RAL-TR-2002-001

Preprocessing for quadratic programming

Nicholas I. M. Gould[1,2] and Philippe L. Toint[3,4]

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] Computational Science and Engineering Department, Rutherford Appleton Laboratory,
Chilton, Oxfordshire, OX11 0QX, England, EU.
Email : n.gould@rl.ac.uk

[2] Current reports available from "http://www.numerical.rl.ac.uk/reports/reports.html".

[3] Department of Mathematics, University of Namur,
61, rue de Bruxelles, B-5000 Namur, Belgium, EU.
Email : philippe.toint@fundp.ac.be

[4] Current reports available from "http://www.fundp.ac.be/~phtoint/pht/publications.html".

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxfordshire OX11 0QX

January 1st, 2002.

1 Introduction

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

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

where the vector of variables x as well as the vectors g, $\underline{x}$ and $\overline{x}$ belong to $\mathbb{R}^n$, 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 $\mathbb{R}^m$. 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.

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 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 (1983*a*, 1983*b*). General primal presolve techniques are also described in Fourer and Gay (1994). Presolving for 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

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

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

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

$$y_i \left\{ \begin{array}{lllllll} \geq 0 & \text{when} & \underline{c}_i & = & [Ax]_i, & & \\ = 0 & \text{when} & \underline{c}_i & < & [Ax]_i & < & \overline{c}_i, \\ \leq 0 & \text{when} & & & [Ax]_i & = & \overline{c}_i, \end{array} \right. \tag{1.5}$$

and

$$z_j \left\{ \begin{array}{lllllll} \geq 0 & \text{when} & \underline{x}_j & = & x_j, & & \\ = 0 & \text{when} & \underline{x}_j & < & x_j & < & \overline{x}_j, \\ \leq 0 & \text{when} & & & x_j & = & \overline{x}_j, \end{array} \right. \tag{1.6}$$

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

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

and

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

where $\underline{y}, \overline{y} \in \mathbb{R}^m$ and $\underline{z}, \overline{z} \in \mathbb{R}^n$ 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 $i = 1, \ldots, m$ and $j = 1, \ldots, n$. 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 briefly 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 other bounds in the problem, that is bounds on the dual variables z_j, bounds on the constraint values $[Ax]_i$ and bounds on the multipliers y_i. In the approach described in this paper, 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. In this case, one must be careful to monitor *all* new values of the a_{ij} in the updated row, as cancellation may imply that more than a single zero is created. This typically occurs when combining linearly dependent constraints.

3 Problem reduction

We now review the problem simplications techniques, in which the above transformations may be used. Although 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.

3.1 Bound compatibility

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

$$\underline{x}_j \leq \overline{x}_j \quad (j = 1, \ldots, n), \tag{3.1}$$

and

$$\underline{c}_i \leq \overline{c}_i \quad (i = 1, \ldots, m). \tag{3.2}$$

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,

$$\overline{x}_j = +\infty \;\Rightarrow\; z_j \geq 0 \;\Rightarrow\; \underline{z}_j \geq 0 \qquad \text{and} \qquad \underline{x}_j = -\infty \;\Rightarrow\; z_j \leq 0, \;\Rightarrow\; \overline{z}_j \leq 0,$$

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

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

and

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

and, conversely, that

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

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. In this case, either

$$\underline{c}_i \leq 0 \leq \overline{c}_i, \tag{3.3}$$

in which case the problem is compatible and the i-th linear constraint may be removed, or (3.3) 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

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

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. We also note that, when one of the tighter bounds is active at the solution, then the optimal value of y_i may then be simply computed from that of z_j as $y_i = z_j/a_{ij}$.

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

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

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

$$\frac{c_i - a_{ij} \underline{x}_j}{a_{ik}} \leq x_k \leq \frac{c_i - a_{ij} \overline{x}_j}{a_{ik}} \tag{3.5}$$

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

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

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

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

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

An 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

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

is transformed into

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

if $a_{ij} > 0$, and

$$a_{ij}x_j + \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k = c_i \qquad \text{and} \qquad c_i - a_{ij}\underline{x}_j \leq \sum_{\substack{k=1\\k\neq j}}^{n} a_{ik}x_k \leq c_i - a_{ij}\overline{x}_j \tag{3.7}$$

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.

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

$$[Ax]_i = \sum_{j=1}^{n} a_{ij}x_j \leq \sum_{\substack{j=1\\a_{ij}>0}} a_{ij}\overline{x}_j + \sum_{\substack{j=1\\a_{ij}<0}} a_{ij}\underline{x}_j \stackrel{\text{def}}{=} u_i, \tag{3.8}$$

and, similarly, that

$$[Ax]_i \geq \sum_{\substack{j=1\\a_{ij}>0}} a_{ij}\underline{x}_j + \sum_{\substack{j=1\\a_{ij}<0}} a_{ij}\overline{x}_j \stackrel{\text{def}}{=} \ell_i. \tag{3.9}$$

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

$$x_j = \overline{x}_j \;\text{ for all }\; j \mid a_{ij} > 0 \qquad \text{and} \qquad x_j = \underline{x}_j \;\text{ for all }\; j \mid a_{ij} < 0$$

if $u_i = \underline{c}_i$, or

$$x_j = \underline{x}_j \;\text{ for all }\; j \mid a_{ij} > 0 \qquad \text{and} \qquad x_j = \overline{x}_j \;\text{ for all }\; j \mid a_{ij} < 0$$

if $\ell_i = \overline{c}_i$. 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 dropped from the problem.

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 the i-th constraint with $a_{ik} > 0$. Then

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

which then implies that

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

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

$$\ell_i + a_{ik}(x_k - \overline{x}_k) \leq \sum_{j=1}^{n} a_{ij}x_j \leq \overline{c}_i \tag{3.11}$$

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, for $a_{ik} > 0$,

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

implies that

$$x_k \geq \overline{x}_k + \frac{\underline{c}_i - u_i}{a_{ik}},$$

while, for $a_{ik} < 0$,

$$\underline{c}_i \leq \sum_{j=1}^{n} a_{ij}x_j \leq u_i + a_{ik}(x_k - \underline{x}_k)$$

implies that

$$x_k \leq \underline{x}_k + \frac{\underline{c}_i - u_i}{a_{ik}}.$$

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 (3.9) 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 (3.8) or (3.9). 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

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

and therefore that

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

when $a_{ik} > 0$ and $\overline{x}_k = +\infty$, while the inequality

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

implies that

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

when $a_{ik} < 0$ and $\underline{x}_k = -\infty$.

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 (that is $a_{ij} \neq 0 \neq a_{kj}$) and where the i-th constraint is an equality constraint (i.e. $\underline{c}_i = \overline{c}_i$). It is then possible to perform the transformation

$$[\text{constraint } k]^+ = [\text{constraint } k] - \frac{a_{kj}}{a_{ij}} [\text{constraint } i], \tag{3.12}$$

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

$$\underline{y}_k + = \underline{y}_k, \qquad \overline{y}_k + = \overline{y}_k,$$

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}, and

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

otherwise.

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. This is one circumstance where the monitoring of cancellation is useful, 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, it is useful to limit the number of transformations of the type just described. In our implementation, this number is kept reasonable by limiting 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 when transforming the constraints, 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

$$y_i = \frac{g_j}{a_{ij}}. \tag{3.13}$$

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

$$x_j = \frac{1}{a_{ij}} \left(\overline{c}_i^+ - \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} a_{i\ell} x_\ell \right). \tag{3.14}$$

When a_{ij} is not too small[1], we then remove this variable from the problem, while remembering that its value can be computed from (3.14), 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 (3.13) and (3.14), we obtain that these updates are given by

$$f^+ = f + \overline{c}_i y_i \qquad \text{and} \qquad g_\ell^+ = g_\ell - a_{i\ell} y_i \quad (\ell \neq j, a_{i\ell} \neq 0). \tag{3.15}$$

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 makes the problem feasible, without affecting optimality. We may thus, for instance, choose (3.14) or

$$x_j = \frac{1}{a_{ij}} \left(\frac{\underline{c}_i + \overline{c}_i}{2} - \sum_{\substack{\ell=1 \\ \ell \neq j}}^{n} a_{i\ell} x_\ell \right), \tag{3.16}$$

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 (3.14) or (3.16).

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

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

and

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

3.3.2 Doubleton columns

Assume now that column j of A only contains two nonzeros, a_{ij} and a_{kj}, that variable j is linear (meaning that $h_{ij} = 0$ for $i = 1, \ldots, n$) and free or implied free, and that constraint i is an equality constraint. Then, as above, variable j may substituted from this constraint, yielding (3.14). However, the situation is now slightly more complicated in that not only the objective function and gradient must be updated using (3.15), but (3.14) must also be substituted into constraint k. This transformation may be expressed, provided a_{ij} is not too small, as

$$a_{k\ell}^+ = a_{k\ell} - \frac{a_{kj} a_{i\ell}}{a_{ij}} \tag{3.17}$$

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

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

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

$$g_j - a_{ij} y_i - a_{kj} y_k = z_j = 0 \tag{3.18}$$

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

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

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

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 (3.17) 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 to 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 (3.14) 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

$$\sum_{i=1}^{m} a_{ij}y_i - \sum_{\ell=1}^{n} h_{\ell j}x_\ell = g_j - z_j. \tag{3.19}$$

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

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

and

$$w_j \stackrel{\rm def}{=} \sum_{\substack{i=1\\ a_{ij}>0}}^{m} a_{ij}\overline{y}_i + \sum_{\substack{i=1\\ a_{ij}<0}}^{m} a_{ij}\underline{y}_i - \sum_{\substack{\ell=1\\ h_{\ell j}>0}}^{n} h_{\ell j}\underline{x}_\ell - \sum_{\substack{\ell=1\\ h_{\ell j}<0}}^{n} h_{\ell j}\overline{x}_\ell. \tag{3.21}$$

From these definitions, it is clear that

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

and therefore, from (3.19), that

$$v_j \leq g_j - z_j \leq w_j. \tag{3.22}$$

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

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

Let us consider first the case where $\overline{z}_j \leq 0$, 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 (3.22) that

$$z_j \geq g_j - w_j > 0, \tag{3.23}$$

which is inconsistent with our assumption that $\overline{z}_j \leq 0$. 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 (3.22) that

$$z_j \leq g_j - v_j < 0, \tag{3.24}$$

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 $\underline{z}_j \geq 0$ (for instance if $\overline{x}_j = +\infty$). Then, if $g_j < v_j$, we deduce from (3.22) that (3.24) holds, which is again incompatible with our assumption that $\underline{z}_j \geq 0$. The problem is also dual infeasible in this case, and our preprocessor exits with this information. If $g_j < v_j$, then it follows as above that (3.23) holds. The j-th dual constraint is therefore 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. More precisely, we need to fix

$$x_k = \underline{x}_k \text{ if } (v_j = g_j - \underline{z}_j \text{ and } h_{kj} < 0) \text{ or } (w_j = g_j - \overline{z}_j \text{ and } h_{kj} > 0),$$

or

$$x_k = \overline{x}_k \text{ if } (w_j = g_j - \underline{z}_j \text{ and } h_{kj} < 0) \text{ or } (v_j = g_j - \overline{z}_j \text{ and } h_{kj} > 0),$$

for $k = 1, \ldots, n$, as well as

$$y_i = \underline{y}_i \text{ if } (v_j = g_j - \underline{z}_j \text{ and } a_{kj} > 0) \text{ or } (w_j = g_j - \overline{z}_j \text{ and } a_{kj} < 0),$$

or

$$y_i = \overline{y}_i \text{ if } (w_j = g_j - \underline{z}_j \text{ and } a_{kj} > 0) \text{ or } (v_j = g_j - \overline{z}_j \text{ and } a_{kj} < 0),$$

for $i = 1, \ldots, m$.

3.3.4 Further use of the implied bounds on dual constraints

If (3.19) 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 (3.22) that, for $k = 1, \ldots, m$,

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

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

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

Similarly, we also obtain from the definition of w_j and (3.22) that, for $k = 1, \ldots, n$,

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

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

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

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

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

and

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

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 (3.20) or (3.21)). 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 in infinite. Assume first that it corresponds to the k-th multiplier (that is $\underline{y}_k = -\infty$ or $\overline{y}_k = +\infty$), and define

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

and

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

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

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

and that

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

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

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

and

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

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

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

and

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

and obtain that

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

and therefore that

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

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

$$x_k \leq -\frac{g_j - w_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} > 0 \text{ and } \underline{x}_k = -\infty, \tag{3.25}$$

and that

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

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

$$x_k \geq -\frac{g_j - v_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} > 0 \text{ and } \overline{x}_k = +\infty, \tag{3.27}$$

and

$$x_k \leq -\frac{g_j - v_j^{[x_k]}}{h_{kj}} \quad \text{if } h_{kj} < 0 \text{ and } \underline{x}_k = -\infty. \tag{3.28}$$

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

$$\alpha h_{\ell k} = h_{\ell j} \quad (\ell = 1, \ldots, n) \qquad \text{and} \qquad \alpha a_{ik} = a_{ij} \quad (i = 1, \ldots, m) \tag{3.29}$$

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 from another and by rejecting variables k within this list whose associated column is diagonal in H. Indeed, if $h_{kk} \neq 0$, then (3.29) 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 (3.29) 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 (3.29) as soon as a row index is found such that the second part of (3.29) 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 inspired by Tomlin and Welch (1983*a*), Andersen and Andersen (1995) and Gondzio (1997), with adaptations for the presence of the Hessian H. If (3.29) 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

$$a_{\ell k}x_k^+ = a_{\ell k}x_k + \alpha a_{\ell k}x_j = a_{\ell k}x_k + a_{\ell j}x_j \quad (\ell = 1, \ldots, m).$$

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 is always possible if variable j is free or if it is suitably unbounded. More formally, using (3.29), the symmetry of H and our assumption on g_k, we have that

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

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 (3.30) 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

$$\alpha z_k = \alpha g_k + \alpha\sum_{\ell=1}^{n} h_{k\ell}x_\ell - \alpha\sum_{i=1}^{m} a_{ik}y_i < \alpha g_j + \sum_{\ell=1}^{n} h_{j\ell}x_\ell - \sum a_{ij}y_i = \alpha z_j, \tag{3.31}$$

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 (3.30) and (3.31) 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.

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, pp?), 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). Similarly, it is advantageous to be able to access to a complete row (or column) 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 constrained 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 in

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

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

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

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

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 digonal 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 is 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 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 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. In none of the bounds in (3.4) or (3.5) 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

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

If, on the other hand, one of the bounds in (3.4) or (3.5) 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

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

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 (3.25)–(3.28). 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 in of (3.6) 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 $k = 1, \ldots, n$. It can then be verified that we may recover the value of z_j and y_i from the relations

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

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 (3.8)) 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 (3.9)) 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 (3.12) 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 (3.18) 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 (3.15) with (3.13) and (3.17). 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

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

(the last equality resulting from (3.18)) translate the optimality conditions of the transformed problem into optimality conditions for the untransformed 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 95 implementation of the ideas discussed above. The resulting package, `PRESOLVE` is an integral component of the GALAHAD optimization library (see Gould, Orban and Toint, 2002). 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.

7.1 Problem reduction

We start by considering the efficiency of the presolving process in terms of reduction of the problem sizes. Figures 7.1–7.4 report the effect of presolving on a collection of 178 linear and quadratic programs, of which 8 only involves only simple bounds, from the CUTE collection of test problems (see Bongartz, Conn, Gould and Toint, 1995) which includes the Netlib linear programs and 70 problems with a non-trivial quadratic term. Results were obtained on a Pentium (biprocessor) running Linux Red Hat 7.0 with 256 MBytes of memory, using the frt Fujitsu Fortran compiler without optimization.

Figure 7.1, 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 22% if one considers linear problems only, and of 16% 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.

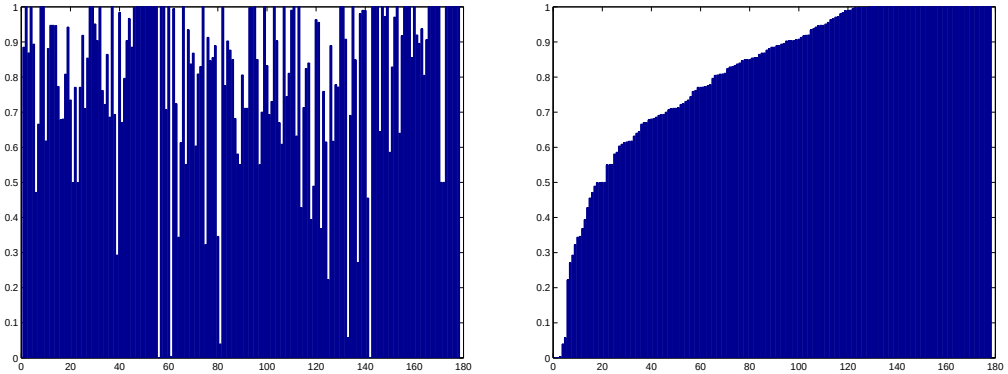

Figure 7.1: The reduction obtained by presolving on the number of problem variables, sorted by decreasing original problem size (left) and by decreasing reduction (right).

Figures 7.2–7.4 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 there is slightly more potential in reducing m (on average by 21%) than n, the amount of reduction being again uncorrelated to the original value of m. Figures 7.3 and 7.4 show that the reduction of the number of nonzero entries in A and H

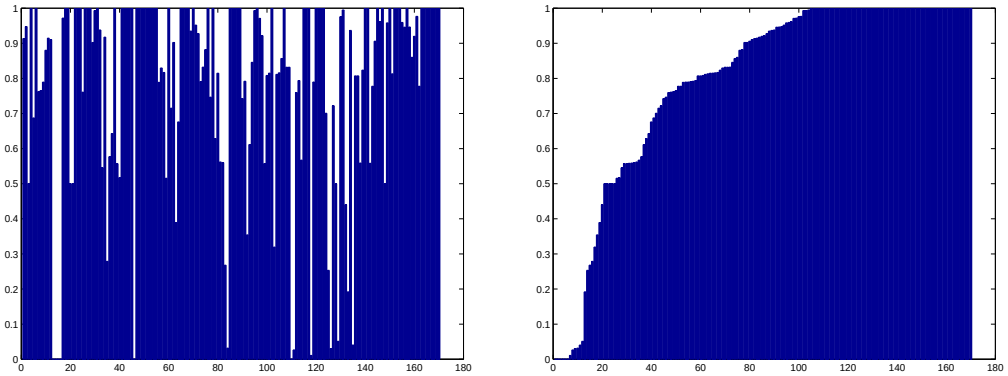

Figure 7.2: The reduction obtained by presolving on the number of constraints, sorted by decreasing number of constraints in the original problem (left) and by decreasing reduction (right).

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 (21%), which indicates again that the potential of presolving quadratic programming problems is comparable to that of linear programs, at least in terms of problem size reduction.

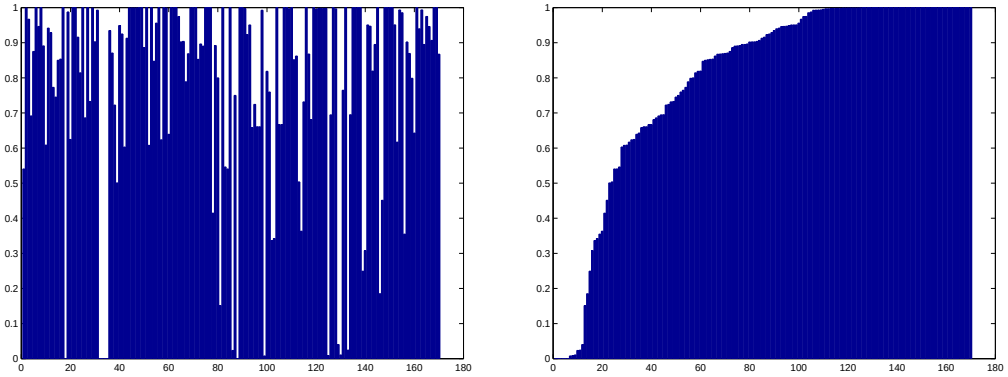

Figure 7.3: The reduction obtained by presolving on the number of nonzero entries in A, sorted by decreasing number of such entries in the original problem (left) and by decreasing reduction (right).

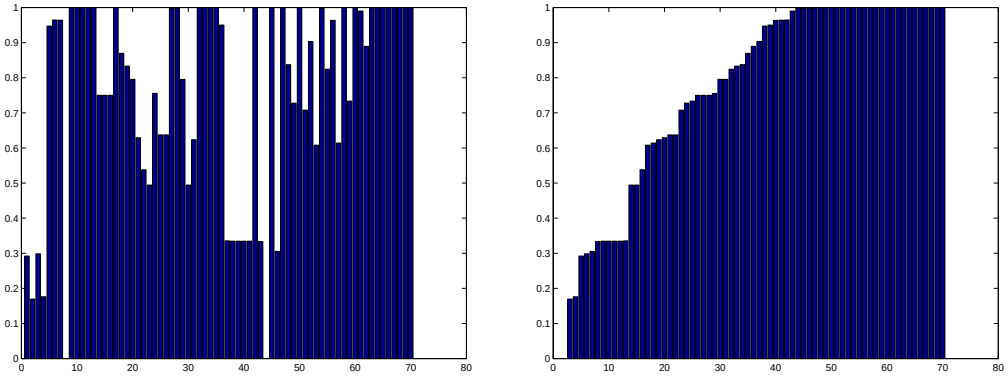

Figure 7.4: The reduction obtained by presolving on the number of nonzero entries in H, sorted by decreasing number of such entries in the original problem (left) and by decreasing reduction (right).

Tables A.1–A.5 (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 64 of the 93 comparable problems, smaller m for 50 and smaller number of nonzeros in A for 44, Gondzio's results showing smaller n for 29 problems, smaller m for 23 problems and smaller number of nonzeros in A for 43 problems, the rest being ties. A second point of interest is that comparison of the presolve times with numerical solution times for the tested 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`, `GMCASE2`, `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`, `STNPQ1`, `STNQP2` and `SOSQP1` in which the presolve removes all the variables and constraints, and thus reveals the complete solution to the QP under consideration.

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`, also from GALAHAD, an interior point algorithm for nonconvex quadratic programs described in Conn, Gould, Orban and Toint (2000*b*), Gould, Orban, Sartenaer and Toint (2001), or Gould and Toint (2001). Note that we excluded four problems from the comparison: `STATIC3`, which is unbounded below,[2], `DFL001` and `QAP12` which for which the memory of our computer was insufficient, and `FIT2D` for which the solution time exceeded the upper limit of 20000 seconds for all presolving modes.

We first report in Table 7.1 the causes of failure to solve the (possibly reduced) problem for all presolving modes (including no presolving at all).

Presolving mode	apparently infeasible	conditioning too large	total
none	1	2	3
tightest	1	1	2
medium	0	1	1
loosest	1	2	3

Table 7.1: Failure causes for problem solution.

Figure 7.5 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 171 problems where both solves were successful. As expected, the impact of presolving on solution times is variable. This impact is very favourable[3] in 34 of the 172 compared problems, favourable[4] for 60 problems, relatively neutral[5] for 42, unfavourable[6] for 26 and very unfavorable[7] for 9, with an average gain of 10% (11% for linear problems and 7% for quadratic ones).

Figures 7.6 and 7.7 show the same ratios for the medium and loosest presolving modes, respectively. For the medium mode, the average gain over the 172 comparable problems is 14% (16% for linear problems

[2] As successfully reported by the solver in all cases.
[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.

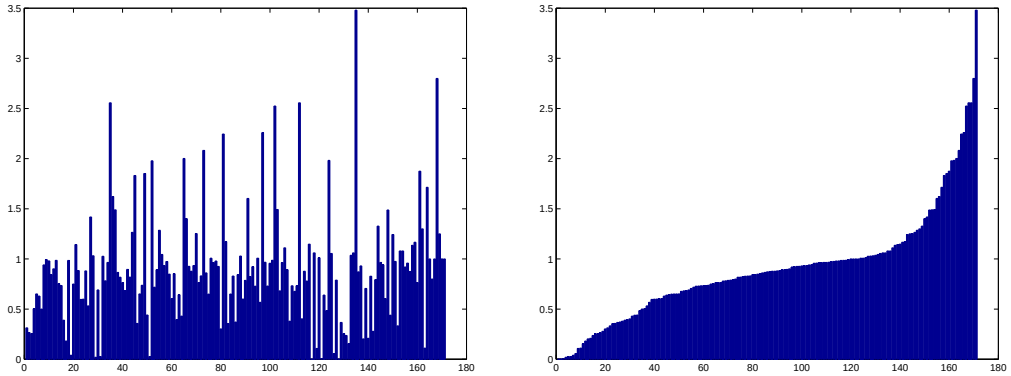

Figure 7.5: The reduction (or increase) of solution time with presolving in tightest mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

and 10% for quadratic ones), with 28 very favourable, 53 favourables, 61 neutral, 25 unfavourable and 5 very unfavourable cases. For the loosest mode, the average gain is 13% over 171 comparable problems (15% for linear problems and 10% for quadratic ones), with 29 very favourable, 54 favourable, 70 neutral, 13 unfavourable and 5 very unfavourable cases. The medium mode therefore appears as a potentially attractive middle ground between the two other strategies.

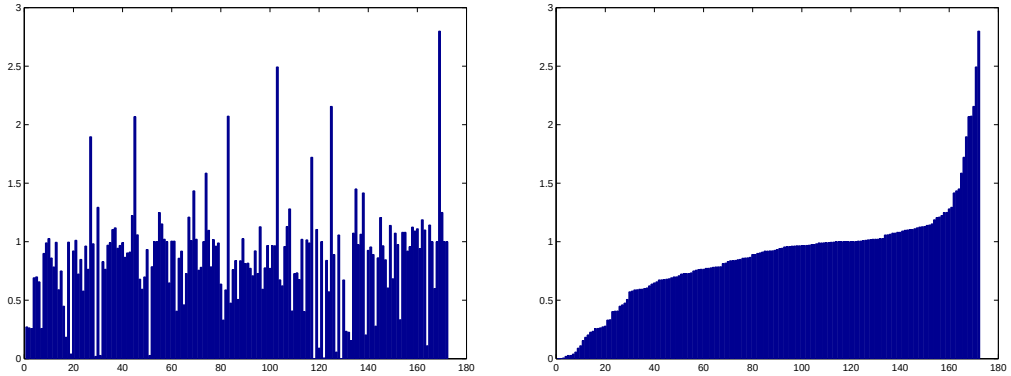

Figure 7.6: The reduction (or increase) of solution time with presolving in medium mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

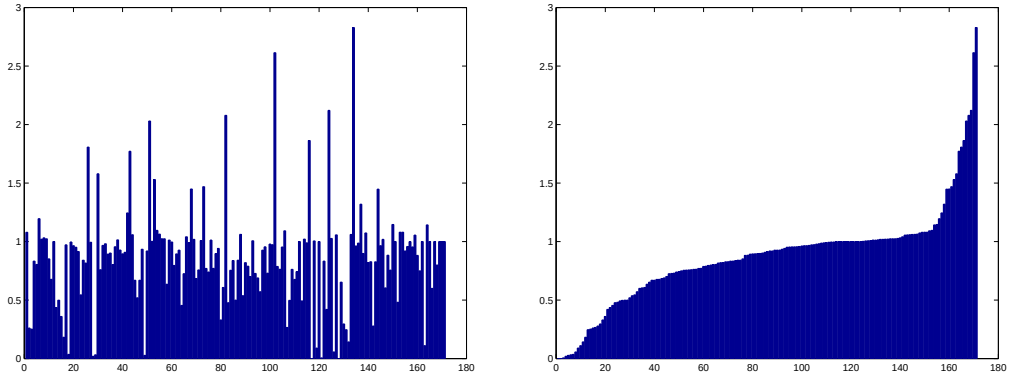

Figure 7.7: The reduction (or increase) of solution time with presolving in loosest mode, sorted by decreasing solution time for the original problem (left) and by decreasing reduction (right).

Tables B.1–B.5 (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, "time" that the CPU-time limit was reached before finding a solution, and "unb." that the problem is unbounded below.

Globally speaking, the effect of presolving on solution time therefore seems to be relatively positive, with a slightly larger gain for linear problems compared to quadratic ones.

7.3 Preprocessing in the context of SQP algorithms

An important class of algorithms that require the solution of quadratic programs are the "Sequential Quadratic Programming" (SQP) methods for nonlinear optimization. These methods attempt to solve general nonlinear minimization problems (that is involving possibly nonlinear and nonconvex objective function and constraints) by solving a sequence of quadratic programs whose first-order optimality conditions are identified with the Newton system arising from the first-order optimality conditions of the nonlinear program. We will not describe these methods in any detail here (we refer the interested reader to Gould and Toint, 2000, Nocedal and Wright, 1999 or Conn, Gould and Toint, 2000*a* and the references therein for further discussion), we simply notice that the quadratic programs that are successively solved (at each iteration of the SQP algorithm) may differ from one to the next in values of the coefficients of the objective function and constraints, since they correspond to linearizations of the original nonlinear problem at different points in the solution space. However, the structure of the problem is globally constant, and, more importantly, any linear constraints present in the original problem continue to appear in successive quadratic programs with the same coefficients (note however that this might not be the case for bound constraints on the variables, since these could be used in the definition of an iteration dependent trust region). If we wish to apply presolving techniques in this context, one may therefore hope to exploit the fact that linear constraints (and, possibly, the objective function if it is quadratic in our general nonlinear optimization problem) may be analyzed once and for all.

We have implemented this possibility in an early version of our code, but the advantage in solution time turned out in practice to be at best marginal and often negative, compared to the simpler application of the full set of presolving techniques to each QP under consideration. We have thus decided not to keep this option in our final code.

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 tohether 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 are almost as amenable to presolving as linear ones, both in terms of reduced problem size and reduced solution time. The resulting thread-safe Fortran 95 package `PRESOLVE` is freely available as part of the GALAHAD optimization library (see Gould et al., 2002).

Acknowledgements

The authors wish to thank J. Gondzio for his encouraging comments while this work was in progress and to P. Y. Bernard for proofreading an early version of the manuscript and helping to debug the code.

References

E. D. Andersen and K. D. Andersen. Presolving in linear-programming. *Mathematical Programming, Series A*, **71**(2), 221–245, 1995.

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

G. H. Bradley, G. G. Brown, and G. W. Graves. Structural redundancy in large-scale optimization models. *In* M. H. Karwan et al., ed., 'Redundancy in Mathematical Programming', pp. 145–169, Springer Verlag, Heidelberg, Berlin, New York, 1983.

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

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*. SIAM, Philadelphia, 2000*a*.

A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, **87**(2), 215–249, 2000*b*.

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

M. C. Ferris and T. S. Munson. Preprocessing complementarity problems. *In* M. C. Ferris, O. Mangasarian and J. S. Pang, eds, 'Complementarity: Applications, Algorithms and Extensions', Vol. 50, pp. 143–164, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

R. Fourer and D. M. Gay. 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', pp. 135–154, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.

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

N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. *In* M. J. D. Powell and S. Scholtes, eds, 'System Modelling and Optimization, Methods, Theory and Applications', pp. 149–178, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

N. I. M. Gould and Ph. L. Toint. An iterative working-set method for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-026, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. Technical Report in preparation, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002.

N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. *SIAM Journal on Optimization*, **11**(4), 974–1002, 2001.

J. Nocedal and S. J. Wright. *Large sparse numerical optimization*. Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, 1999.

J. A. Tomlin and J. S. Welch. Formal optimization of some reduced linear-programming problems. *Mathematical Programming*, **27**(2), 232–240, 1983*a*.

J. A. Tomlin and J. S. Welch. A pathological case in the reduction of linear programs. *Operations Research Letters*, **2**, 53–57, 1983*b*.

Appendix A: The effect of presolving on problem size

Problem	Before presolve				After presolve				time
	n	m	$\|A\|$	$\|H\|$	n	m	$\|A\|$	$\|H\|$	
25FV47	1571	821	10400	0	1468	741	9870	0	3.5
80BAU3B	9799	2262	21002	0	8759	1991	19238	0	14.5
ADLITTLE	97	56	383	0	89	53	360	0	0.2
AFIRO	32	27	83	0	29	21	72	0	0.1
AGG	163	488	2410	0	105	173	875	0	1.0
AGG2	302	516	4284	0	233	289	2315	0	1.7
AGG3	302	516	4300	0	235	290	2349	0	1.7
AUG2D	3280	1600	6400	3120	2964	1444	5776	2964	1.9
AUG2DQP	3280	1600	6400	3120	3120	1599	6238	3120	7.5
AUG2DC	3280	1600	6400	3280	3280	1600	6400	3280	1.2
AUG2DCQP	3280	1600	6400	3280	3280	1600	6400	3280	1.8
BANDM	472	305	2494	0	202	173	1256	0	0.9
BEACONFD	262	173	3375	0	15	7	26	0	0.7
BLEND	83	74	491	0	71	70	427	0	0.2
BLOCKQP1	2005	1001	9005	3005	2005	1001	8998	1005	6.3
BLOCKQP2	2005	1001	9005	3005	2005	1001	8998	1005	5.7
BLOCKQP3	2005	1001	9005	3005	2005	1001	8998	1005	6.2
BLOCKQP4	2005	1001	9005	3005	2005	1001	8998	1005	7.0
BLOCKQP5	2010	1001	14010	3010	2005	1001	13998	1010	7.0
BLOWEYA	2002	1002	5003	4003	2002	1002	5003	3002	3.2
BLOWEYB	2002	1002	5003	4003	2002	1002	5003	3002	3.1
BLOWEYC	2002	1002	5003	4003	2002	1002	5003	3002	3.2
BNL1	1175	643	5121	0	1073	535	4560	0	2.1
BNL2	3489	2324	13999	0	2980	1835	12636	0	10.0
BOEING1	384	351	3485	0	367	292	2303	0	1.3
BOEING2	143	166	1196	0	139	134	1133	0	0.4
BORE3D	315	233	1428	0	70	59	356	0	0.0
BQPGABIM	50	0	0	172	46	0	0	153	0.0
BQPGAUSS	2003	0	0	9298	2003	0	0	9298	1.0
BRANDY	249	220	2148	0	172	110	1492	0	0.7
CAPRI	353	271	1767	0	268	214	1351	0	0.8
CVXQP1	1000	500	1498	3984	700	500	1498	2484	1.1
CVXQP2	1000	250	749	3984	550	250	749	1969	0.8
CVXQP3	1000	750	2247	3984	850	750	2243	3169	1.2
CYCLE	2857	1903	20720	0	2063	1448	16873	0	8.0

Table A.1: The effect of presolving on the problem dimensions (1)

Problem	Before presolve				After presolve				time
	n	m	$\|A\|$	$\|H\|$	n	m	$\|A\|$	$\|H\|$	
CZPROB	3523	929	10669	0	2502	479	5343	0	9.2
D2Q06C	5167	2171	32417	0	4177	1986	30108	0	14.1
D6CUBE	6184	415	37704	0	5447	403	33601	0	22.6
DEGEN2	534	444	3978	0	529	441	3968	0	1.3
DEGEN3	1818	1503	24646	0	1808	1493	24343	0	8.5
DFL001	12230	6071	35632	0	10626	5754	33559	0	28.5
DTOC3	2999	1998	6993	2997	2996	1997	6986	2996	1.9
DUAL1	85	1	85	3558	85	1	85	3558	0.5
DUAL2	96	1	96	4508	96	1	96	4508	0.6
DUALC1	9	215	1935	45	9	11	78	45	0.7
DUALC2	7	229	1603	28	7	7	39	28	0.4
DUALC5	8	278	2224	36	8	3	20	36	0.5
DUALC8	8	503	4024	36	8	16	94	36	0.9
E226	282	223	2578	0	256	161	2223	0	0.6
ETAMACRO	688	400	2409	0	461	326	1762	0	0.8
FFFFF800	854	524	6227	0	624	427	4912	0	2.1
FINNIS	614	497	2310	0	457	369	1573	0	1.0
FIT1D	1026	24	13404	0	1025	24	13308	0	4.6
FIT1P	1677	627	9968	0	1028	627	9219	0	6.1
FIT2D	10500	25	129018	0	10485	25	128882	0	142.2
FIT2P	13525	3000	50284	0	13525	3000	50284	0	81.5
FORPLAN	421	161	4563	0	206	90	1892	0	1.4
GANGES	1681	1309	6912	0	578	714	4418	0	7.1
GFRD-PNC	1092	616	2377	0	934	460	2063	0	1.0
GMNCASE1	175	300	23940	11803	175	300	23940	11802	2.3
GMNCASE2	175	1050	28546	11803	175	543	24361	11803	10.6
GMNCASE3	175	1050	28546	11803	175	585	24266	11803	10.6
GMNCASE4	175	350	27510	15330	0	0	0	0	2.5
GOULDQP2	1999	999	2997	1997	1999	999	2997	1997	1.2
GOULDQP3	1999	999	2997	3995	1999	999	2997	3995	1.3
GREENBEA	5405	2392	30877	0	3678	1830	23011	0	11.8
GREENBEB	5405	2392	30877	0	3665	1824	23864	0	12.5
GROW15	645	300	5620	0	645	300	5620	0	0.9
GROW22	946	440	8252	0	946	440	8252	0	1.3
GROW7	301	140	2612	0	301	140	2612	0	0.4
HUESTIS	1000	2	2000	1000	1000	2	1994	1000	0.8
ISRAEL	142	174	2269	0	142	163	2258	0	0.5
JNLBRNG1	5625	0	0	16725	5329	0	0	15841	3.4
KB2	41	43	286	0	33	42	256	0	0.1

Table A.2: The effect of presolving on the problem dimensions (2)

Problem	Before presolve				After presolve				time
	n	m	$\|A\|$	$\|H\|$	n	m	$\|A\|$	$\|H\|$	
KSIP	20	1001	20001	20	20	1000	20000	20	3.9
LISWET8	103	100	400	103	103	100	400	103	0.1
LOTFI	308	153	1078	0	274	126	965	0	0.5
MAROS	1443	846	9614	0	871	605	5797	0	3.3
MAROS-R7	9408	3136	144848	0	4435	2156	78295	0	77.0
MODEL	1831	339	1893	0	6	9	20	0	0.4
MODSZK1	1620	687	3168	0	893	654	2407	0	4.0
MOSARQP1	2500	700	3422	2545	732	700	3397	777	4.8
MOSARQP2	900	600	2930	945	624	600	2921	669	0.8
NCVXBQP1	1000	0	0	3984	998	0	0	3976	0.5
NCVXBQP2	1000	0	0	3984	998	0	0	3976	0.5
NCVXQP1	1000	500	1498	3984	711	500	1498	2539	1.2
NCVXQP2	1000	500	1498	3984	711	500	1498	2539	1.3
NCVXQP3	1000	500	1498	3984	806	500	1498	3010	1.9
NCVXQP4	1000	250	749	3984	550	250	749	1969	1.6
NCVXQP5	1000	250	749	3984	581	250	749	2144	1.3
NCVXQP6	1000	250	749	3984	682	250	749	2507	1.6
NCVXQP7	1000	750	2247	3984	850	750	2243	3169	1.3
NCVXQP8	1000	750	2247	3984	877	750	2243	3319	1.6
NCVXQP9	1000	750	2247	3984	903	750	2243	3464	1.7
NESM	2923	662	13288	0	2227	614	12421	0	7.7
NOBNDTOR	100	0	0	240	64	0	0	176	0.0
OBSTCLAE	5625	0	0	16425	5329	0	0	15841	2.8
OET1	3	1002	3004	0	3	1002	3004	0	1.3
PEROLD	1376	635	6018	0	1113	560	5228	0	1.9
PILOT	3652	1451	43167	0	3356	1361	40855	0	18.9
PILOT4	1000	410	5141	0	775	378	4609	0	1.4
PILOT87	4883	2030	73152	0	4602	1973	70750	0	23.5
PILOT-JA	1988	940	14698	0	1405	768	10768	0	4.2
PILOT-WE	2789	722	9126	0	2411	675	8332	0	6.2
PILOTNOV	2172	975	13057	0	1730	809	11369	0	7.1
PRIMAL1	325	85	5815	324	200	85	5815	199	0.5
PRIMAL2	649	96	8042	648	395	96	8042	394	0.8
PRIMAL3	745	111	21547	744	673	111	21547	672	1.8
PRIMAL4	1489	75	16031	1488	1247	75	16031	1246	1.7
PRIMALC1	230	9	2070	229	230	9	2070	229	0.4
PRIMALC5	287	8	2296	286	287	8	2296	286	0.4
PRIMALC8	520	8	4160	520	520	8	4160	519	0.7
PT	2	501	1002	0	2	501	1002	0	0.2

Table A.3: The effect of presolving on the problem dimensions (3)

Problem	Before presolve				After presolve				time
	n	m	$\|A\|$	$\|H\|$	n	m	$\|A\|$	$\|H\|$	
QAP12	8856	3192	38304	0	8856	3192	38304	0	10.3
QAP8	1632	912	7296	0	1632	912	7296	0	1.8
QPCBLEND	83	74	491	83	83	71	443	83	0.1
QPCBOEI1	384	351	3485	384	370	292	2303	370	1.4
QPCBOEI2	143	166	1196	143	143	134	1137	143	0.3
QPCSTAIR	467	356	3856	467	385	356	3666	385	0.5
READING2	303	200	800	0	187	88	361	0	0.4
RECIPELP	180	91	663	0	82	74	409	0	0.2
SC105	103	105	280	0	100	101	273	0	0.1
SC205	203	205	551	0	199	200	543	0	0.3
SC50A	48	50	130	0	45	46	123	0	0.1
SC50B	48	50	118	0	43	43	107	0	0.1
SCAGR25	500	471	1554	0	316	288	1080	0	0.9
SCAGR7	140	129	420	0	82	72	270	0	0.2
SCFMX1	457	330	2589	0	384	262	2205	0	1.0
SCFMX2	914	660	5183	0	761	522	4420	0	2.8
SCFMX3	1371	990	7777	0	1138	781	6589	0	4.2
SCORPION	358	388	1426	0	132	124	439	0	0.6
SCRS8	1169	490	3182	0	989	388	2605	0	2.3
SCSD1	760	77	2388	0	760	77	2388	0	0.5
SCSD6	1350	147	4316	0	1350	147	4316	0	0.9
SCSD8	2750	397	8584	0	2750	397	8584	0	1.6
SCTAP1	480	300	1692	0	480	300	1692	0	0.4
SCTAP2	1880	1090	6714	0	1880	1090	6714	0	1.4
SCTAP3	2480	1480	8874	0	2440	1480	8874	0	2.0
SEBA	1028	515	4352	0	41	138	658	0	2.4
SHARE1B	225	117	1151	0	191	106	943	0	0.4
SHARE2B	79	96	694	0	79	92	660	0	0.2
SHELL	1775	536	3556	0	1285	337	2576	0	2.7
SHIP04L	2118	402	6332	0	1915	325	5723	0	2.1
SHIP04S	1458	402	4352	0	1266	224	3481	0	1.7
SHIP08L	4283	778	12802	0	3147	526	9249	0	4.8
SHIP08S	2387	778	7114	0	1601	303	4436	0	2.4
SHIP12L	5427	1151	16170	0	4196	664	11092	0	7.5
SHIP12S	2763	1151	8178	0	1895	321	4969	0	3.0
SIERRA	2036	1227	7302	0	1967	1126	6980	0	4.0
SIPOW1	2	2000	4000	0	2	2000	4000	0	0.8
SIPOW1M	2	2000	4000	0	2	2000	4000	0	0.7

Table A.4: The effect of presolving on the problem dimensions (4)

Problem	Before presolve				After presolve				time
	n	m	$\|A\|$	$\|H\|$	n	m	$\|A\|$	$\|H\|$	
SIPOW2	2	2000	3000	0	2	1000	2000	0	0.8
SIPOW2M	2	2000	3000	0	2	1000	2000	0	0.8
SIPOW3	4	2000	5992	0	2	1998	5990	0	1.3
SIPOW4	4	2000	7000	0	2	2000	7000	0	0.7
SOSQP1	2000	1001	4000	3000	0	0	0	0	1.0
SOSQP2	2000	1001	4000	3000	2000	1001	3000	1000	7.0
SSEBLIN	194	72	312	0	192	72	310	0	0.1
STAIR	467	356	3856	0	333	305	3560	0	1.1
STANDATA	1075	359	3031	0	372	293	1037	0	1.6
STANDGUB	1184	361	3139	0	382	293	1057	0	1.6
STANDMPS	1075	467	3679	0	956	395	2421	0	2.0
STATIC3	434	96	496	1014	171	48	176	738	0.4
STCQP1	4097	2052	13338	26603	3158	0	0	4683	2.7
STCQP2	4097	2052	13338	26603	2045	0	0	7943	2.9
STNQP1	4097	2052	13338	26603	3158	0	0	4512	2.6
STNQP2	4097	2052	13338	26603	2045	0	0	7770	3.3
STOCFOR1	111	117	447	0	92	91	357	0	0.2
STOCFOR2	2031	2157	8343	0	1798	1964	7392	0	4.4
STOCFOR3	15695	16675	64875	0	13892	15236	56774	0	48.7
TORSION1	5476	0	0	15984	5184	0	0	15408	2.5
TRUSS	8806	1000	27836	0	8806	1000	27836	0	6.7
TUFF	587	333	4520	0	476	253	4032	0	2.2
UBH1	9009	6000	24000	3003	6000	3003	14991	3003	74.4
VTP-BASE	203	198	908	0	55	38	168	0	0.3
WOOD1P	2595	244	70215	0	1800	171	48552	0	25.2
WOODW	8405	1098	37474	0	5194	706	22800	0	16.8
YAO	202	200	600	202	200	199	596	200	0.3

Table A.5: The effect of presolving on the problem dimensions (5)

Appendix B: The effect of presolving on solution time

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
25FV47	249.5		285.2		252.6		240.9	
80BAU3B	827.1		778.0		744.4		844.3	
ADLITTLE	1.6		3.0		1.9		1.2	
AFIRO	0.2		0.2		0.2		0.2	
AGG	5.8		7.7		7.0		8.4	
AGG2	21.1	cond.	8.0		8.6		10.5	
AGG3	18.86		7.6		7.6		9.3	
AUG2D	24.1		17.6		18.6		17.6	
AUG2DQP	58.6		107.9	inf.	53.9		54.3	
AUG2DC	24.4		23.6		23.6		23.3	
AUG2DCQP	62.4		-	cond.	62.7		63.2	
BANDM	18.2		19.3		20.1		18.3	
BEACONFD	16.3		0.1		0.1		0.1	
BLEND	2.4		2.1		2.7		2.4	
BLOCKQP1	19.5		14.3		13.2		14.5	
BLOCKQP2	30.0		23.6		24.5		24.6	
BLOCKQP3	19.5		13.2		14.3		13.2	
BLOCKQP4	32.0		27.0		26.8		26.9	
BLOCKQP5	27.6		20.1		20.1		20.1	
BLOWEYA	96.7		34.5		102.4		102.4	
BLOWEYB	48.3		42.4		69.3		70.0	
BLOWEYC	88.4		39.0		82.5		82.6	
BNL1	50.9		71.3		61.6		53.0	
BNL2	228.2		136.1		193.7		208.8	
BOEING1	37.8		35.0		24.1		35.6	
BOEING2	11.0		11.6		9.8		11.3	
BORE3D	9.8		2.5		2.3		2.9	
BQPGABIM	0.5		1.4		1.4		0.4	
BQPGAUSS	39.9		40.2		40.6		40.4	
BRANDY	21.4		19.1		27.4		5.7	
CAPRI	11.5		22.8		24.8		24.4	
CVXQP1	65.3		39.4		42.3		41.5	
CVXQP2	30.5		18.3		24.8		16.4	
CVXQP3	105.0		86.0		95.7		95.4	
CYCLE	201.3		285.6		381.8		363.9	

Table B.1: The effect of presolving on the problem solution time (1)

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
CZPROB	97.4		178.4		201.5		172.6	
D2Q06C	1972.5		1284.6		1379.6		1587.6	
D6CUBE	626.5		564.9		491.2		424.5	
DEGEN2	19.4		49.6		19.8		19.4	
DEGEN3	130.1		332.7		129.0		127.7	
DFL001	-	mem.	-	mem.	-	mem.	-	mem.
DTOC3	647.2		545.5		556.9		551.7	
DUAL1	6.4		5.3		5.7		5.3	
DUAL2	5.8		4.6		5.0		4.8	
DUALC1	9.7		2.3		2.2		2.4	
DUALC2	8.4		1.3		1.3		1.2	
DUALC5	10.6		0.6		0.6		0.6	
DUALC8	17.7		1.9		1.6		1.6	
E226	41.0		35.3		45.0		31.6	
ETAMACRO	27.4		27.6		30.9		18.9	
FFFFF800	59.8		23.6		24.3		47.6	
FINNIS	19.8		14.5		14.4		15.1	
FIT1D	560.2		552.5		557.4		560.1	
FIT1P	22.3		56.3		55.6		58.3	
FIT2D	-	time	-	time	-	time	-	time
FIT2P	345.2		340.2		343.9		335.6	
FORPLAN	35.4		10.7		11.7		11.7	
GANGES	56.3		24.2		25.9		25.6	
GFRD-PNC	40.0		26.0		31.4		29.6	
GMNCASE1	105.6		94.6		95.5		94.6	
GMNCASE2	141.8		110.9		108.4		107.8	
GMNCASE3	226.3		135.6		130.8		123.2	
GMNCASE4	18.2		0.0		0.0		0.0	
GOULDQP2	2.5		2.3		2.3		2.3	
GOULDQP3	2.4		2.3		2.3		2.3	
GREENBEA	838.0	cond.	419.6		216.7		-	cond.
GREENBEB	399.9		294.0		300.0		199.6	
GROW15	42.0		34.9		42.0		42.4	
GROW22	61.2		52.2		61.5		61.0	
GROW7	18.8		16.5		19.1		19.2	
HUESTIS	-	inf.	32.1		26.8		-	inf.
ISRAEL	108.6		74.3		94.1		100.7	
JNLBRNG1	746.3		740.8		739.0		770.0	
KB2	1.8		2.1		2.0		1.9	

Table B.2: The effect of presolving on the problem solution time (2)

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
KSIP	34.2		40.1		70.9		71.1	
LISWET8	10.4		8.2		11.0		11.0	
LOTFI	9.8		3.6		6.6		6.4	
MAROS	392.2		152.6		176.1		141.2	
MAROS-R7	3158.0		804.5		813.6		795.3	
MODEL	0.9		0.1		0.1		0.1	
MODSZK1	21.6		24.0		24.4		23.6	
MOSARQP1	42.5		32.5		33.2		32.2	
MOSARQP2	32.8		27.2		27.5		27.5	
NCVXBQP1	8.2		8.7	stp.	11.9		23.2	
NCVXBQP2	84.2		60.5		66.2		171.0	
NCVXQP1	89.6		66.1		53.2		46.6	
NCVXQP2	41.3		86.0		65.5		60.7	
NCVXQP3	145.8		149.7		120.9		230.3	
NCVXQP4	26.6		15.1		15.8		15.2	
NCVXQP5	33.5		21.8		25.5		25.3	
NCVXQP6	74.1		95.3		74.1		113.4	
NCVXQP7	129.4		209.9		142.8		115.5	
NCVXQP8	125.3		186.7		140.2		113.0	
NCVXQP9	229.2		202.7		165.8		218.2	
NESM	258.4		193.8		238.0		257.4	
NOBNDTOR	0.6		0.6		0.6		0.6	
OBSTCLAE	668.1		655.1		686.2		684.4	
OET1	4.1		5.1		4.4		4.7	
PEROLD	70.5		65.8		81.2		75.0	
PILOT	951.1		598.8		624.7		1137.3	
PILOT4	30.8		31.7		31.6		32.7	
PILOT87	8264.0		2585.0		2247.3		8921.5	
PILOT-JA	172.2		119.0		222.8		-	cond.
PILOT-WE	84.9		168.0		-	cond.	78.1	
PILOTNOV	73.1		76.4		91.3		80.1	
PRIMAL1	65.9		55.9		65.9		67.5	
PRIMAL2	66.2		64.5		67.6		67.8	
PRIMAL3	194.0		200.2		190.6		192.9	
PRIMAL4	116.0		94.9		112.3		110.9	
PRIMALC1	23.0		22.7		22.2		22.4	
PRIMALC5	23.4		22.4		22.6		22.9	
PRIMALC8	363.5		66.0		66.6		66.4	
PT	0.5		0.5		0.5		0.5	

Table B.3: The effect of presolving on the problem solution time (3)

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	c
QAP12	-	mem.	-	mem.	-	mem.	-	mem.
QAP8	48.8		45.2		49.3		48.4	
QPCBLEND	5.2		4.9		4.4		5.3	
QPCBOEI1	92.1		60.0		62.6		61.7	
QPCBOEI2	29.5		27.2		27.2		29.7	
QPCSTAIR	104.1		131.7		127.3		129.6	
READING2	0.3		0.3		0.3		0.3	
RECIPELP	4.8		2.9		2.9		2.9	
SC105	1.0		1.3		1.1		1.0	
SC205	2.2		2.5		2.4		2.1	
SC50A	0.4		0.5		0.5		0.4	
SC50B	0.5		0.4		0.3		0.3	
SCAGR25	7.8		6.8		8.3		7.7	
SCAGR7	1.7		1.3		1.6		1.5	
SCFMX1	38.1		37.3		37.7		34.2	
SCFMX2	123.1		106.6		116.2		99.0	
SCFMX3	218.6		192.5		210.5		183.8	
SCORPION	6.1		1.7		1.7		1.7	
SCRS8	29.5		24.3		20.9		20.7	
SCSD1	4.0		3.9		3.9		4.0	
SCSD6	8.2		8.5		8.8		8.7	
SCSD8	17.5		17.7		17.5		17.5	
SCTAP1	18.6		14.5		18.4		18.4	
SCTAP2	81.1		72.4		81.2		81.3	
SCTAP3	115.2		88.0		114.2		116.9	
SEBA	12.6		6.1		7.2		5.3	
SHARE1B	15.0		9.6		12.6		12.5	
SHARE2B	4.3		6.4		4.9		3.8	
SHELL	22.0		15.0		13.7		16.8	
SHIP04L	25.9		58.5		20.1		24.0	
SHIP04S	22.1		33.0		14.9		17.4	
SHIP08L	52.9		105.9		38.5		38.4	
SHIP08S	29.5		47.3		22.8		23.3	
SHIP12L	89.3		165.4		62.4		59.8	
SHIP12S	43.1		54.0		32.6		29.5	
SIERRA	46.9		43.8		47.9		47.8	
SIPOW1	2.5		2.7		2.7		2.7	
SIPOW1M	2.5		2.7		2.7		2.7	

Table B.4: The effect of presolving on the problem solution time (4)

Problem	No presolve		Presolve (t)		Presolve (m)		Presolve (l)	
	time	c	time	c	time	c	time	
SIPOW2	6.9		1.4		1.4		6.2	
SIPOW2M	6.7		1.4		6.4		5.5	
SIPOW3	6.8		4.8		6.3		7.3	
SIPOW4	5.7		5.5		5.5		5.5	
SOSQP1	10.0		0.0		0.0		0.0	
SOSQP2	58.8		37.8		50.5		52.6	
SSEBLIN	0.7		1.2		0.8		0.8	
STAIR	7.9		27.5		7.7		7.6	
STANDATA	32.6		12.1		16.5		16.3	
STANDGUB	34.1		12.1		16.2		16.3	
STANDMPS	39.3		37.9		37.8		30.3	
STATIC3	-	unb.	-	unb.	-	unb.	-	unb.
STCQP1	178.3		3.6		3.6		3.6	
STCQP2	284.3		11.2		11.3		10.2	
STNQP1	87.6		2.4		2.4		2.4	
STNQP2	169.6		4.7		4.7		5.6	
STOCFOR1	4.1		1.8		2.8		3.1	
STOCFOR2	211.8		112.7		161.8		173.0	
STOCFOR3	2728.4		1375.6		1888.3		2272.4	
TORSION1	21.8		21.0		20.9		20.8	
TRUSS	135.4		130.7		131.4		131.0	
TUFF	18.3		21.0		31.5		34.1	
UBH1	34.2		76.8		20.1		20.8	
VTP-BASE	2.7		0.9		0.9		1.3	
WOOD1P	496.3		373.8		292.4		216.4	
WOODW	3568.7		944.9		939.4		932.7	
YAO	7.2		6.7		10.2		9.5	

Table B.5: The effect of presolving on the problem solution time (5)