Raport Badawczy
Research Report

RB/20/2010

Bracketing methods for the continuous quadratic knapsack problem

K.C. Kiwiel

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences

POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ul. Newelska 6

01-447 Warszawa

tel.: (+48) (22) 3810100

fax: (+48) (22) 3810105

Kierownik Pracowni zgłaszający pracę:
Prof. dr hab. inż. Krzysztof C. Kiwiel

Warszawa 2010

Bracketing methods for the continuous quadratic knapsack problem

Krzysztof C. Kiwiel*

December 10, 2010

Abstract

We show that breakpoint searching and variable fixing methods for the continuous quadratic knapsack problem fit a common bracketing framework. This allows us to develop several more efficient versions. Extensive computational results are given.

Key words. Nonlinear programming, convex programming, quadratic programming, separable programming, singly constrained quadratic program.

1 Introduction

The *continuous quadratic knapsack problem* is defined by

$$\mathrm{P}: \quad \min \quad f(x) := \tfrac{1}{2}x^T Dx - a^T x \quad \text{s.t.} \quad b^T x = r, \quad l \le x \le u, \tag{1.1}$$

where x is an n-vector of variables, $a, b, l, u \in \mathbb{R}^n$, $r \in \mathbb{R}$, $D = \mathrm{diag}(d)$ with $d > 0$, so that the objective f is strictly convex. Assuming P is feasible, let x^* denote its unique solution.

Problem P has applications in resource allocation [BiH81, BrS97, HoH95], hierarchical production planning [BiH81], network flows [Ven91], transportation problems [CoH94], multicommodity network flows [HKL80, NiZ92, ShM90], constrained matrix problems [CDZ86], integer quadratic knapsack problems [BSS95, BSS96], integer and continuous quadratic optimization over submodular constraints [HoH95], Lagrangian relaxation via subgradient optimization [HWC74], and quasi-Newton updates with bounds [CaM87].

Specialized algorithms for P employ either breakpoint searching or variable fixing. Breakpoint searching methods solve the dual of P by finding a Lagrange multiplier t_* that solves the equation $g(t) = r$, where g is a monotone piecewise linear function with $2n$ breakpoints (cf. §2). The earliest $O(n \log n)$ methods [HWC74, HKL80] sort the breakpoints initially, whereas the $O(n)$ algorithms [Bru84, CaM87, MdP89, PaK90, CoH94, HoH95, MMP97, MSMJ03] use medians of breakpoint subsets. A general framework for such methods was recently given in [Kiw08a], together with various modifications.

*Systems Research Institute, Newelska 6, 01-447 Warsaw, Poland (kiwiel@ibspan.waw.pl)

The variable fixing methods of [BiH81, Ven91, RJL92, BSS96] determine at each iteration the optimal value of at least one variable; such variables are fixed and hence effectively removed for the next iteration. Our recent paper [Kiw08b] clarified some convergence issues of these methods and gave more efficient versions. Although these methods have worst-case performance of $O(n^2)$, they may be competitive in practice [Ven91, RJL92], since they don't need sorting or median calculations.

This paper shows that both classes of methods fit a unified bracketing framework, in which the optimal multiplier is localized by a shrinking interval $[t_L, t_U]$ that is updated by evaluating g at a trial multiplier $\hat{t} \in (t_L, t_U)$. Specific methods differ in two aspects: (1) the updates of auxiliary quantities used for evaluating $g(\hat{t})$, and (2) the selection of the next multiplier $\hat{t}$. As for the first aspect, extending the earlier work of [Bru84, CaM87, Kiw08a], we identify three updates which seem to be most efficient with respect to the numbers of arithmetic operations and comparisons, as well as their simplified variants that access memory in more efficient ways at the price of making more comparisons. Concerning the second aspect, we discuss several efficient multiplier selections in addition to those given in [Kiw08a]. First, we may choose $\hat{t}$ as the median of a certain superset of the set T of breakpoints in (t_L, t_U) (which may be easier to handle than T itself; cf. §4.1), or as the median of a small randomly chosen subset of T, or simply as the average of T (this choice is quite competitive; cf. §7). Second, we may select $\hat{t}$ as in the variable fixing methods; this leads to breakpoint versions of variable fixing methods that seem to be most efficient in practice (cf. §7). We also give an initial problem transformation which reduces work per iteration. Last but not least, we present extensive numerical comparisons of more than forty versions of our methods on large-scale problems.

The paper is organized as follows. In §2 we review some properties of P and introduce a general bracketing method. Efficient g-evaluations are described in §3. Multiplier selections inspired by breakpoint searching and variable fixing are discussed in §§4 and 5, respectively. Initial problem transformations are the subject of §6. Finally, our computational results are reported in §7.

2 The bracketing method

Viewing $t \in \mathbb{R}$ as a multiplier for the equality constraint of P in (1.1), consider the *Lagrangian primal solution* (the minimizer of $f(x) + t(b^T x - r)$ s.t. $l \le x \le u$)

$$x(t) := \min\{\, \max[\, l, D^{-1}(a - tb)\,], u\,\} \tag{2.1}$$

(where the min and max are taken componentwise) and its *constraint value*

$$g(t) := b^T x(t). \tag{2.2}$$

Solving P amounts to solving $g(t) = r$ for a multiplier lying in the *optimal dual set*

$$T_* := \{\, t : g(t) = r \,\}. \tag{2.3}$$

Indeed, invoking the Karush–Kuhn–Tucker conditions for P as in [CaM87, Thm. 2.1], [HKL80, §2], [NiZ92, §1.2], [PaK90, Thm. 2.1] gives the following result.

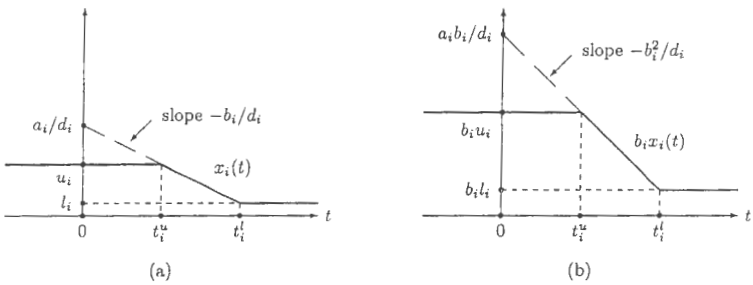

Figure 2.1: (a) Illustration of $x_i(t) := \min\{\max[l_i, (a_i - tb_i)/d_i], u_i\}$. (b) Illustration of $b_ix_i(t) = \min\{\max[b_il_i, (a_ib_i - tb_i^2)/d_i], b_iu_i\}$.

Fact 2.1. *$x^* = x(t)$ iff $t \in T_*$. Further, the set T_* is nonempty.*

As in [Bru84], we assume for simplicity that $b > 0$, because if $b_i = 0$, x_i may be eliminated ($x_i^* = \min\{\max[l_i, a_i/d_i], u_i\}$), whereas if $b_i < 0$, we may replace $\{x_i, a_i, b_i, l_i, u_i\}$ by $-\{x_i, a_i, b_i, u_i, l_i\}$ (in fact, this transformation may be *implicit*).

By (2.1)-(2.2), the function g has the following *breakpoints*

$$t_i^l := (a_i - l_id_i)/b_i \quad \text{and} \quad t_i^u := (a_i - u_id_i)/b_i, \quad i = 1{:}n, \tag{2.4}$$

with $t_i^u \le t_i^l$ (from $l_i \le u_i$ and $b_i > 0$), and each $x_i(t)$ may be expressed as

$$x_i(t) = \begin{cases} u_i & \text{if } t \le t_i^u, \\ (a_i - tb_i)/d_i & \text{if } t_i^u \le t \le t_i^l, \\ l_i & \text{if } t_i^l \le t. \end{cases} \tag{2.5}$$

Thus $g(t)$ is a continuous, piecewise linear and *nonincreasing* function of t (cf. Fig. 2.1). Let $T_0 := \{t_i^l\}_{i\in N} \cup \{t_i^u\}_{i\in N}$ denote the (multi)set of breakpoints, where $N := \{1{:}n\}$.

To locate an optimal t_* in T_*, the algorithm below generates a *bracketing interval* $[t_L, t_U]$ that contains T_* by evaluating g at trial multipliers $\hat{t}$ in $[t_L, t_U]$ until $T_0 \cap (t_L, t_U)$ becomes empty; then g is linear on $[t_L, t_U]$, and t_* is found by interpolation.

Algorithm 2.2.

Step 0 (*Initiation*). Set $T_0 := \{t_i^l\}_{i\in N} \cup \{t_i^u\}_{i\in N}$, $t_L := -\infty$, $t_U := \infty$.

Step 1 (*Trial multiplier selection*). Choose $\hat{t}$ in $[t_L, t_U] \cap \mathbb{R}$.

Step 2 (*Computing $g(\hat{t})$*). Calculate $g(\hat{t})$.

Step 3 (*Optimality check*). If $g(\hat{t}) = r$, stop with $t_* := \hat{t}$.

Step 4 (*Lower bracket updating*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$.

Step 5 (*Upper bracket updating*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$.

Step 6 (*Stopping criterion*). If $T := T_0 \cap (t_L, t_U) \neq \emptyset$, go to Step 1; otherwise, stop with

$$t_* := t_L - [\, g(t_L) - r \,] \frac{t_U - t_L}{g(t_U) - g(t_L)}. \tag{2.6}$$

At each iteration in Step 2 we have $t_L < t_U$ (since g is nonincreasing). Upon termination, $x^* = x(t_*)$ (cf. Fact 2.1) is recovered via (2.1) in order n operations. Several choices of $\hat{t}$ in Step 1 are given later. We first show how to compute $g(\hat{t})$ efficiently.

3 Updates for evaluating g

3.1 Partitions of the index set

For the current bracket $[t_L, t_U]$, we may partition the set N into the following sets

$$L := \{\, i : t_i^l \leq t_L \,\}, \quad M := \{\, i : t_L, t_U \in [\, t_i^u, t_i^l \,] \,\}, \quad U := \{\, i : t_U \leq t_i^u \,\}, \tag{3.1a}$$

$$I := \{\, i : t_i^l \in (t_L, t_U) \text{ or } t_i^u \in (t_L, t_U) \,\}, \tag{3.1b}$$

which are disjoint because $t_L < t_U$ and $t_i^u \leq t_i^l$ for all i. Further, we have

$$I = I_l \cup I_u \quad \text{with} \quad I_l := \{\, i : t_i^l \in (t_L, t_U) \,\}, \quad I_u := \{\, i : t_i^u \in (t_L, t_U) \,\}, \tag{3.2}$$

and the *bracketed breakpoint set* $T := T_0 \cap (t_L, t_U)$ may be represented as $T = \{t_i^l\}_{i \in I_l} \cup \{t_i^u\}_{i \in I_u}$. Thus $|I| \leq |T|$; in particular, at Step 6 we have $T = \emptyset$ iff $I = \emptyset$.

Frequently it is convenient to partition the set I (cf. (3.2)) into the sets

$$J_m := \{\, i : t_L < t_i^u \leq t_i^l < t_U \,\}, \tag{3.3a}$$

$$J_l := \{\, i : t_i^u \leq t_L < t_i^l < t_U \,\} \quad \text{and} \quad J_u := \{\, i : t_L < t_i^u < t_U \leq t_i^l \,\}, \tag{3.3b}$$

with $I = J_m \cup J_l \cup J_u$, $I_l = J_m \cup J_l$, $I_u = J_m \cup J_u$. Thus $J_m = I_l \cap I_u$, $J_l = I_l \setminus I_u$ and $J_u = I_u \setminus I_l$ index the *middle, lower and upper breakpoints* of $T = \{t_i^l\}_{i \in J_m \cup J_l} \cup \{t_i^u\}_{i \in J_m \cup J_u}$.

To shorten notation, for any subsets $\hat{M}$, $\hat{L}$, $\hat{U}$ of N, we let

$$p(\hat{M}) := \sum_{i \in \hat{M}} a_i b_i / d_i, \quad q(\hat{M}) := \sum_{i \in \hat{M}} b_i^2 / d_i, \quad s_l(\hat{L}) := \sum_{i \in \hat{L}} b_i l_i, \quad s_u(\hat{U}) := \sum_{i \in \hat{U}} b_i u_i. \tag{3.4}$$

3.2 Classical updates

Using (2.2), (2.5), (3.1), (3.4) and the fact $\hat{t} \in [t_L, t_U]$, at Step 2 we have

$$g(\hat{t}) = \sum_{i \in \hat{M}} b_i (a_i - \hat{t} b_i)/d_i + s_l(\hat{L}) + s_u(\hat{U}) + (\, p - \hat{t} q \,) + s, \tag{3.5}$$

where

$$\hat{M} := \{\, i \in I : \hat{t} \in [\, t_i^u, t_i^l \,] \,\}, \quad \hat{L} := \{\, i \in I : t_i^l < \hat{t} \,\}, \quad \hat{U} := \{\, i \in I : \hat{t} < t_i^u \,\}, \tag{3.6}$$

$$p := p(M), \quad q := q(M) \quad \text{and} \quad s := s_l(L) + s_u(U). \tag{3.7}$$

Setting $I := N$, $p, q, s := 0$ at Step 0, at Step 6 we may update I, p, q and s as follows:

$$\begin{array}{l}\text{for } i \in I \text{ do}\\ \quad \text{if } t_i^l \le t_L \text{, set } I := I \setminus \{i\},\ s := s + b_i l_i;\\ \quad \text{if } t_U \le t_i^u \text{, set } I := I \setminus \{i\},\ s := s + b_i u_i;\\ \quad \text{if } t_L, t_U \in [t_i^u, t_i^l] \text{, set } I := I \setminus \{i\},\ p := p + a_i b_i/d_i,\ q := q + b_i^2/d_i.\end{array} \tag{3.8}$$

This update may be improved as follows. Using the sets (cf. (3.6))

$$\check{L} := \{\, i \in I : t_i^l \le \hat{t} \,\} = \hat{L} \cup \hat{I}_l \quad \text{with} \quad \hat{I}_l := \{\, i \in I : t_i^l = \hat{t} \,\}, \tag{3.9a}$$

$$\check{U} := \{\, i \in I : \hat{t} \le t_i^u \,\} = \hat{U} \cup \hat{I}_u \quad \text{with} \quad \hat{I}_u := \{\, i \in I : t_i^u = \hat{t} \,\}, \tag{3.9b}$$

we have $s_l(\check{L}) = s_l(\hat{L}) + s_l(\hat{I}_l)$, $s_u(\check{U}) = s_u(\hat{U}) + s_u(\hat{I}_u)$. Denoting updated quantities with superscript $^+$, we have $L^+ = L \cup \check{L}$ and $U^+ = U$ if $t_L^+ = \hat{t}$, $L^+ = L$ and $U^+ = U \cup \check{U}$ if $t_U^+ = \hat{t}$ (cf. (3.1)). Hence the two middle lines of (3.8) may be replaced by the update

$$s := s + \begin{cases} s_l(\hat{L}) + s_l(\hat{I}_l) & \text{if } g(\hat{t}) > r, \\ s_u(\hat{U}) + s_u(\hat{I}_u) & \text{if } g(\hat{t}) < r. \end{cases} \tag{3.10}$$

Instead of evaluating $s_l(\hat{I}_l)$, $s_u(\hat{I}_u)$ directly, we may use the fact that by (2.4) and (3.4),

$$s_l(\hat{I}_l) = p(\hat{I}_l) - \hat{t}q(\hat{I}_l) \quad \text{and} \quad s_u(\hat{I}_u) = p(\hat{I}_u) - \hat{t}q(\hat{I}_u), \tag{3.11}$$

where $p(\hat{I}_l)$, $q(\hat{I}_l)$, $p(\hat{I}_u)$, $q(\hat{I}_u)$ may be computed during the scan of $\hat{M}$ for (3.5).

When the set T becomes empty, we have $I = \emptyset$ and hence $\hat{M} = \hat{L} = \hat{U} = \emptyset$ in (3.5) for *any* $\hat{t} \in [t_L, t_U]$. Thus $g(t) = p - tq + s \ \forall t \in [t_L, t_U]$, so (2.6) may be replaced by

$$t_* := (p + s - r)/q. \tag{3.12}$$

3.3 Partitioned incremental updates

By (3.4) and (3.5), at Step 2 we have

$$g(\hat{t}) = p(\hat{M}) - \hat{t}q(\hat{M}) + s_l(\hat{L}) + s_u(\hat{U}) + (\,p - \hat{t}q\,) + s. \tag{3.13}$$

Together with the partition $I = J_m \cup J_l \cup J_u$ (cf. (3.3)), we may use the partitions $\hat{M} = \hat{M}_m \cup \hat{M}_l \cup \hat{M}_u$ with

$$\begin{array}{rl} \hat{M}_m := \{\, i \in J_m : t_i^u \le \hat{t} \le t_i^l \,\}, & \hat{M}_l := \{\, i \in J_l : \hat{t} \le t_i^l \,\}, \quad \hat{M}_u := \{\, i \in J_u : t_i^u \le \hat{t} \,\}, \\ \hat{L} = \hat{L}_m \cup \hat{L}_l \quad \text{with} & \hat{L}_m := \{\, i \in J_m : t_i^l < \hat{t} \,\}, \quad \hat{L}_l := \{\, i \in J_l : t_i^l < \hat{t} \,\}, \\ \hat{U} = \hat{U}_m \cup \hat{L}_u \quad \text{with} & \hat{U}_m := \{\, i \in J_m : \hat{t} < t_i^u \,\}, \quad \hat{U}_u := \{\, i \in J_u : \hat{t} < t_i^u \,\}, \\ \hat{I}_l = \hat{J}_m^l \cup \hat{J}_l \quad \text{with} & \hat{J}_m^l := \{\, i \in J_m : t_i^l = \hat{t} \,\}, \quad \hat{J}_l := \{\, i \in J_l : t_i^l = \hat{t} \,\}, \\ \hat{I}_u = \hat{J}_m^u \cup \hat{J}_u \quad \text{with} & \hat{J}_m^u := \{\, i \in J_m : t_i^u = \hat{t} \,\}, \quad \hat{J}_u := \{\, i \in J_u : t_i^u = \hat{t} \,\}. \end{array}$$

Then a single scan of J_m, J_l and J_u suffices for computing the quantities $p(\hat{M}) = p(\hat{M}_m) + p(\hat{M}_l) + p(\hat{M}_u)$, $q(\hat{M}) = q(\hat{M}_m) + q(\hat{M}_l) + q(\hat{M}_u)$, $s_l(\hat{L}) = s_l(\hat{L}_m) + s_l(\hat{L}_l)$ and

$s_u(\hat{U}) = s_u(\hat{U}_m) + s_u(\hat{U}_u)$ required in (3.13), as well as the quantities $p(\hat{I}_l) = p(\hat{J}_m^l) + p(\hat{J}_l)$, $q(\hat{I}_l) = q(\hat{J}_m^l) + q(\hat{J}_l)$, $p(\hat{I}_u) = p(\hat{J}_m^u) + p(\hat{J}_u)$ and $q(\hat{I}_u) = q(\hat{J}_m^u) + q(\hat{J}_u)$ needed for updating s via (3.10)–(3.11).

Further, it is not difficult to see that $M^+ \setminus M = \hat{M}_u$ if $t_L^+ = \hat{t}$, $M^+ \setminus M = \hat{M}_l$ if $t_U^+ = \hat{t}$ (cf. [Kiw08a, §4]). Hence we may update

$$\begin{array}{lll} p := p + p(\hat{M}_u) & \text{and} & q := q + q(\hat{M}_u) \quad \text{if } g(\hat{t}) > r, \\ p := p + p(\hat{M}_l) & \text{and} & q := q + q(\hat{M}_l) \quad \text{if } g(\hat{t}) < r. \end{array} \tag{3.14}$$

When Step 4 sets $t_L := \hat{t}$, a single scan of J_l, J_u and J_m suffices for updating

$$J_l^+ = \{\, i \in J_l : t_L < t_i^l \,\} \cup \{\, i \in J_m : t_i^u \le t_L < t_i^l \,\},$$
$$J_u^+ = \{\, i \in J_u : t_L < t_i^u \,\} \quad \text{and} \quad J_m^+ = \{\, i \in J_m : t_L < t_i^u \,\}.$$

Similarly, when Step 5 sets $t_U := \hat{t}$, a single scan of J_l, J_u and J_m suffices for finding

$$J_u^+ = \{\, i \in J_u : t_i^u < t_U \,\} \cup \{\, i \in J_m : t_i^u < t_U \le t_i^l \,\},$$
$$J_l^+ = \{\, i \in J_l : t_i^l < t_U \,\} \quad \text{and} \quad J_m^+ = \{\, i \in J_m : t_i^l < t_U \,\}.$$

3.4 Decremental updates

Using the partition $I = \hat{M} \cup \hat{L} \cup \hat{U}$ (cf. (3.6)), (3.4) and (3.7) in (3.13) yields

$$g(\hat{t}) = p\left((I \cup M) \setminus (\hat{L} \cup \hat{U})\right) - \hat{t}q\left((I \cup M) \setminus (\hat{L} \cup \hat{U})\right) + s_l(\hat{L}) + s_u(\hat{U}) + s.$$

Hence, in terms of the following complement of $L \cup U$ in N (cf. (3.1), (3.2))

$$K := \{\, i : t_L < t_i^l \text{ and } t_i^u < t_U \,\} = I \cup M = I_l \cup I_u \cup M \tag{3.15}$$

and the associated *redefined* quantities (employed instead of (3.7))

$$p := p(K), \quad q := q(K) \quad \text{and} \quad s := s_l(L) + s_u(U), \tag{3.16}$$

at Step 2 we have

$$g(\hat{t}) = [\,p - p(\hat{L}) - p(\hat{U})\,] - \hat{t}[\,q - q(\hat{L}) - q(\hat{U})\,] + s_l(\hat{L}) + s_u(\hat{U}) + s. \tag{3.17}$$

For the partitions $\hat{L} = \hat{L}_m \cup \hat{L}_l$ and $\hat{U} = \hat{U}_m \cup \hat{U}_u$ of §3.3, a single scan of $J_m \cup J_l \cup J_u$ suffices for computing the quantities $p(\hat{L}) = p(\hat{L}_m) + p(\hat{L}_l)$, $q(\hat{L}) = q(\hat{L}_m) + q(\hat{L}_l)$, $p(\hat{U}) = p(\hat{U}_m) + p(\hat{U}_u)$ and $q(\hat{U}) = q(\hat{U}_m) + q(\hat{U}_u)$ needed in (3.17), as well as the quantities $p(\hat{I}_l)$, $q(\hat{I}_l)$, $p(\hat{I}_u)$ and $q(\hat{I}_u)$ required for updating s via (3.10)–(3.11).

Further, using (3.15), (3.1) and (3.9), we have the partitions $K = K^+ \cup \hat{L} \cup \hat{I}_l$ if $t_L^+ = \hat{t}$, $K = K^+ \cup \hat{U} \cup \hat{I}_u$ if $t_U^+ = \hat{t}$ (cf. [Kiw08a, §5]). Hence we may update

$$\begin{array}{lll} p := p - p(\hat{L}) - p(\hat{I}_l) & \text{and} & q := q - q(\hat{L}) - q(\hat{I}_l) \quad \text{if } g(\hat{t}) > r, \\ p := p - p(\hat{U}) - p(\hat{I}_u) & \text{and} & q := q - q(\hat{U}) - q(\hat{I}_u) \quad \text{if } g(\hat{t}) < r. \end{array} \tag{3.18}$$

We may still compute the final t_* by (3.12) when T becomes empty, because for $I = \emptyset$ we have $K = M$ in (3.15) and hence (3.16) coincides with (3.7).

3.5 Alternative decremental updates

Adding and subtracting (3.11) from (3.17) and using (3.9) yields

$$g(\hat{t}) = [\,p - p(\check{L}) - p(\check{U})\,] - \hat{t}[\,q - q(\check{L}) - q(\check{U})\,] + s_l(\check{L}) + s_u(\check{U}) + s. \tag{3.19}$$

Further, in view of (3.9), the updates (3.10)–(3.11) and (3.18) are equivalent to

$$\begin{array}{llll} p := p - p(\check{L}), & q := q - q(\check{L}) & \text{and} & s := s + s_l(\check{L}) \quad \text{if } g(\hat{t}) > r, \\ p := p - p(\check{U}), & q := q - q(\check{U}) & \text{and} & s := s + s_u(\check{U}) \quad \text{if } g(\hat{t}) < r. \end{array} \tag{3.20}$$

For $\hat{t} \in (t_L, t_U)$, since $\check{L} \cap J_u = \check{U} \cap J_l = \emptyset$ by (3.3b), we may use the partitions

$$\check{L} = \check{L}_m \cup \check{L}_l \quad \text{with} \quad \check{L}_m := \{\, i \in J_m : t_i^l \le \hat{t} \,\}, \quad \check{L}_l := \{\, i \in J_l : t_i^l \le \hat{t} \,\},$$
$$\check{U} = \check{U}_m \cup \check{U}_u \quad \text{with} \quad \check{U}_m := \{\, i \in J_m : \hat{t} \le t_i^u \,\}, \quad \check{U}_u := \{\, i \in J_u : \hat{t} \le t_i^u \,\}.$$

Then a single scan of $J_m \cup J_l \cup J_u$ suffices for computing the quantities $p(\check{L}) = p(\check{L}_m) + p(\check{L}_l)$, $q(\check{L}) = q(\check{L}_m) + q(\check{L}_l)$, $s_l(\check{L}) = s_l(\check{L}_m) + s_l(\check{L}_l)$, $p(\check{U}) = p(\check{U}_m) + p(\check{U}_u)$, $q(\check{U}) = q(\check{U}_m) + q(\check{U}_u)$, $s_u(\check{U}) = s_u(\check{U}_m) + s_u(\check{U}_u)$ involved in (3.19). If $\hat{t} = t_L$ (or $\hat{t} = t_U$), then $g(\hat{t})$ is available from the iteration that updated t_L (or t_U).

3.6 Implicitly partitioned updates

The updates of §§3.3–3.5 may be implemented by using I alone instead of J_m, J_l and J_u.

In the context of §3.3, a single scan of I suffices for computing $s_l(\hat{L})$, $s_u(\hat{U})$, $p(\hat{M}_l)$, $p(\hat{M}_u)$, $p(\hat{M}_m)$, $p(\hat{I}_l)$, etc.; for efficiency, we may exploit the fact that $\hat{I}_l, \hat{I}_u \subset \hat{M} = I \setminus (\hat{L} \cup \hat{U})$, $\hat{M}_l = \{i \in \hat{M} : t_i^u \le t_L\}$, $\hat{M}_u = \{i \in \hat{M} : t_U \le t_i^l\}$, $\hat{M}_m = \hat{M} \setminus (\hat{M}_l \cup \hat{M}_u)$.

Similarly, in a single scan of I we may compute the quantities involved in (3.11) and (3.17), or those in (3.19) via (3.9) and (3.4); the logic is simpler in the latter case.

When Step 4 sets $t_L := \hat{t}$, we may update I as follows:

$$\text{for } i \in I, \text{ set } I := I \setminus \{i\} \text{ if } t_i^l \le t_L, \text{ or } t_i^u \le t_L \text{ and } t_U \le t_i^l. \tag{3.21}$$

Similarly, when Step 5 sets $t_U := \hat{t}$, we may update I as follows:

$$\text{for } i \in I, \text{ set } I := I \setminus \{i\} \text{ if } t_U \le t_i^u, \text{ or } t_U \le t_i^l \text{ and } t_i^u \le t_L. \tag{3.22}$$

Relative to the updates of §§3.3–3.5, their implicit versions using I as above make the same numbers of arithmetic operations, but more comparisons in computing $g(\hat{t})$ and updating I instead of J_m, J_l and J_u. On the other hand, when the index sets are maintained as linked lists, the values in I and J_m are increasing, whereas the values in J_l and J_u may have any order. Thus the implicit versions access memory in more regular ways, and this may make them faster in practice (see §7).

4 Selecting trial multipliers

4.1 Exact medians

In the framework of [Kiw08a], the trial multiplier $\hat{t}$ is chosen in the bracketed breakpoint set $T := T_0 \cap (t_L, t_U)$. Setting $T := \{t_i^l\}_{i\in N} \cup \{t_i^u\}_{i\in N}$ at Step 0, we may update

$$T := \begin{cases} \{\, t \in T : \hat{t} < t \,\} & \text{if } g(\hat{t}) > r, \\ \{\, t \in T : t < \hat{t} \,\} & \text{if } g(\hat{t}) < r; \end{cases} \tag{4.1}$$

alternatively we may recover $T = \{t_i^l\}_{i\in I_l} \cup \{t_i^u\}_{i\in I_u}$. Since each iteration reduces T, the algorithm is finite. Computing $g(\hat{t})$ as in §3 takes order $|I| \le |T|$ operations. Thus, for an arbitrary choice of $\hat{t}$ in T, Algorithm 2.2 requires order n^2 operations in the worst case.

The complexity can be improved to order n by selecting $\hat{t}$ as the median of T, which requires order $|T|$ operations; see, e.g., [Knu98, §5.3.3]. Thus the complexity of each iteration is $O(|T|)$. Since $|T|$ is originally $2n$ and is approximately halved at each iteration, the algorithm makes $O(\log n)$ iterations in time $O(n)$.

For the median finding routine of [Kiw05], which permutes T to place elements $< \hat{t}$ first, then elements $= \hat{t}$, and finally elements $> \hat{t}$, the update of (4.1) requires only a change of one pointer. For less smart routines, the effort in maintaining T may be mitigated by using a slightly larger set $\tilde{T}$ as follows. Setting $\tilde{T} := T_0$ at Step 0 and $\hat{t} := \text{median}(\tilde{T})$ at Step 1, set

$$\tilde{T} := \begin{cases} \{\, t \in \tilde{T} : \hat{t} \le t \,\} \setminus \{\, \hat{t} \,\} & \text{if } g(\hat{t}) > r, \\ \{\, t \in \tilde{T} : t \le \hat{t} \,\} \setminus \{\, \hat{t} \,\} & \text{if } g(\hat{t}) < r, \end{cases} \tag{4.2}$$

without removing from $\tilde{T}$ all $t = \hat{t}$ as in (4.1). If the median-finding routine permutes $\tilde{T}$ to place elements $\le \hat{t}$ before $\hat{t}$ and elements $\ge \hat{t}$ after $\hat{t}$, the update (4.2) of $\tilde{T}$ requires only a change of one pointer. Since $|\tilde{T}^+| \le \frac{1}{2}|\tilde{T}|$, linear-time complexity is retained.

4.2 Random median estimates

As suggested by [PaK90], to avoid the effort of finding $\hat{t} := \text{median}(T)$, in practice it may be preferable to choose $\hat{t}$ in T at random, with an expected number of iterations of $O(\log n)$ in an expected time $O(n)$. When the set I is maintained as a linked list, locating $\hat{t}$ requires an additional scan of I (half of I on average).

4.3 Random sample median estimates

An obvious extension of the choice of §4.2 is to use $\hat{t} := \text{median}(\hat{T})$, where $\hat{T}$ is a randomly chosen subset of T. In practice the average performance improves even for fairly small $\hat{T}$ (e.g., $|\hat{T}| = 20$), for which the cost of median finding is negligible. However, locating $\hat{T}$ needs an additional scan of I.

4.4 Average median estimates

Yet another estimate of median(T) is provided by the *average* $\hat{t} := \frac{1}{|T|}\sum_{t\in T} t$. The cost of finding $\hat{t}$ is small, since $\sum_{t\in T} t$ is easily computed while updating I or J_m, J_l and J_u. Since $\hat{t} \in [t_{\min}, t_{\max}]$, where $t_{\min} := \min_{t\in T} t$ and $t_{\max} := \max_{t\in T} t$, we have $|T^+| < |T|$ and hence the algorithm is finite. However, in practice cycling may occur when $\hat{t} \notin [t_{\min}, t_{\max}]$ due to rounding errors. Hence if our implementation discovers that $|T^+| = |T|$, then $t_{\min}$ and $t_{\max}$ are computed and the next $\hat{t}$ is projected onto $[t_{\min}, t_{\max}]$. This safeguard is cheap, because usually it activates only once, when $|T|$ is small.

5 Variable fixing methods

5.1 A symmetric variable fixing algorithm

We first give a slightly unusual description of the algorithm of [Kiw08b, §3], using the notation of (3.15)–(3.16) and $\hat{x}(t) := D^{-1}(a - tb)$, so that $x(t) = \min\{\max[l, \hat{x}(t)], u\}$.

Algorithm 5.1.
Step 0 (*Initiation*). Set $K := N$, $p := p(N)$, $q := q(N)$, $s := 0$, $t_L := -\infty$, $t_U := \infty$.

Step 1 (*Trial multiplier selection*). Set $\hat{t} := (p + s - r)/q$.

Step 2 (*Computing $g(\hat{t})$*). Set $g(\hat{t}) := r + \nabla - \Delta$, where

$$\nabla := \sum_{i\in K_l} b_i[\, l_i - \hat{x}_i(\hat{t})\,] \quad \text{with} \quad K_l := \{\, i \in K : \hat{x}_i(\hat{t}) \le l_i \,\}, \tag{5.1a}$$

$$\Delta := \sum_{i\in K_u} b_i[\, \hat{x}_i(\hat{t}) - u_i\,] \quad \text{with} \quad K_u := \{\, i \in K : \hat{x}_i(\hat{t}) \ge u_i \,\}. \tag{5.1b}$$

Step 3 (*Optimality check*). If $g(\hat{t}) = r$, stop with $t_* := \hat{t}$.

Step 4 (*Lower bracket updating*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $K := K \setminus K_l$, $p := p - p(K_l)$, $q := q - q(K_l)$, $s := s + s_l(K_l)$.

Step 5 (*Upper bracket updating*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $K := K \setminus K_u$, $p := p - p(K_u)$, $q := q - q(K_u)$, $s := s + s_u(K_u)$.

Step 6 (*Loop*). Go to Step 1.

5.2 Convergence of the variable fixing algorithm

Algorithm 5.1 may be validated inductively in the following way, which extends the analysis of [Kiw08b, §4] by exposing additional properties of the method.

Suppose (3.15)–(3.16) hold at Step 1 with $K \neq \emptyset$, $g(t_L) > r$ if $t_L > -\infty$, $g(t_U) < r$ if $t_U < \infty$. Then $\hat{g}(t) := \sum_{i\in K} b_i\hat{x}_i(t) + s$ is a decreasing function of t ($b > 0$), expressible as $\hat{g}(t) = p - tq + s$ thanks to (3.16), with $\hat{g}(\hat{t}) = r$ by the choice of $\hat{t}$. If $t_L > -\infty$, then, using (2.2), (2.5), the partition $N = K \cup L \cup U$ (cf. (3.1), (3.15)) and the fact $s = s_l(L) + s_u(U)$ (cf. (3.16)), we get $g(t_L) = \sum_{i\in K} b_i x_i(t_L) + s$ with $x_i(t_L) = \min\{u_i, \hat{x}_i(t_L)\} \le \hat{x}_i(t_L)$ if

$i \notin L$. Thus $\hat{g}(t_L) \geq g(t_L) > r$ and hence $t_L < \hat{t}$ because $\hat{g}(\hat{t}) = r$ and $\hat{g}$ is decreasing. If $t_U < \infty$, a symmetric argument yields $\hat{t} < t_U$.

Consequently, $\hat{t} \in (t_L, t_U)$. Hence again (2.5) gives $g(\hat{t}) = \sum_{i \in K} b_i x_i(\hat{t}) + s$, whereas (5.1) yields $x_i(\hat{t}) = l_i \ \forall i \in K_l$, $x_i(\hat{t}) = u_i \ \forall i \in K_u$, $x_i(\hat{t}) = \hat{x}_i(\hat{t}) \ \forall i \in K \setminus (K_l \cup K_u)$, so using $\hat{g}(\hat{t}) = r$ gives $g(\hat{t}) - r = \sum_{i \in K} b_i[x_i(\hat{t}) - \hat{x}_i(\hat{t})] = \nabla - \Delta$, as required in Step 2.

Next, note that (2.4), (5.1) and the fact that $\hat{x}_i(\hat{t}) = (a_i - \hat{t}b_i)/d_i$ imply

$$K_l = \{\, i \in K : t_i^l \leq \hat{t} \,\} \quad \text{and} \quad K_u = \{\, i \in K : \hat{t} \leq t_i^u \,\}. \tag{5.2}$$

Hence, by (3.1), the updates of Steps 4 and 5 maintain (3.15)–(3.16). If Step 4 produced $K = \emptyset$, (2.5) and (3.1) with $I = M = \emptyset$ would give $g(t) = g(t_L) > r \ \forall t \in [t_L, t_U]$, but for $t \in T_*$ we would have $g(t) = r$, a contradiction. Similarly, Step 5 produces $K \neq \emptyset$.

Finally, we observe that Steps 4 and 5 reduce K (e.g., $g(\hat{t}) > r$ at Step 2 implies $\nabla > \Delta$ and hence $K_l \neq \emptyset$). It follows that Algorithm 5.1 is finite.

5.3 Standard implementation

A straightforward implementation of Algorithm 5.1 doesn't use the breakpoints. At Step 2 a single scan of K suffices for computing $\nabla_x := \sum_{i \in K_l} b_i \hat{x}_i(\hat{t})$, $\Delta_x := \sum_{i \in K_u} b_i \hat{x}_i(\hat{t})$, $s_l(K_l)$ and $s_u(K_u)$; then $\nabla = s_l(K_l) - \nabla_x$ and $\Delta = \Delta_x - s_u(K_u)$ by (5.1) and (3.4). Another scan of K is needed for updating K, as well as computing $p(K_l)$ and $q(K_l)$ at Step 4, or $p(K_u)$ and $q(K_u)$ at Step 5. Both scans compute $\hat{x}_i(\hat{t})$ for $i \in K$.

5.4 Breakpoint interpretation and implementation

A breakpoint-based implementation of Algorithm 5.1 is derived as follows. Since $\hat{t} \in (t_L, t_U)$ implies $t_i^u < \hat{t} < t_i^l \ \forall i \in M$ (cf. (3.1a)), we may replace $K := I \cup M$ by I in (5.2), so $K_l = \check{L}$ and $K_u = \check{U}$ by (3.9). Hence Algorithm 5.1 is a special case of Algorithm 2.2 which uses $\hat{t} := (p + s - r)/q$, computes $g(\hat{t})$ via (3.17) (or (3.19)) and employs the updates of §3.4 (or §3.5), except that when Algorithm 2.2 terminates at Step 6 with t_* given by (3.12), Algorithm 5.1 makes an additional iteration with $\hat{t} = t_*$.

6 Initial problem transformations

6.1 A general breakpoint-preserving transformation

To reduce work per iteration, changing the variables via $x = H\bar{x} + D^{-1}a$ with $H := \mathrm{diag}(h)$ and $h > 0$, we may initially transform P into the equivalent problem

$$\bar{\mathrm{P}}: \quad \min \quad \bar{f}(\bar{x}) := \tfrac{1}{2}\bar{x}^T \bar{D}\bar{x} - \bar{a}^T\bar{x} - c_0 \quad \text{s.t.} \quad \bar{b}^T\bar{x} = \bar{r}, \quad \bar{l} \leq \bar{x} \leq \bar{u}, \tag{6.1}$$

where $\bar{D} := \mathrm{diag}(\bar{d})$ with $\bar{d} := H^2 d$, $\bar{a} := 0$, $c_0 := \frac{1}{2}a^T D^{-1} a$, $\bar{b} := Hb$, $\bar{r} := r - b^T D^{-1} a$, $\bar{l} := H^{-1}(l - D^{-1}a)$, $\bar{u} := H^{-1}(u - D^{-1}a)$. Then $\bar{x}(t) := \min\{\max[\bar{l}, -t\bar{D}^{-1}\bar{b}], \bar{u}\}$ (cf. (2.1)), and $\bar{g}(t) := \bar{b}^T\bar{x}(t)$ has the breakpoints $\bar{t}_i^l := -\bar{l}_i\bar{d}_i/\bar{b}_i = t_i^l$, $\bar{t}_i^u := -\bar{u}_i\bar{d}_i/\bar{b}_i = t_i^u$, $i \in N$ (cf. (2.4)), i.e., the breakpoints are preserved by this transformation.

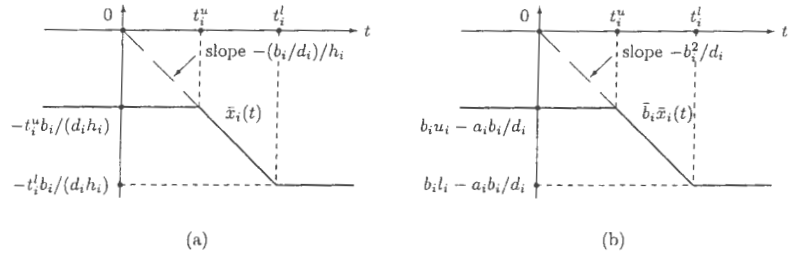

Figure 6.1: (a) Illustration of $\bar{x}_i(t)$ produced by the transformation of §6.1. (b) Illustration of $\bar{b}_i\bar{x}_i(t)$ produced by the transformation of §6.1.

More insight into this transformation stems from the facts that $\bar{b}_i\bar{x}_i(t) = b_i x_i(t) - a_i b_i/d_i$, $i \in N$, and $\bar{g}(t) - \bar{r} = g(t) - r$ with $\bar{r} = r - p(N)$ (cf. (3.4)); thus $g(t) = r$ is transformed naturally into $\bar{g}(t) = \bar{r}$ (cf. Figs. 2.1 and 6.1). Further, (3.4) becomes: $\bar{p}(\cdot) = 0$, $\bar{q}(\cdot) := \sum_{i\in\cdot} \bar{b}_i^2/\bar{d}_i = q(\cdot)$, $\bar{s}_l(\cdot) := \sum_{i\in\cdot} \bar{b}_i\bar{l}_i = s_l(\cdot) - p(\cdot)$, $\bar{s}_u(\cdot) := \sum_{i\in\cdot} \bar{b}_i\bar{u}_i = s_u(\cdot) - p(\cdot)$.

These properties imply that each multiplier selection from §§4-5 produces the *same* sequence of multipliers for both P and $\bar{\mathrm{P}}$. This is clear for the breakpoint-based selections of §4, whereas for $\hat{t} := (p(K) + s_l(L) + s_u(U) - r)/q(K)$ as in §5 we have $p(N) = p(K) + p(L) + p(U)$ from the partition $N = K \cup L \cup U$, so $\hat{t} = (\bar{p}(K) + \bar{s}_l(L) + \bar{s}_u(U) - \bar{r})/\bar{q}(K)$.

We consider two choices of H below. Let $e := (1, \ldots, 1) \in \mathbb{R}^n$.

6.2 The Robinson–Jiang–Lerme transformation

The choice $H = D^{-1/2}$ of [RJL92, BSS96] yields $\bar{d} = e$ and $\bar{b} = D^{-1/2}b$ in (6.1). Thus, regarding $\bar{\mathrm{P}}$ as an instance of P with $d = e$, $a = 0$, we have $p(\cdot) = 0$ and $q(\cdot) = \sum_{i\in\cdot} b_i^2$ in (3.4), so p disappears and q in (3.7) and (3.16) is updated with less effort. Further, the form of $\hat{x}(t) = -tb$ needs less work in the variable fixing implementation of §5.3.

6.3 A more refined transformation

The choice of $H = D^{-1}B$ with $B := \mathrm{diag}(b)$ ($h_i = b_i/d_i$) yields $\bar{d}_i = \bar{b}_i = b_i^2/d_i$, $\bar{l}_i = -t_i^l$, $\bar{u}_i = -t_i^u$, $\bar{t}_i^l = -\bar{l}_i$, $\bar{t}_i^u = -\bar{u}_i$, $i \in N$. Note that $\bar{r} = r - p(N)$, whereas the cost of forming $\bar{d} = \bar{b}$ is comparable to that of finding $q(N)$ (cf. (3.4)). In terms of Fig. 6.1, the offsets $a_i b_i/d_i$ are removed and the slopes $-\bar{b}_i = -b_i^2/d_i$ are computed "once for all".

Now, we may regard $\bar{\mathrm{P}}$ as an instance of P with $d = b > 0$ and $a = 0$. In this case $p(\cdot) = 0$ and $q(\cdot) = \sum_{i\in\cdot} b_i$ in (3.4), so p disappears and q in (3.7) and (3.16) is updated with less effort. Further, the breakpoints $t_i^l = -l_i$ and $t_i^u = -u_i$ needn't be stored.

Compared to the first choice of §6.2, this transformation saves work in updating q and computing $\hat{x}(t) = -te$, and avoids the square root computations of the first one.

The updates of §5.3 for the variable fixing method simplify as follows. A single scan of K suffices for computing $q(K_l)$, $q(K_u)$, $s_l(K_l)$, $s_u(K_u)$ with $K_l = \{i \in K : -\hat{t} \le l_i\}$,

$K_u = \{i \in K : u_i \leq -\hat{t}\}$; then $\nabla_x = -\hat{t}q(K_l)$ and $\Delta_x = -\hat{t}q(K_u)$. Another scan of K simply updates $K := \{i \in K : l_i < -\hat{t}\}$ at Step 4, or $K := \{i \in K : -\hat{t} < u_i\}$ at Step 5.

7 Numerical results

More than forty routines implementing different versions of Algorithms 2.2 and 5.1 were programmed in Fortran 77 and run on a notebook PC (Pentium M 755 2 GHz, 1.5 GB RAM) under MS Windows XP. For Algorithm 2.2, the routines have names of the form `k1bxyz`, where `x` is `m` for the median breakpoint selection of §4.1, `r` for the random selection of §4.2, `s` for the random sample selection of §4.3 with $|\hat{T}| \leq 20$, `a` for the average selection of §4.4, `f` for the variable fixing selection of §5.4; next, `y` is `o` for P in its original form, `r` for P transformed as in §6.2, `t` for P transformed as in §6.3; finally, `z` is `a`, `b` or `c` for the updates of §3.3, §3.4 or §3.5, respectively, and `z` is `d`, `e` or `f` for these updates implemented implicitly (cf. §3.6). The routines implementing Algorithm 5.1 as in §5.3 have names of the form `k1vfy`, where `y` is `o`, `r` or `t` as above.

Routines k1bmyz computed medians of T via the method of [Kiw05].

The sets J_m, J_l, J_u, I and K were maintained as linked lists in an integer*4 array of length n. Additional real*8 storage varied between $6n$ and $10n$ for P in its original form, and between $9n$ and $11n$ for P in its transformed form; in the latter case, the storage may be reduced by $3n$ if some of the original data can be overwritten, as explained below.

For P in its original form, k1vfo uses six real*8 arrays of length n (for a, b, d, l, u, x), k1baoz, k1bfoz, k1broz, k1bsoz need an additional array of length $2 * n$ (for breakpoints), whereas k1bmoz needs two such arrays (for breakpoints and median finding). For P in its transformed form, k1batz, k1bftz, k1brtz, k1bstz, k1vfr and k1vft need three additional arrays of length n (for $\bar{b}, \bar{l}, \bar{u}$), whereas k1bmtz also needs an array of length $2 * n$ (for median finding). To save storage, the three arrays holding b, l, u on input may also hold $\bar{b}, \bar{l}, \bar{u}$ on output. To this end, we copy b to x initially, and use this copy for the final computation of $x^* = x(t_*)$ and $g(t_*) = b^T x^*$. For the transformation of §6.2, we set $x_i^* := (d_i^{1/2}\bar{x}_i(t_*) + a_i)/d_i$ with $\bar{x}_i(t_*) := \min\{\max[\bar{l}_i, -t_*\bar{b}_i], \bar{u}_i\}$, whereas for the transformation of §6.3, we set $x_i^* := (b_i\bar{x}_i(t_*) + a_i)/d_i$ with $\bar{x}_i(t_*) := \min\{\max[\bar{l}_i, -t_*], \bar{u}_i\}$, $i \in N$.

To account for rounding errors, Step 3 employed the stopping criterion $|\bar{g}(\hat{t}) - \bar{r}| \leq 10^{-10}\max\{1, |r|\}$, with $\bar{g}(\hat{t}) - \bar{r}$ replaced by $g(\hat{t}) - r$ for P in its original form.

Our test problems were randomly generated with n ranging between 50000 and 2000000. As in [BSS95, §2], all parameters were distributed uniformly in the intervals of the following three problem classes: (1) uncorrelated: $a_i, b_i, d_i \in [10, 25]$; (2) weakly correlated: $b_i \in [10, 25]$, $a_i, d_i \in [b_i - 5, b_i + 5]$; (3) strongly correlated: $b_i \in [10, 25]$, $a_i = d_i = b_i + 5$; further, $l_i, u_i \in [1, 15]$, $i \in N$, $r \in [b^T l, b^T u]$. For each problem size, 20 instances were generated in each class. Table 7.1 gives the average, maximum and minimum numbers of free variables (the final $|M|$) for the test problems.

Tables 7.2–7.4 report the average, maximum and minimum run times in seconds over the 20 instances for each of the listed problem sizes and classes, as well as overall statistics, for the standard implementations of variable fixing methods. The run times include initial problem transformations and the final computations of $x(t_*)$ and $g(t_*)$ for the original

Table 7.1: Numbers of free variables in test problems.

	uncorrelated			weakly correl.			strongly correl.		
n	avg	max	min	avg	max	min	avg	max	min
50000	15340	21157	1749	19302	23717	5685	17684	24924	2781
100000	34021	42831	7031	40320	47331	9920	42068	49847	34653
500000	152760	211930	44415	183117	237225	74184	217206	248494	121387
1000000	315116	422942	58172	359899	474184	76174	406475	494977	208761
1500000	474849	635823	93226	529721	710771	156708	536501	746876	78995
2000000	649078	848731	127835	735040	944503	67341	771609	995125	116545

Table 7.2: Run times of k1vfo (standard variable fixing, original P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.02	0.01	0.02	0.02	0.01	0.02	0.02	0.01	0.02	0.02	0.01
100000	0.05	0.05	0.04	0.05	0.05	0.04	0.05	0.05	0.04	0.05	0.05	0.04
500000	0.25	0.28	0.22	0.24	0.27	0.22	0.23	0.25	0.20	0.24	0.28	0.20
1000000	0.50	0.55	0.45	0.48	0.55	0.44	0.48	0.50	0.44	0.49	0.55	0.44
1500000	0.72	0.83	0.59	0.72	0.81	0.66	0.71	0.75	0.58	0.72	0.83	0.58
2000000	0.97	1.08	0.88	0.94	1.04	0.78	0.95	1.00	0.88	0.95	1.08	0.78

Table 7.3: Run times of k1vfr (standard variable fixing, RJL transformation of P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.02	0.01
100000	0.03	0.04	0.03	0.03	0.04	0.03	0.03	0.04	0.03	0.03	0.04	0.03
500000	0.20	0.22	0.18	0.19	0.21	0.18	0.19	0.21	0.17	0.19	0.22	0.17
1000000	0.40	0.42	0.37	0.39	0.42	0.36	0.38	0.40	0.36	0.39	0.42	0.36
1500000	0.58	0.63	0.55	0.58	0.63	0.54	0.58	0.60	0.50	0.58	0.63	0.50
2000000	0.79	0.84	0.73	0.76	0.81	0.67	0.77	0.80	0.73	0.77	0.84	0.67

Table 7.4: Run times of k1vft (standard variable fixing, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.00	0.01	0.03	0.00
100000	0.03	0.03	0.02	0.02	0.03	0.02	0.02	0.03	0.02	0.02	0.03	0.02
500000	0.16	0.16	0.15	0.15	0.17	0.14	0.15	0.16	0.13	0.15	0.17	0.13
1000000	0.31	0.33	0.29	0.30	0.33	0.28	0.30	0.31	0.28	0.31	0.33	0.28
1500000	0.46	0.50	0.43	0.46	0.49	0.43	0.45	0.47	0.39	0.45	0.50	0.39
2000000	0.62	0.66	0.57	0.60	0.63	0.52	0.60	0.63	0.57	0.60	0.66	0.52

Table 7.5: Run times of k1bfoe (breakpoint variable fixing, original P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.01	0.03	0.01	0.01	0.04	0.01	0.02	0.05	0.01	0.02	0.05	0.01
100000	0.03	0.04	0.02	0.03	0.03	0.02	0.03	0.03	0.02	0.03	0.04	0.02
500000	0.15	0.18	0.14	0.15	0.17	0.13	0.14	0.16	0.14	0.15	0.18	0.13
1000000	0.31	0.36	0.28	0.30	0.34	0.28	0.29	0.32	0.28	0.30	0.36	0.28
1500000	0.46	0.54	0.41	0.45	0.50	0.41	0.45	0.53	0.39	0.45	0.54	0.39
2000000	0.62	0.71	0.57	0.60	0.69	0.53	0.59	0.71	0.54	0.60	0.71	0.53

Table 7.6: Run times of k1bfte (breakpoint variable fixing, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.02	0.00	0.01	0.02	0.00
100000	0.02	0.03	0.02	0.02	0.02	0.02	0.02	0.03	0.02	0.02	0.03	0.02
500000	0.12	0.14	0.11	0.12	0.13	0.10	0.11	0.12	0.11	0.12	0.14	0.10
1000000	0.24	0.27	0.22	0.23	0.26	0.22	0.23	0.24	0.21	0.23	0.27	0.21
1500000	0.35	0.40	0.31	0.34	0.38	0.32	0.34	0.41	0.30	0.35	0.41	0.30
2000000	0.47	0.53	0.43	0.45	0.52	0.41	0.45	0.54	0.41	0.46	0.54	0.41

problem P. Tables 7.5 and 7.6 give the run times for the breakpoint implementations of §5.4 with the updates of §3.4 implemented implicitly as in §3.6.

In view of Tables 7.2–7.6 and the discussion of §6, in what follows we ignore the first transformation and illustrate only the influence of the second one for exact median selections in Tables 7.7–7.8 and for random median selections in Tables 7.9–7.10.

Tables 7.11–7.12 report the run times for sample and average selections. As can be seen from Tables 7.10–7.11, increasing the sample size from 1 to 20 is beneficial; cf. [Grü99]. However, further increases needn't bring much improvement. For instance, the run times decreased by just 2% for a variation of the strategy of [Pis97, §2] in which the sample size was set to $\min\{|T|^{1/2}, 1000\}$ if $|T| \geq 100$, or to 3 otherwise.

Table 7.7: Run times of k1bmoe (exact medians, original P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.03	0.02	0.02	0.05	0.02	0.02	0.04	0.02	0.02	0.05	0.02
100000	0.05	0.05	0.05	0.05	0.05	0.04	0.05	0.05	0.04	0.05	0.05	0.04
500000	0.24	0.25	0.23	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.25	0.23
1000000	0.48	0.48	0.47	0.48	0.49	0.47	0.48	0.48	0.47	0.48	0.49	0.47
1500000	0.72	0.72	0.71	0.72	0.72	0.71	0.72	0.72	0.71	0.72	0.72	0.71
2000000	0.96	0.97	0.95	0.96	0.96	0.95	0.96	0.96	0.95	0.96	0.97	0.95

Table 7.8: Run times of klbmte (exact medians, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.04	0.01	0.02	0.06	0.01	0.03	0.04	0.01	0.02	0.06	0.01
100000	0.04	0.04	0.03	0.04	0.04	0.03	0.04	0.04	0.03	0.04	0.04	0.03
500000	0.20	0.20	0.19	0.20	0.20	0.19	0.20	0.20	0.19	0.20	0.20	0.19
1000000	0.40	0.40	0.39	0.40	0.40	0.39	0.40	0.40	0.39	0.40	0.40	0.39
1500000	0.59	0.60	0.59	0.60	0.61	0.59	0.59	0.60	0.59	0.59	0.61	0.59
2000000	0.79	0.80	0.78	0.79	0.79	0.78	0.79	0.79	0.78	0.79	0.80	0.78

Table 7.9: Run times of klbroe (random medians, original P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.04	0.02	0.03	0.06	0.01	0.04	0.06	0.02	0.03	0.06	0.01
100000	0.05	0.07	0.03	0.06	0.09	0.04	0.05	0.07	0.03	0.05	0.09	0.03
500000	0.28	0.35	0.17	0.28	0.39	0.17	0.28	0.46	0.16	0.28	0.46	0.16
1000000	0.60	0.89	0.35	0.56	0.71	0.32	0.56	0.87	0.26	0.57	0.89	0.26
1500000	0.85	1.24	0.53	0.81	1.13	0.54	0.89	1.30	0.40	0.85	1.30	0.40
2000000	1.12	1.65	0.81	1.28	1.91	0.73	1.22	1.95	0.66	1.21	1.95	0.66

Table 7.13 reports the final iteration numbers for various methods; here klbfte behaved like klvfo, klvfr, klvft and klbfoe, klbmoe like klbmte, and klbroe like klbrte.

Tables 7.5–7.13 give details only for klbxye, i.e., the decremental updates of §3.4 implemented implicitly, because they tended to be fastest as illustrated in Table 7.14 for large problems, although klbxyd or klbxyf were sometimes marginally faster. To achieve comparable performance of all updates, most programming effort went into reducing the average number of comparisons while computing $g(\hat{t})$ and updating J_m, J_l, J_u or I (our first, less sophisticated implementations had been much slower). We add that the improved classical updates of §3.2 were slower than klbxye by up to 10%.

Of course, our limited experiments do not warrant firm conclusions, but it seems safe to make the following comments.

The average run times of all methods grew linearly with the problem size.

The updates of §§3.3–3.5 (in klbxya, klbxyb, klbxyc) were comparable in speed, but slower than their implicit versions of §3.6 (in klbxyd, klbxye, klbxyf).

The advantages of our more refined problem transformation were clear both for the variable fixing methods (cf. Tabs. 7.2–7.4 and Tabs. 7.5–7.6) and for the slower versions of breakpoint searching methods (cf. Tabs. 7.7–7.8 and Tabs. 7.9–7.10).

The breakpoint implementations of the variable fixing method (klbfoz and klbftz in Tab. 7.14) were faster than the standard implementations (cf. Tabs. 7.2–7.4).

The relatively good performance of the exact median versions was due to the high efficiency of the median finding routine of [Kiw05]. In particular, for problem P in its

Table 7.10: Run times of k1brte (random medians, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.03	0.04	0.01	0.03	0.04	0.01	0.04	0.06	0.01	0.03	0.06	0.01
100000	0.03	0.05	0.02	0.03	0.05	0.02	0.03	0.04	0.02	0.03	0.05	0.02
500000	0.21	0.26	0.14	0.22	0.29	0.14	0.21	0.33	0.14	0.21	0.33	0.14
1000000	0.44	0.64	0.29	0.42	0.52	0.25	0.41	0.62	0.22	0.42	0.64	0.22
1500000	0.62	0.87	0.41	0.60	0.81	0.43	0.66	0.93	0.33	0.62	0.93	0.33
2000000	0.82	1.17	0.61	0.93	1.32	0.59	0.88	1.37	0.51	0.88	1.37	0.51

Table 7.11: Run times of k1bste (median samples of size 20, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.04	0.01	0.02	0.04	0.01	0.03	0.06	0.01	0.03	0.06	0.01
100000	0.03	0.04	0.02	0.03	0.04	0.03	0.03	0.03	0.02	0.03	0.04	0.02
500000	0.18	0.21	0.15	0.18	0.21	0.16	0.17	0.20	0.15	0.18	0.21	0.15
1000000	0.37	0.43	0.29	0.36	0.41	0.30	0.36	0.41	0.30	0.36	0.43	0.29
1500000	0.54	0.59	0.50	0.54	0.64	0.43	0.52	0.59	0.45	0.53	0.64	0.43
2000000	0.72	0.86	0.59	0.71	0.78	0.59	0.71	0.82	0.63	0.71	0.86	0.59

original form, k1bmoe was as fast as k1vfo (cf. Tabs. 7.2 and 7.7). This contradicts the popular belief that variable fixing methods are much faster than median based methods. For instance, [RJL92] reported the median-based method of [Bru84] to be about 8 times slower than its variable fixing implementation. We add that the median-based methods of [Bru84, CaM87] were slower than k1bmoe by about 36%; see [Kiw08a].

Relative to the exact medians, single random selections performed quite well on average, but exhibited much larger variations in run times. Moderately sized random samples gave smaller averages and variations in run times (cf. Tabs. 7.10–7.11). Yet the average median estimates were even better; in fact they performed similarly to standard implementations of variable fixing methods (cf. Tabs. 7.4 and 7.12).

Table 7.12: Run times of k1bate (average selections, transformed P).

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.04	0.01	0.02	0.04	0.01	0.02	0.04	0.00	0.02	0.04	0.00
100000	0.02	0.03	0.02	0.02	0.03	0.02	0.03	0.03	0.02	0.02	0.03	0.02
500000	0.15	0.16	0.14	0.16	0.16	0.15	0.15	0.15	0.15	0.15	0.16	0.14
1000000	0.31	0.32	0.28	0.31	0.32	0.30	0.31	0.31	0.30	0.31	0.32	0.28
1500000	0.47	0.48	0.42	0.47	0.47	0.46	0.47	0.47	0.46	0.47	0.48	0.42
2000000	0.62	0.64	0.56	0.62	0.63	0.58	0.62	0.63	0.60	0.62	0.64	0.56

Table 7.13: Iteration numbers for various multiplier selections.

		uncorrelated			weakly correl.			strongly correl.		
routine	n	avg	max	min	avg	max	min	avg	max	min
klbate	1000000	21	23	21	21	22	21	21	23	21
	2000000	22	24	21	22	24	21	22	25	22
klbfte	1000000	7	10	7	7	9	6	7	8	6
	2000000	7	10	7	7	10	6	7	10	7
klbmoe	1000000	20	21	20	21	21	21	20	21	20
	2000000	21	22	21	22	22	22	21	22	21
klbroe	1000000	30	43	19	29	37	20	27	39	16
	2000000	29	40	20	31	41	24	30	43	19
klbste	1000000	21	25	19	21	25	19	22	26	19
	2000000	22	26	18	22	25	19	22	24	19

Acknowledgment. I would like to acknowledge discussions with N. Kovoor, N. Maculan, J.J. Moré, P.M. Pardalos, D. Pisinger, A.G. Robinson, B. Shetty and P. Toth.

References

[BiH81] G. R. Bitran and A. C. Hax, *Disaggregation and resource allocation using convex knapsack problems with bounded variables*, Management Sci. **27** (1981) 431–441.

[BrS97] K. M. Bretthauer and B. Shetty, *Quadratic resource allocation with generalized upper bounds*, Oper. Res. Lett. **20** (1997) 51–57.

[Bru84] P. Brucker, *An $O(n)$ algorithm for quadratic knapsack problems*, Oper. Res. Lett. **3** (1984) 163–166.

[BSS95] K. M. Bretthauer, B. Shetty and S. Syam, *A branch and bound algorithm for integer quadratic knapsack problems*, ORSA J. Comput. **7** (1995) 109–116.

[BSS96] ———, *A projection method for the integer quadratic knapsack problem*, J. Oper. Res. Soc. **47** (1996) 457–462.

[CaM87] P. H. Calamai and J. J. Moré, *Quasi-Newton updates with bounds*, SIAM J. Numer. Anal. **24** (1987) 1434–1441.

[CDZ86] R. W. Cottle, S. G. Duvall and K. Zikan, *A Lagrangean relaxation algorithm for the constrained matrix problem*, Naval Res. Logist. Quart. **33** (1986) 55–76.

[CoH94] S. Cosares and D. S. Hochbaum, *Strongly polynomial algorithms for the quadratic transportation problem with a fixed number of sources*, Math. Oper. Res. **19** (1994) 94–111.

[Grü99] R. Grübel, *On the median-of-k version of Hoare's selection algorithm*, Theor. Inform. Appl. **33** (1999) 177–192.

[HKL80] K. Helgason, J. Kennington and H. Lall, *A polynomially bounded algorithm for a singly constrained quadratic program*, Math. Program. **18** (1980) 338–343.

[HoH95] D. S. Hochbaum and S. P. Hong, *About strongly polynomial time algorithms for quadratic optimization over submodular constraints*, Math. Program. **69** (1995) 269–309.

[HWC74] M. Held, P. Wolfe and H. P. Crowder, *Validation of subgradient optimization*, Math. Program. **6** (1974) 62–88.

[Kiw05] K. C. Kiwiel, *On Floyd and Rivest's* SELECT *algorithm*, Theoretical Computer Sci. **347** (2005) 214–238.

Table 7.14: Run times of various updates for $n = 2000000$.

	uncorrelated			weakly correl.			strongly correl.			overall		
routine	avg	max	min	avg	max	min	avg	max	min	avg	max	min
k1bata	0.83	0.87	0.70	0.82	0.83	0.74	0.81	0.81	0.78	0.82	0.87	0.70
k1batb	0.82	0.85	0.69	0.80	0.82	0.73	0.80	0.86	0.77	0.81	0.86	0.69
k1batc	0.82	0.85	0.69	0.80	0.82	0.73	0.80	0.80	0.76	0.80	0.85	0.69
k1batd	0.65	0.66	0.58	0.64	0.65	0.60	0.64	0.65	0.62	0.64	0.66	0.58
k1bate	0.62	0.64	0.56	0.62	0.63	0.58	0.62	0.63	0.60	0.62	0.64	0.56
k1batf	0.63	0.64	0.57	0.62	0.63	0.58	0.62	0.63	0.60	0.62	0.64	0.57
k1bfob	0.74	0.92	0.65	0.69	0.83	0.59	0.68	0.86	0.59	0.71	0.92	0.59
k1bfoc	0.73	0.91	0.64	0.69	0.83	0.59	0.68	0.85	0.58	0.70	0.91	0.58
k1bfoe	0.62	0.71	0.57	0.60	0.69	0.53	0.59	0.71	0.54	0.60	0.71	0.53
k1bfof	0.60	0.70	0.55	0.58	0.67	0.52	0.58	0.68	0.52	0.59	0.70	0.52
k1bftb	0.55	0.68	0.48	0.53	0.62	0.46	0.52	0.65	0.46	0.53	0.68	0.46
k1bftc	0.55	0.68	0.48	0.52	0.62	0.46	0.52	0.64	0.46	0.53	0.68	0.46
k1bfte	0.47	0.53	0.43	0.45	0.52	0.41	0.45	0.54	0.41	0.46	0.54	0.41
k1bftf	0.46	0.52	0.43	0.45	0.51	0.41	0.45	0.53	0.41	0.45	0.53	0.41
k1bmoa	1.21	1.23	1.19	1.21	1.22	1.19	1.21	1.21	1.19	1.21	1.23	1.19
k1bmob	1.23	1.24	1.22	1.22	1.23	1.21	1.21	1.22	1.21	1.22	1.24	1.21
k1bmoc	1.23	1.24	1.21	1.22	1.23	1.21	1.21	1.22	1.20	1.22	1.24	1.20
k1bmod	0.93	0.94	0.91	0.94	0.95	0.92	0.94	0.95	0.92	0.94	0.95	0.91
k1bmoe	0.96	0.97	0.95	0.96	0.96	0.95	0.96	0.96	0.95	0.96	0.97	0.95
k1bmof	0.96	0.96	0.95	0.96	0.96	0.95	0.96	0.96	0.95	0.96	0.96	0.95
k1bmta	0.98	0.99	0.97	0.98	1.00	0.97	0.98	0.98	0.97	0.98	1.00	0.97
k1bmtb	0.97	0.98	0.96	0.97	0.97	0.96	0.96	0.97	0.96	0.97	0.98	0.96
k1bmtc	0.97	0.98	0.96	0.96	0.97	0.96	0.96	0.97	0.96	0.97	0.98	0.96
k1bmtd	0.81	0.81	0.80	0.81	0.82	0.80	0.81	0.82	0.80	0.81	0.82	0.80
k1bmte	0.79	0.80	0.78	0.79	0.79	0.78	0.79	0.79	0.78	0.79	0.80	0.78
k1bmtf	0.79	0.79	0.78	0.79	0.79	0.78	0.79	0.79	0.78	0.79	0.79	0.78
k1broa	1.71	4.33	0.67	1.69	3.12	0.76	1.79	3.40	0.79	1.73	4.33	0.67
k1brob	1.74	4.33	0.73	1.70	3.12	0.78	1.78	3.37	0.83	1.74	4.33	0.73
k1broc	1.73	4.32	0.73	1.70	3.12	0.78	1.78	3.34	0.84	1.74	4.32	0.73
k1brod	1.09	1.65	0.77	1.27	1.89	0.73	1.20	1.95	0.60	1.19	1.95	0.60
k1broe	1.12	1.65	0.81	1.28	1.91	0.73	1.22	1.95	0.66	1.21	1.95	0.66
k1brof	1.11	1.64	0.80	1.28	1.90	0.72	1.21	1.94	0.65	1.20	1.94	0.65
k1brta	1.32	3.34	0.58	1.30	2.37	0.63	1.36	2.61	0.67	1.33	3.34	0.58
k1brtb	1.28	3.23	0.57	1.26	2.29	0.62	1.33	2.51	0.65	1.29	3.23	0.57
k1brtc	1.28	3.22	0.57	1.25	2.28	0.63	1.31	2.50	0.65	1.28	3.22	0.57
k1brtd	0.86	1.23	0.64	0.98	1.39	0.63	0.92	1.44	0.52	0.92	1.44	0.52
k1brte	0.82	1.17	0.61	0.93	1.32	0.59	0.88	1.37	0.51	0.88	1.37	0.51
k1brtf	0.83	1.17	0.61	0.94	1.33	0.59	0.88	1.38	0.51	0.88	1.38	0.51
k1bsta	0.92	1.20	0.71	0.89	1.10	0.68	0.90	1.10	0.67	0.90	1.20	0.67
k1bstb	0.90	1.17	0.69	0.87	1.08	0.67	0.88	1.08	0.66	0.88	1.17	0.66
k1bstc	0.90	1.17	0.69	0.87	1.08	0.67	0.88	1.08	0.65	0.88	1.17	0.65
k1bstd	0.76	0.91	0.61	0.74	0.82	0.62	0.74	0.85	0.66	0.74	0.91	0.61
k1bste	0.72	0.86	0.59	0.71	0.78	0.59	0.71	0.82	0.63	0.71	0.86	0.59
k1bstf	0.72	0.86	0.58	0.71	0.79	0.59	0.71	0.82	0.62	0.71	0.86	0.58

[Kiw08a] ———, *Breakpoint searching algorithms for the continuous quadratic knapsack problem*, Math. Program. **112** (2008) 473–491.

[Kiw08b] ———, *Variable fixing algorithms for the continuous quadratic knapsack problem*, J. Optim. Theory Appl. **136** (2008) 445–458.

[Knu98] D. E. Knuth, *The Art of Computer Programming. Volume III: Sorting and Searching*, second ed., Addison-Wesley, Reading, MA, 1998.

[MdP89] N. Maculan and G. G. de Paula, Jr., *A linear-time median-finding algorithm for projecting a vector on the simplex of* R^n, Oper. Res. Lett. **8** (1989) 219–222.

[MMP97] N. Maculan, M. Minoux and G. Plateau, *An* $O(n)$ *algorithm for projecting a vector on the intersection of a hyperplane and* R^n_+, RAIRO Rech. Opér. **31** (1997) 7–16.

[MPT00] S. Martello, D. Pisinger and P. Toth, *New trends in exact algorithms for the* 0–1 *knapsack problem*, European J. Oper. Res. **123** (2000) 325–332.

[MSMJ03] N. Maculan, C. P. Santiago, E. M. Macambira and M. H. C. Jardim, *An* $O(n)$ *algorithm for projecting a vector on the intersection of a hyperplane and a box in* R^n, J. Optim. Theory Appl. **117** (2003) 553–574.

[NiZ92] S. S. Nielsen and S. A. Zenios, *Massively parallel algorithms for singly constrained convex programs*, ORSA J. Comput. **4** (1992) 166–181.

[PaK90] P. M. Pardalos and N. Kovoor, *An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds*, Math. Program. **46** (1990) 321–328.

[Pis97] D. Pisinger, *A minimal algorithm for the* 0–1 *knapsack problem*, Oper. Res. **46** (1997) 758–767.

[RJL92] A. G. Robinson, N. Jiang and C. S. Lerme, *On the continuous quadratic knapsack problem*, Math. Program. **55** (1992) 99–108.

[ShM90] B. Shetty and R. Muthukrishnan, *A parallel projection for the multicommodity network model*, J. Oper. Res. Soc. **41** (1990) 837–842.

[Ste04] S. M. Stefanov, *Convex quadratic minimization subject to a linear constraint and box constraints*, Appl. Math. Res. eXpress no. 1 (2004) 17–42.

[Ven91] J. A. Ventura, *Computational development of a Lagrangian dual approach for quadratic networks*, Networks **21** (1991) 469–485.