	inf.	0.0		0.0		0.0	
DFL001	1584.8	cond.	—	swap	—	swap	0.0	swap
E226	0.8		0.9		0.7		0.6	
ETAMACRO	1.8		2.1		1.7		2.0	
FFFFF800	–	inf.	1.6		2.1		1.6	
FINNIS	0.7		0.6		0.6		0.6	
FIT1D	4.0		4.0		4.0		4.2	
FIT1P	1.6		4.6		4.4		3.8	
FIT2D	234.0		233.1		232.8		232.2	
FIT2P	44.5		47.4		47.6		47.3	
FORPLAN	0.4		0.2		0.6		0.2	
GANGES	3.0		0.8		1.3		0.8	
GFRD-PNC	0.7		0.5		0.5		0.6	
GREENBEA	–	inf.	9.2		9.4		–	stp.
GREENBEB	45.6	cond.	10.1		10.0		11.3	
GROW15	0.7		0.7		1.0		0.7	
GROW22	1.0		0.9		1.0		0.9	
GROW7	0.3		0.2		0.2		0.2	
ISRAEL	3.2		2.2		2.4		2.2	
KB2	0.1		0.1		0.1		0.1	
LOTFI	0.2		0.1		0.1		0.1	
MAROS	14.3		4.2		4.0		4.3	
MAROS-R7	122.7		51.1		51.1		52.2	
MODEL	0.0		0.0		0.0		0.0	
MODSZK1	1.3		1.6		1.2		1.6	
NESM	8.8		6.6		6.3		6.4	
OET1	0.3		0.3		0.3		0.3	

Table B.1. Continued

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
PEROLD	7.5		6.4		6.2		6.2	
PILOT	100.3		55.3		55.3		55.1	
PILOT4	4.3		2.2		2.2		2.6	
PILOT87	913.2		296.6		297.0		297.9	
PILOT-JA	14.3		10.7		10.8		10.8	
PILOTNOV	–	inf.	–	inf.	–	inf.	–	inf.
PILOT-WE	6.8		4.7		4.7		4.3	
PT	0.4		0.2		0.2		0.2	
QAP12	–	inf.	–	inf.	–	inf.	–	inf.
QAP8	–	inf.	–	inf.	–	inf.	–	inf.
READING2	0.1		0.0		0.0		0.0	
RECIPELP	0.1		0.1		0.1		0.1	
SC105	0.0		0.1		0.0		0.0	
SC205	0.1		0.1		0.1		0.1	
SC50A	0.0		0.0		0.0		0.0	
SC50B	0.0		0.0		0.0		0.0	
SCAGR25	0.2		0.2		0.2		0.2	
SCAGR7	0.0		0.0		0.0		0.0	
SCFXM1	0.8		0.8		1.1		0.8	
SCFXM2	3.0		3.3		3.2		3.3	
SCFXM3	6.2		4.5		5.2		5.1	
SCORPION	0.1		0.1		0.1		0.1	
SCRS8	0.8		0.8		1.1		0.8	
SCSD1	0.1		0.1		0.1		0.1	
SCSD6	0.2		0.2		0.3		0.2	
SCSD8	0.4		0.5		0.5		0.5	
SCTAP1	0.5		0.4		0.4		0.7	
SCTAP2	3.6		3.5		3.4		2.8	
SCTAP3	4.8		4.2		4.2		4.1	
SEBA	1.5		0.3		0.3		0.3	
SHARE1B	0.2		0.1		0.1		0.1	
SHARE2B	0.1		0.1		0.1		0.1	
SHELL	0.5		0.5		0.4		0.4	
SHIP04L	0.6		1.0		1.0		1.0	
SHIP04S	0.8		0.6		0.6		0.8	
SHIP08L	1.1		2.0		2.8		2.8	
SHIP08S	0.7		0.7		0.7		0.7	
SHIP12L	2.2		3.8		4.1		3.7	
SHIP12S	1.0		0.8		0.9		0.8	
SIERRA	1.1		2.0		1.2		1.8	
SIPOW1	0.2		0.2		0.2		0.2	
SIPOW1M	0.2		0.2		0.2		0.2	
SIPOW2	1.8		0.1		0.1		0.1	
SIPOW2M	1.8		0.8		0.2		0.2	
SIPOW3	0.5		0.4		0.5		0.4	
SIPOW4	0.5		0.6		1.2		0.6	
SSEBLIN	0.0		0.0		0.0		0.0	
STAIR	0.6		0.9		1.3		0.9	
STANDATA	0.5		0.3		0.3		0.4	
STANDGUB	0.5		0.3		0.9		0.3	
STANDMPS	0.8		0.8		0.8		0.8	
STOCFOR1	0.1		0.1		0.1		0.1	
STOCFOR2	8.3		4.3		4.5		4.4	
STOCFOR3	119.8		56.7		57.0		56.2	

Table B.1. Continued

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
TRUSS	6.8		7.1		7.6		6.8	
TUFF	1.4		0.7		0.7		0.7	
VTP-BASE	0.1		0.0		0.0		0.0	
WOOD1P	18.2		24.2		24.1		23.6	
WOODW	11.3		19.9		19.2		19.5	

Table B.2. The effect of presolving on QP solution time

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)
	time	c	time	c	time	c	time
AUG2D	3.5		3.3		3.3		3.6
AUG2DC	3.6		4.3		4.2		3.5
AUG2DCQP	1.9		3.8		3.5		3.5
AUG2DQP	2.7		4.2		4.5		3.7
BLOCKQP1	1.5		1.7		1.7		1.9
BLOCKQP2	1.3		1.4		1.4		1.4
BLOCKQP3	0.7		1.3		0.8		0.8
BLOCKQP4	1.5		1.8		1.6		1.6
BLOCKQP5	1.4		1.8		1.4		1.4
BLOWEYA	9.7		1.4		1.4		1.4
BLOWEYB	232.1		1.4		1.4		2.4
BLOWEYC	4.8		1.7		2.0		1.6
BQPGABIM	0.1		0.1		0.1		0.0
BQPGAUSS	110.8		115.5		116.6		111.2
CVXQP1	7.0		3.8		4.8		4.2
CVXQP2	391.9	cond.	1.0		1.0		1.4
CVXQP3	12.4		10.2		9.5		10.5
DTOC3	9.5		10.1		10.1		9.9
DUAL1	1.5		1.1		1.1		1.1
DUAL2	10.4		10.9		11.1		10.9
DUALC1	0.4		0.0		0.0		0.0
DUALC2	0.3		0.0		0.0		0.0
DUALC5	0.4		0.0		0.0		0.0
DUALC8	0.9		0.1		0.1		0.1
GMNCASE1	22.0		3.5		4.3		4.3
GMNCASE2	6.1		4.0		3.4		3.8
GMNCASE3	13.7		4.2		3.5		4.3
GMNCASE4	170.7		0.1		0.1		0.1
GOULDQP2	0.1		0.1		0.2		0.1
GOULDQP3	0.3		0.3		0.3		0.6
HUESTIS	–	inf.	0.3		0.3		0.3
JNLBRNG1	309.7		319.1		314.6		315.3
KSIP	4.5		4.6		4.8		4.5
LISWET8	0.1		0.1		0.1		0.2
MOSARQP1	3.7		1.6		1.3		1.3
MOSARQP2	1.8		1.3		1.6		1.3
NCVXBQP1	0.1		0.1		0.1		0.1
NCVXQP1	7.8		4.1		45.3	cond.	3.1
NCVXQP2	7.3		3.3		35.0	cond.	1.2
NCVXQP3	12.1		167.3	cond.	35.0	cond.	17.9
NCVXQP4	0.5		0.3		0.3		0.4
NCVXQP5	0.8		0.4		0.4		0.4
NCVXQP6	9.2		2.2		2.7		1.8
NCVXQP7	4.3	cond.	9.6		20.1		20.3

Table B.2. Continued

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	
NCVXQP8	20.9		52.6		27.4		32.1	
NCVXQP9	54.3	cond.	127.8	cond.	46.2		17.5	
NOBNDTOR	0.0		0.0		0.0		0.0	
OBSTCLAE	298.0		301.0		331.2		310.4	
PRIMAL1	0.5		0.5		0.5		0.5	
PRIMAL2	1.1		0.7		0.7		0.7	
PRIMALC1	0.2		0.3		0.3		0.3	
PRIMALC2	0.3		0.3		0.3		0.3	
PRIMALC5	0.1		0.1		0.1		0.1	
PRIMALC8	–	stp.	–	stp.	–	stp.	–	stp.
QPCBLEND	0.1		0.1		0.1		0.1	
QPCBOEI1	2.2		1.3		2.0		1.3	
QPCBOEI2	0.5		0.5		0.5		0.5	
QPCSTAIR	3.7		4.9		4.8		4.8	
SOSQP1	0.3		0.0		0.0		0.0	
SOSQP2	1.0		0.7		0.6		0.7	
STATIC3	–	cond.	–	unb.	–	unb.	–	unb.
STCQP1	–	inf.	0.8		0.8		0.8	
STCQP2	15.7		49.8		48.4		49.9	
STNQP1	–	inf.	0.1		0.2		0.2	
STNQP2	2.5		0.2		0.2		0.1	
TORSION1	295.9		298.6		300.1		301.9	
UBH1	3.0		2.8		3.2		2.7	
YAO	0.2		0.1		0.1		0.2	

References

Andersen, E.D., Andersen, K.D.: Presolving in linear programming. Math. Program. **71**(2), 221–245 (1995)

Bongartz, I., Conn, A.R., Gould, N.I.M., Toint, Ph.L.: CUTE: Constrained and Unconstrained Testing Environment. Transactions of the ACM on Math. Softw. **21**(1), 123–160 (1995)

Bradley, G.H., Brown, G.G., Graves, G.W.: Structural redundancy in large-scale optimization models. In: M.H. Karwan et al., (ed.), Redundancy in Mathematical Programming, Springer Verlag, Heidelberg, 1983, pp. 145–169

Brearley, A.L., Mitra, G., Williams, H.P.: Analysis of mathematical programming problems prior to applying the simplex algorithm. Math. Program. **8**(1), 54–83 (1975)

Conn, A.R., Gould, N.I.M., Orban, D., Toint, Ph.L.: A primal-dual trust-region algorithm for minimizing a non-convex function subject to bound and linear equality constraints. Math. Program. **87**(2), 215–249 (2000)

Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Oxford University Press, Oxford, England, 1986

Ferris, M.C., Munson, T.S.: Preprocessing complementarity problems. In: M.C. Ferris, O.Mangasarian and J.S. Pang, (eds.), Complementarity: Applications, Algorithms and Extensions, Kluwer Academic Publishers, Dordrecht, 2001 pp. 143–164

Fourer, R., Gay, D.M.: Experience with a primal presolve algorithm. In: W.W. Hager, D.W. Hearn and P.M. Pardalos, (eds.), Large Scale Optimization: State of the Art, Kluwer Academic Publishers, Dordrecht, 1994, pp. 135–154

Gondzio, J.: Presolve analysis of linear programs prior to applying an interior point method. INFORMS J. Comput. **9**(1), 73–91 (1997)

Gould, N.I.M., Toint, Ph.L.: Numerical methods for large-scale non-convex quadratic programming. In: A.H. Siddiqi and M.Kočvara, (eds.), Trends in Industrial and Applied Mathematics, Kluwer Academic Publishers, Dordrecht, 2002 pp. 149–179

Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEr, a contrained and unconstrained testing environment, revisited. Transactions of the ACM on Mathematical Software, to appear (2003a)

Gould, N.I.M., Orban, D., Toint, Ph.L.: GALAHAD – a library of thread-safe Fortran90 packages for large-scale nonlinear optimization. Transactions of the ACM on Mathematical Software, to appear (2003b)

Gould, N.I.M., Orban, D., Sartenaer, A., Toint, Ph.L.: Componentwise fast convergence in solving nonlinear equations. Math. Program., Ser. B **92**(3), 481–508 (2002)

Ioslovich, I.: Robust reduction of a class of large-scale linear programs. SIAM J. Optim. **12**(1), 262–282 (2001)

Tomlin, J.A., Welch, J.S.: Formal optimization of some reduced linear programming problems. Math. Program. **27**(2), 232–240 (1983a)

Tomlin, J.A., Welch, J.S.: A pathological case in the reduction of linear programs. Oper. Res. Lett. **2**, 53–57 (1983b)