45/2006

Raport Badawczy
Research Report

RB/59/2006

Breakpoint searching algorithms 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) 8373578

fax: (+48) (22) 8372772

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

Warszawa 2006

Math. Program., Ser. A
DOI 10.1007/s10107-006-0050-z

FULL LENGTH PAPER

Breakpoint searching algorithms for the continuous quadratic knapsack problem

Krzysztof C. Kiwiel

Received: 4 March 2006 / Accepted: 29 September 2006

Abstract We give several linear time algorithms for the continuous quadratic knapsack problem. In addition, we report cycling and wrong-convergence examples in a number of existing algorithms, and give encouraging computational results for large-scale problems.

Keywords Nonlinear programming · Convex programming · Quadratic programming · Separable programming · Singly constrained quadratic program

Mathematics Subject Classification (2000) 65K05 · 90C25

1 Introduction

The *continuous quadratic knapsack problem* is defined by

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

where x is an n-vector of variables, $a, b, l, u \in \mathbb{R}^n$, $r \in \mathbb{R}$, $D = \operatorname{diag}(d)$ 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 [2,3,13], hierarchical production planning [2], network flows [26], transportation problems [9], multicommodity network flows [12,22,25], constrained matrix problems [10], integer

K. C. Kiwiel (✉)
Systems Research Institute, Polish Academy of Sciences,
Newelska 6, 01-447, Warsaw, Poland
e-mail: kiwiel@ibspan.waw.pl

quadratic knapsack problems [4,5], integer and continuous quadratic optimization over submodular constraints [13], Lagrangian relaxation via subgradient optimization [11], and quasi-Newton updates with bounds [7].

Specialized algorithms for P solve its dual problem 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. Sect. 2). The earliest $O(n \log n)$ methods [11,12] sort the breakpoints initially, whereas the $O(n)$ algorithms [6,7,9,13,18,19,23] use medians of breakpoint subsets (see [1,20] for extensions); [23] also proposed an approximate median version with an average-case performance of $O(n)$. Another class of methods with worst-case performance of $O(n^2)$ [2,5,21,24,26,27] employs variable fixing [17].

This paper focuses on linear time algorithms for P. The existing algorithms differ in two aspects: (1) the choice of the current breakpoint subset for which the median is found; and (2) the updates of quantities used for evaluating the function g at the median.

As for the first aspect, we give a breakpoint searching framework that is conceptually simpler than those in [6,7,9,13,18,19,23]. In particular, the simplest method resulting from our framework seems to be competitive in practice with the more complex methods of [6,7] (see Sect. 10). Moreover, we show that the remaining methods [9,13,18,19,23] may cycle on simple examples, due to insufficient reduction of the breakpoint subsets.

Concerning the second aspect, we introduce a more refined version of the standard g-evaluations of [6,7], and a complementary one that extends some ideas in [9,13]; their practical performance will be discussed elsewhere [15].

The paper is organized as follows. Basic properties of P are reviewed in Sect. 2. Our simplest algorithm is introduced in Sect. 3 together with the standard g-evaluation of [6,7]. A more refined g-evaluation is derived in Sect. 4, and a complementary one in Sect. 5. To ease comparisons with related methods, in Sect. 6 we state simplifications for quadratic resource allocation. Extensions of the two median approach of [6] and the additional breakpoint removal of [7] are discussed in Sects. 7 and 8, respectively. Section 9 discusses relations with the methods of [9,13,18,19,23]. Finally, preliminary computational results for large-scale problems are reported in Sect. 10.

2 Basic properties of the problem

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 $f(x) + t(b^{\mathrm{T}}x - r)$ 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), its *constraint value*

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

and the associated multipliers for the constraints $l - x \leq 0$ and $x - u \leq 0$, respectively,

$$\mu(t) := \max\{Dl - a + tb, 0\} \quad \text{and} \quad \nu(t) := \max\{a - tb - Du, 0\}. \tag{2.3}$$

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

$$T_* := \{t : g(t) = r\}. \tag{2.4}$$

Indeed, invoking the Karush–Kuhn–Tucker conditions for P as in [7, Theorem 2.1], [12, Sect. 2], [22, Sect. 2], [23, Theorem 2.1] gives the following result.

Fact 2.1 $x^* = x(t)$ iff $t \in T_*$. Further, the set T_* is nonempty, and t, $\mu(t)$, $\nu(t)$ are Lagrange multipliers of P whenever $t \in T_*$.

As in [6], 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*

$$t_i^l := \frac{(a_i - l_i d_i)}{b_i} \quad \text{and} \quad t_i^u := \frac{(a_i - u_i d_i)}{b_i}, \qquad i = 1 : n. \tag{2.5}$$

Note that $t_i^u \leq t_i^l$ from $l_i \leq u_i$ and $b_i > 0$ in (2.5). Further, each $x_i(t)$ may be expressed as

$$x_i(t) = \begin{cases} u_i & \text{if } t \leq t_i^u, \\ (a_i - tb_i)/d_i & \text{if } t_i^u \leq t \leq t_i^l, \\ l_i & \text{if } t_i^l \leq t. \end{cases} \tag{2.6}$$

Thus $g(t)$ is a continuous, piecewise linear and *nonincreasing* function of t (see Fig. 1).

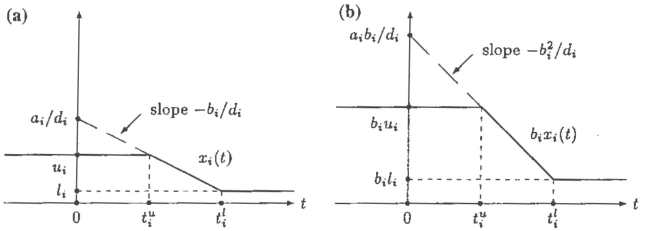

Fig. 1 **a** Illustration of $x_i(t) := \min\{\max[l_i, (a_i - tb_i)/d_i], u_i\}$. **b** Illustration of $b_i x_i(t) = \min\{\max[b_i l_i, (a_i b_i - tb_i^2)/d_i], b_i u_i\}$ (for $b_i > 0$)

Hence the optimal set T_* of (2.4) is an interval (possibly infinite) of the form

$$T_* = \left[t_L^*, t_U^*\right] \cap \mathbb{R} \quad \text{with} \quad t_L^* := \inf\{t : g(t) = r\}, \quad t_U^* := \sup\{t : g(t) = r\}, \tag{2.7}$$

with $g(t_L^*) = r$ if $t_L^* > -\infty$, $g(t_U^*) = r$ if $t_U^* < \infty$; clearly, $g(t) > r$ iff $t < t_L^*$, $g(t) < r$ iff $t_U^* < t$. Denoting the *minimal and maximal breakpoints* by $t_{\min}^u := \min_i t_i^u$ and $t_{\max}^l := \max_i t_i^l$, we have $g(t) = b^T u \geq r$ for all $t \leq t_{\min}^u$, $g(t) = b^T l \leq r$ for all $t \geq t_{\max}^l$.

3 The breakpoint searching algorithm

In this section we state our algorithm and discuss its simplest implementation.

The algorithm below generates successive nondecreasing *underestimates* t_L of t_L^* and nonincreasing *overestimates* t_U of t_U^* in (2.7) by evaluating g at trial breakpoints in (t_L, t_U) until t_L and t_U become two consecutive breakpoints; then g is linear on $[t_L, t_U]$, and t_* is found by interpolation. Let $N := \{1: n\}$ denote the set of all variables.

Algorithm 3.1

STEP 0 (*Initiation*). Set $T_0 := \{t_i^l\}_{i\in N} \cup \{t_i^u\}_{i\in N}$, $T := T_0$, $t_L := -\infty$, $t_U := \infty$.
STEP 1 (*Breakpoint selection*). Choose a breakpoint $\hat{t}$ in T.
STEP 2 (*Computing $g(\hat{t})$*). Calculate the constraint value $g(\hat{t})$.
STEP 3 (*Optimality check*). If $g(\hat{t}) = r$, stop with $t_* := \hat{t}$.
STEP 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$.
STEP 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$.
STEP 6 (*Stopping criterion*). If $T \neq \emptyset$, go to Step 1; otherwise, stop with

$$t_* := t_L - [g(t_L) - r]\frac{t_U - t_L}{g(t_U) - g(t_L)}. \tag{3.1}$$

The following comments clarify the nature of the algorithm.

Remark 3.2 (a) At each iteration in Step 2 we have $t_L < t_U$, $T_* \subset [t_L, t_U]$ and $\hat{t} \in T = T_0 \cap (t_L, t_U)$ (this follows by induction from the properties of g given in Sect. 2).

(b) To compute $g(\hat{t})$ efficiently, we may partition the set N into the following sets

$$L := \left\{i : t_i^l \leq t_L\right\}, \quad M := \left\{i : t_L, t_U \in \left[t_i^u, t_i^l\right]\right\}, \quad U := \left\{i : t_U \leq t_i^u\right\}, \tag{3.2a}$$

$$I := \left\{i : t_i^l \in (t_L, t_U) \text{ or } t_i^u \in (t_L, t_U)\right\}, \tag{3.2b}$$

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

$$I = I_l \cup I_u \quad \text{with} \quad I_l := \left\{i : t_i^l \in (t_L, t_U)\right\}, \quad I_u := \left\{i : t_i^u \in (t_L, t_U)\right\}, \tag{3.3}$$

and

$$T = \{t_i^l\}_{i \in I_l} \cup \{t_i^u\}_{i \in I_u}; \tag{3.4}$$

hence $|I| \le |T|$. Thus, by (2.2), (2.6) and (3.2),

$$g(t) = \sum_{i \in I} b_i x_i(t) + (p - tq) + s \quad \forall t \in [t_L, t_U], \tag{3.5}$$

where

$$\sum_{i \in I} b_i x_i(t) = \sum_{i \in I : t \in [t_i^u, t_i^l]} \frac{b_i(a_i - tb_i)}{d_i} + \sum_{i \in I : t_i^l < t} b_i l_i + \sum_{i \in I : t < t_i^u} b_i u_i, \tag{3.6}$$

$$p := \sum_{i \in M} \frac{a_i b_i}{d_i}, \quad q := \sum_{i \in M} \frac{b_i^2}{d_i} \quad \text{and} \quad s := \sum_{i \in L} b_i l_i + \sum_{i \in U} b_i u_i. \tag{3.7}$$

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 and the calculation of $g(\hat{t})$ due to [6] require order $|I| \le |T|$ operations.

(c) When the set T becomes empty, then $I = \emptyset$ in (3.5), so g is linear on $[t_L, t_U]$ and (3.1) yields $g(t_*) = r$. (Note that $g(t_L)$ and $g(t_U)$ must have been evaluated earlier: $t_U = \infty$ would imply $t_L = t_{\max}^l$ and $g(t_L) = b^{\mathrm{T}} l \le r$, contradicting $g(t_L) > r$ (cf. Step 4); similarly $t_L = -\infty$ would yield $t_U = t_{\min}^u$ and $g(t_U) = b^{\mathrm{T}} u \ge r$, another contradiction.) Alternatively, (3.5) with $I = \emptyset$ shows that (3.1) is equivalent to

$$t_* := \frac{(p + s - r)}{q}. \tag{3.9}$$

(d) Since each iteration reduces the set T, Algorithm 3.1 must terminate with $t_* \in T_*$; then $x^* = x(t_*)$ (cf. Fact 2.1) is recovered via (2.1) in order n operations [cf. (2.6)].

The choice of $\hat{t}$ in T at Step 1 is crucial for efficiency, as explained below.

Remark 3.3 (a) For an arbitrary choice of $\hat{t}$, Algorithm 3.1 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., [8, Sect. 9.3]. Thus the complexity of each iteration is $O(|T|)$. Since $|T|$ is originally $2n$ and is at least halved at each iteration, the total work is of order $2n + n + n/2 + \cdots = 4n$. Thus the algorithm makes $O(\log n)$ iterations in time $O(n)$; see, e.g., [7, p. 1438] for a more general proof.

(b) As suggested by [23], 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 $O(n)$, which can be derived as in [8, Sect. 9.2].

We now briefly describe several useful modifications.

Remark 3.4 (a) Step 0 may set $t_L := t^u_{\min}$, $t_U := t^l_{\max}$, $T := T_0 \cap (t_L, t_U)$, terminating with $t_* := t_L$ if $g(t_L) = r$, or $t_* := t_U$ if $g(t_U) = r$, or t_* given by (3.9) if $T = \emptyset$.

(b) If the set of fixed variables $L^= := \{i : l_i = u_i\}$ is nonempty, at Step 0 we may set $I := N \setminus L^=$, $T := \{t^l_i, t^u_i\}_{i\in I}$, replace L by $L \cup L^=$ in (3.2) and (3.7), modify U and I accordingly, and terminate with any $t_* \in \mathbb{R}$ if $T = \emptyset$.

(c) An extension to infinite bounds is easy, since $t^l_i = \infty$ iff $l_i = -\infty$, $t^u_i = -\infty$ iff $u_i = \infty$. Step 0 may set $T := \{t^l_i\}_{i\in I_l} \cup \{t^u_i\}_{i\in I_u}$ with I_l, I_u given by (3.3), terminating with t_* given by (3.9) if $I = \emptyset$. Thus infinite breakpoints are effectively ignored.

4 More refined updates

In a simple implementation based on (3.5)–(3.8), certain sums of (3.6) are repeated in (3.8). We now give a more refined version of Algorithm 3.1 that eliminates these redundancies.

Our refinement consists in using the following partition of the set I [cf. (3.3)] into

$$J_m := \left\{ i : t_L < t^u_i \le t^l_i < t_U \right\}, \tag{4.1a}$$

$$J_l := \left\{ i : t^u_i \le t_L < t^l_i < t_U \right\} \quad \text{and} \quad J_u := \left\{ i : t_L < t^u_i < t_U \le t^l_i \right\}, \tag{4.1b}$$

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 $T = \{t^l_i\}_{i\in J_m\cup J_l} \cup \{t^u_i\}_{i\in J_m\cup J_u}$. To shorten notation, for any subsets $\hat{M}$, $\hat{L}$, $\hat{U}$ of N, we let [cf. (3.7)]

$$p(\hat{M}) := \sum_{i\in\hat{M}} \frac{a_i b_i}{d_i}, \quad q(\hat{M}) := \sum_{i\in\hat{M}} \frac{b_i^2}{d_i}, \quad s_l(\hat{L}) := \sum_{i\in\hat{L}} b_i l_i, \quad s_u(\hat{U}) := \sum_{i\in\hat{U}} b_i u_i. \tag{4.2}$$

Algorithm 4.1

STEP 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$, $T := \{t^l_i\}_{i\in J_m\cup J_l} \cup \{t^u_i\}_{i\in J_m\cup J_u}$ with J_m, J_l, J_u given by (4.1), $p := p(M)$, $q := q(M)$, $s := s_l(L) + s_u(U)$ with M, L, U given by (3.2).

STEP 1 (*Breakpoint selection*). Choose a breakpoint $\hat{t}$ in T.

STEP 2 (*Computing $g(\hat{t})$*). Set $\hat{M}_m := \{i \in J_m : t^u_i \le \hat{t} \le t^l_i\}$, $\hat{M}_l := \{i \in J_l : \hat{t} \le t^l_i\}$, $\hat{M}_u := \{i \in J_u : t^u_i \le \hat{t}\}$, $\hat{L} := \{i \in I_l : t^l_i < \hat{t}\}$, $\hat{U} := \{i \in I_u : \hat{t} < t^u_i\}$,

$\hat{p} := p + p(\hat{M}_m) + p(\hat{M}_l) + p(\hat{M}_u)$, $\hat{q} := q + q(\hat{M}_m) + q(\hat{M}_l) + q(\hat{M}_u)$, $\hat{s} := s + s_l(\hat{L}) + s_u(\hat{U})$, $g(\hat{t}) = (\hat{p} - \hat{t}\hat{q}) + \hat{s}$.

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

STEP 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$, $p := p + p(\hat{M}_u)$, $q := q + q(\hat{M}_u)$, $\hat{I}_l := \{i \in I_l : t_i^l = \hat{t}\}$, $s := s + s_l(\hat{L}) + s_l(\hat{I}_l)$.

STEP 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$, $p := p + p(\hat{M}_l)$, $q := q + q(\hat{M}_l)$, $\hat{I}_u := \{i \in I_u : t_i^u = \hat{t}\}$, $s := s + s_u(\hat{U}) + s_u(\hat{I}_u)$.

STEP 6 (*Stopping criterion*). If $T \neq \emptyset$, go to Step 1, else stop with t_* given by (3.9).

The sums in Step 2 require a single scan of $I = J_m \cup J_l \cup J_u$; another scan suffices for updating J_m, J_l and J_u at Step 4 or 5 [cf. (4.1); for brevity, explicit updates are omitted]. The work of Step 2 is comparable to that in using (3.5), (3.6); however, relative to (3.8), Steps 4 and 5 save the work needed for (re)computing the sums $p(\hat{M}_u)$, $q(\hat{M}_u)$, etc., available from Step 2. Thus the efficiency estimates of Remark 3.3 remain valid for Algorithm 4.1. It remains to show that the algorithm is correct.

Theorem 4.2 *Algorithm* 4.1 *terminates with* $t_* \in T_*$.

Proof To validate the calculation of $g(\hat{t})$ at Step 2, suppose $\hat{t} \in (t_L, t_U)$ and (3.7) holds (this is true initially; cf. Step 0). Then (3.3) and (4.1) with $t_L \le \hat{t} \le t_U$ imply that $\hat{M}_m$, $\hat{M}_l$ and $\hat{M}_u$ form a partition of $\hat{M} := \{i \in I : t_i^u \le \hat{t} \le t_i^l\}$, with $\hat{M}_m = \hat{M} \cap J_m$, $\hat{M}_l = \hat{M} \cap J_l$, $\hat{M}_u = \hat{M} \cap J_u$, whereas $\hat{M}$ together with $\hat{L} = \{i \in I : t_i^l < \hat{t}\}$ and $\hat{U} = \{i \in I : \hat{t} < t_i^u\}$ form a partition of I. Hence (3.6) and (4.2) yield

$$\sum_{i \in I} b_i x_i(\hat{t}) = p(\hat{M}) - \hat{t}q(\hat{M}) + s_l(\hat{L}) + s_u(\hat{U})$$

$$= p(\hat{M}_m) + p(\hat{M}_l) + p(\hat{M}_u) - \hat{t}[\, q(\hat{M}_m) + q(\hat{M}_l) + q(\hat{M}_u)\,] + s_l(\hat{L}) + s_u(\hat{U}).$$

Combining this with (3.5) and (3.7) shows that Step 2 computes $g(\hat{t})$ correctly.

Thus, as long as (3.7) holds, Algorithm 4.1 may be identified with Algorithm 3.1. We now show that (3.7) is maintained by the updates of Steps 4 and 5, using superscript $^+$ for the updated quantities, e.g., p^+.

First, suppose $t_L^+ = \hat{t}$ at Step 4. Since $t_L \le t_L^+$ and t_U does not change, $U^+ = U$ by (3.2) and $I \setminus I^+$ splits into $M^+ \setminus M$ and $L^+ \setminus L$. The first set $M^+ \setminus M$ consists of $i \in I$ such that $t_i^u \le \hat{t} \le t_i^l$ and $t_i^u \le t_U \le t_i^l$, so, since $t_i^u < t_U\ \forall i \in I$, it coincides with the intersection of $\hat{M}$ and $\{i \in I : t_U \le t_i^l\} = J_u$ [cf. (4.1)], which is $\hat{M}_u$. The second set $L^+ \setminus L$ equals $\check{L} := \{i \in I : t_i^l \le \hat{t}\}$ $(t_L^+ = \hat{t})$, with $\check{L} = \{i \in I_l : t_i^l \le \hat{t}\}$ [using $\hat{t} < t_U$ in (3.3)]. Thus $M^+ = M \cup \hat{M}_u$ with $M \cap \hat{M}_u = \emptyset$, $L^+ = L \cup \check{L}$ with $L \cap \check{L} = \emptyset$, $U^+ = U$. Further, $\check{L} = \hat{L} \cup \hat{I}_l$ with $\hat{L} \cap \hat{I}_l = \emptyset$. Combining the preceding relations with (3.7) and (4.2) gives $p^+ = p(M) + p(\hat{M}_u) = p(M^+)$, $q^+ = q(M) + q(\hat{M}_u) = q(M^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\check{L}) = s_l(L^+) + s_u(U^+)$. Thus (3.7) holds for the updated quantities.

Next, suppose $t_U^+ = \hat{t}$ at Step 5. Since $t_U^+ \le t_U$ and t_L does not change, $L^+ = L$ by (3.2) and $I \setminus I^+$ splits into $M^+ \setminus M$ and $U^+ \setminus U$. The first set $M^+ \setminus M$ consists of $i \in I$ such that $t_i^u \le \hat{t} \le t_i^l$ and $t_i^u \le t_L \le t_i^l$, so, since $t_L < t_i^l \ \forall i \in I$, it coincides with the intersection of $\hat{M}$ and $\{i \in I : t_i^u \le t_L\} = J_l$ [cf. (4.1)], which is $\hat{M}_l$. The second set $U^+ \setminus U$ equals $\check{U} := \{i \in I : \hat{t} \le t_i^u\}$ $(t_U^+ = \hat{t})$, with $\check{U} = \{i \in I_u : \hat{t} \le t_i^u\}$ [using $t_L < \hat{t}$ in (3.3)]. Thus $M^+ = M \cup \hat{M}_l$ with $M \cap \hat{M}_l = \emptyset$, $U^+ = U \cup \check{U}$ with $U \cap \check{U} = \emptyset$, $L^+ = L$. Further, $\check{U} = \hat{U} \cup \hat{I}_u$ with $\hat{U} \cap \hat{I}_u = \emptyset$. Combining the preceding relations with (3.7) and (4.2) gives $p^+ = p(M) + p(\hat{M}_l) = p(M^+)$, $q^+ = q(M) + q(\hat{M}_l) = q(M^+)$, $s^+ = s_l(L) + s_u(U) + s_u(\check{U}) = s_l(L^+) + s_u(U^+)$. Thus (3.7) holds for the updated quantities.

It follows by induction that (3.7) always holds at Steps 2 and 6.

Upon termination with $T = \emptyset$, $t_* \in T_*$ by Remark 3.2(c). □

5 Decremental updates

Algorithm 4.1 works with the quantities $p = p(M)$, $q = q(M)$, $s = s_l(L) + s_u(U)$, incrementing them when M, L and U grow. Using the set [cf. (3.2), (3.3)]

$$K := \left\{ i : t_L < t_i^l \text{ and } t_i^u < t_U \right\} = I \cup M = I_l \cup I_u \cup M, \tag{5.1}$$

we now describe a version of Algorithm 3.1 that employs the *redefined* quantities

$$p = p(K), \quad q = q(K) \quad \text{and} \quad s = s_l(L) + s_u(U), \tag{5.2}$$

decrementing p and q when K shrinks; this idea stems from [9,13].

Algorithm 5.1

STEP 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$, $T := \{t_i^l\}_{i \in I_l} \cup \{t_i^u\}_{i \in I_u}$ with I_l, I_u given by (3.3), set p, q, s via (5.1), (5.2) with I, M, L, U given by (3.2).

STEP 1 (*Breakpoint selection*). Choose a breakpoint $\hat{t}$ in T.

STEP 2 (*Computing* $g(\hat{t})$). Set $\hat{L} := \{i \in I_l : t_i^l < \hat{t}\}$, $\hat{U} := \{i \in I_u : \hat{t} < t_i^u\}$, $\hat{p} := p - p(\hat{L}) - p(\hat{U})$, $\hat{q} := q - q(\hat{L}) - q(\hat{U})$, $\hat{s} := s + s_l(\hat{L}) + s_u(\hat{U})$, $g(\hat{t}) = (\hat{p} - \hat{t}\hat{q}) + \hat{s}$.

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

STEP 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$, $\hat{I}_l := \{i \in I_l : t_i^l = \hat{t}\}$, $p := p - p(\hat{L}) - p(\hat{I}_l)$, $q := q - q(\hat{L}) - q(\hat{I}_l)$, $s := s + s_l(\hat{L}) + s_l(\hat{I}_l)$, $I_l := \{i \in I_l : \hat{t} < t_i^l\}$, $I_u := \{i \in I_u : \hat{t} < t_i^u\}$.

STEP 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$, $\hat{I}_u := \{i \in I_u : t_i^u = \hat{t}\}$, $p := p - p(\hat{U}) - p(\hat{I}_u)$, $q := q - q(\hat{U}) - q(\hat{I}_u)$, $s := s + s_u(\hat{U}) + s_u(\hat{I}_u)$, $I_l := \{i \in I_l : t_i^l < \hat{t}\}$, $I_u := \{i \in I_u : t_i^u < \hat{t}\}$.

STEP 6 (*Stopping criterion*). If $T \neq \emptyset$, go to Step 1, else stop with t_* given by (3.9).

The work of Step 2 in computing $\hat{p}$, $\hat{q}$ is proportional to $|\hat{L}| + |\hat{U}|$, whereas that of Algorithm 4.1 is proportional to $|\hat{M}|$, with $|\hat{M}| + |\hat{L}| + |\hat{U}| = |I|$ (cf. the proof of Theorem 4.2). Hence again the efficiency estimates of Remark 3.3 remain valid, and we need only show that the algorithm is correct.

Theorem 5.2 *Algorithm* 5.1 *terminates with* $t_* \in T_*$.

Proof To validate the calculation of $g(\hat{t})$ at Step 2, suppose $\hat{t} \in (t_L, t_U)$ and (5.2) holds (this is true initially; cf. Step 0). Using (2.6), (3.2), (3.3), (4.2), (5.1) and (5.2), we may express $g(\hat{t}) := \sum_{i\in N} b_i x_i(\hat{t})$ as

$$g(\hat{t}) = \sum_{i\in K} b_i x_i(\hat{t}) + \sum_{i\in L} b_i l_i + \sum_{i\in U} b_i u_i = \sum_{i\in K} b_i x_i(\hat{t}) + s, \tag{5.3a}$$

where in the notation of Step 2 (with $\hat{L}, \hat{U} \subset K$, $\hat{L} \cap \hat{U} = \emptyset$ from $t_i^u \le t_i^l$) we have

$$\begin{aligned}\sum_{i\in K} b_i x_i(\hat{t}) &= \sum_{i\in K\setminus(\hat{L}\cup\hat{U})} b_i x_i(\hat{t}) + \sum_{i\in\hat{L}} b_i l_i + \sum_{i\in\hat{U}} b_i u_i\\ &= \left[p\left(K\setminus(\hat{L}\cup\hat{U})\right) - \hat{t}q\left(K\setminus(\hat{L}\cup\hat{U})\right)\right]\\ &\quad + s_l(\hat{L}) + s_u(\hat{U}) \qquad\qquad (5.3b)\\ &= \left[p(K) - p(\hat{L}) - p(\hat{U})\right] - \hat{t}\left[q(K) - q(\hat{L}) - q(\hat{U})\right]\\ &\quad + s_l(\hat{L}) + s_u(\hat{U}).\end{aligned}$$

Relations (5.3) and (5.2) show that Step 2 computes $g(\hat{t})$ correctly.

Thus, as long as (5.2) holds, Algorithm 5.1 may be identified with Algorithm 3.1. We now show that (5.2) is maintained by the updates of Steps 4 and 5, using superscript $^+$ for the updated quantities, e.g., p^+.

First, suppose $t_L^+ = \hat{t}$ at Step 4. Let $\check{L} := \{i \in I_l : t_i^l \le \hat{t}\}$. Then $K = K^+ \cup \check{L}$ with $K^+ = \{i : \hat{t} < t_i^l \text{ and } t_i^u < t_U\}$ and $K^+ \cap \check{L} = \emptyset$ by (5.1) and (3.3), whereas the partition (3.2) yields $L \cup U = N \setminus K$ and $L^+ \cup U^+ = N \setminus K^+$ with $U^+ = U$ and $\check{L} \cap U = \emptyset$, so $L^+ = L \cup \check{L}$ with $L \cap \check{L} = \emptyset$. Further, $\check{L} = \hat{L} \cup \hat{I}_l$ with $\hat{L} \cap \hat{I}_l = \emptyset$ at Step 2. Combining the preceding relations with (5.2) and the rules of Step 4 gives $p^+ = p(K) - p(\check{L}) = p(K^+)$, $q^+ = q(K) - q(\check{L}) = q(K^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\check{L}) = s_l(L^+) + s_u(U^+)$. Thus (5.2) holds for the updated quantities.

Next, suppose $t_U^+ = \hat{t}$ at Step 5. Let $\check{U} := \{i \in I_u : \hat{t} \le t_i^u\}$. Then $K = K^+ \cup \check{U}$ with $K^+ = \{i : t_L < t_i^l \text{ and } t_i^u < \hat{t}\}$ and $K^+ \cap \check{U} = \emptyset$ by (5.1) and (3.3), whereas the partition (3.2) yields $L \cup U = N \setminus K$ and $L^+ \cup U^+ = N \setminus K^+$ with $L^+ = L$ and $\check{U} \cap L = \emptyset$, so $U^+ = U \cup \check{U}$ with $U \cap \check{U} = \emptyset$. Further, $\check{U} = \hat{U} \cup \hat{I}_u$ with $\hat{U} \cap \hat{I}_u = \emptyset$ at Step 2.

Combining the preceding relations with (5.2) and the rules of Step 5 gives $p^+ = p(K) - p(\check{U}) = p(K^+)$, $q^+ = q(K) - q(\check{U}) = q(K^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\check{U}) = s_l(L^+) + s_u(U^+)$. Thus (5.2) holds for the updated quantities.

Thus, by induction, (5.2) always holds at Steps 2 and 6.

When $T = \{t_i^l\}_{i\in I_l} \cup \{t_i^u\}_{i\in I_u}$ becomes empty, $I_l = I_u = \emptyset$. Then (3.3) and (5.1) show that (5.2) with $K = M$ reduces to (3.7), so $t_* \in T_*$ by Remark 3.2(c). □

Remark 5.3 An *asymmetric* version of Algorithm 5.1 is obtained by replacing $\hat{L}$ with $\check{L} := \{i \in I_l : t_i^l \leq \hat{t}\}$ at Steps 2 and 4 with $\hat{I}_l$ omitted; alternatively we may replace $\hat{U}$ by $\check{U} := \{i \in I_u : \hat{t} \leq t_i^u\}$, omitting $p(\hat{I}_u)$, etc. In fact both replacements may be used whenever $l < u$ [since (5.3b) with $\hat{L}$, $\hat{U}$ replaced by $\check{L}$, $\check{U}$ only needs $\check{L} \cap \check{U} = \emptyset$].

6 Simplifications for quadratic resource allocation

The quadratic resource allocation (QRA) problem is a special instance of P with $l_i = 0$ and $u_i = \infty$ for all i. In this case Algorithm 4.1 simplifies as follows (cf. Remark 3.4(c)).

Algorithm 6.1 (for QRA: $l_i = 0, u_i = \infty \; \forall i \in N$)

STEP 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$, $I := N$, $T := \{t_i^l\}_{i \in N}$, $p := 0$, $q := 0$, $s := 0$.

STEP 1 (*Breakpoint selection*). Choose a breakpoint $\hat{t}$ in T.

STEP 2 (*Computing* $g(\hat{t})$). Set $\hat{M} := \{i \in I : \hat{t} \leq t_i^l\}$, $\hat{p} := p + p(\hat{M})$, $\hat{q} := q + q(\hat{M})$, $g(\hat{t}) = \hat{p} - \hat{t}\hat{q}$.

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

STEP 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$, $I := \{i \in I : \hat{t} < t_i^l\}$.

STEP 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$, $p := \hat{p}$, $q := \hat{q}$, $I := \{i \in I : t_i^l < \hat{t}\}$.

STEP 6 (*Stopping criterion*). If $T \neq \emptyset$, go to Step 1, else stop with t_* given by (3.9).

In a parallel development, also Algorithm 5.1 may be simplified as follows.

Algorithm 6.2 (for QRA: $l_i = 0, u_i = \infty \; \forall i \in N$)

STEP 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$, $I := N$, $T := \{t_i^l\}_{i \in N}$, $p := p(N)$, $q := q(N)$, $s := 0$.

STEP 1 (*Breakpoint selection*). Choose a breakpoint $\hat{t}$ in T.

STEP 2 (*Computing* $g(\hat{t})$). Set $\hat{L} := \{i \in I : t_i^l < \hat{t}\}$, $\hat{p} := p - p(\hat{L})$, $\hat{q} := q - q(\hat{L})$, $g(\hat{t}) = \hat{p} - \hat{t}\hat{q}$.

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

STEP 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$, set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$, $\hat{I} := \{i \in I : t_i^l = \hat{t}\}$, $p := \hat{p} - p(\hat{I})$, $q := \hat{q} - q(\hat{I})$, $I := \{i \in I : \hat{t} < t_i^l\}$.

STEP 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$, set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$, $I := \{i \in I : t_i^l < \hat{t}\}$.

STEP 6 (*Stopping criterion*). If $T \neq \emptyset$, go to Step 1, else stop with t_* given by (3.9).

Note the complementary features of both algorithms, which also appear in their modifications discussed below.

Remark 6.3 (a) For $\check{M} := \{i \in I : \hat{t} < t_i^l\}$ and $\hat{I} := \{i \in I : t_i^l = \hat{t}\}$, we have $\hat{M} = \check{M} \cup \hat{I}$ with $\check{M} \cap \hat{I} = \emptyset$, and $p(\hat{I}) - \hat{t}q(\hat{I}) = 0$ from $(a_i - t_i^l b_i)/d_i = l_i = 0$ $\forall i \in \hat{I}$; thus $p(\hat{M}) - \hat{t}q(\hat{M}) = p(\check{M}) - \hat{t}q(\check{M})$. Hence $\check{M}$ may replace $\hat{M}$ at Step 2 of Algorithm 6.1, but then Step 5 must set $p := \hat{p} + p(\hat{I})$, $q := \hat{q} + q(\hat{I})$.

(b) In the asymmetric version of Algorithm 6.2 discussed in Remark 5.3, the set $\check{L} := \{i \in I : t_i^l \le \hat{t}\}$ replaces $\hat{L}$ at Step 2, and Step 4 sets $p := \hat{p}$, $q := \hat{q}$.

7 A double-median approach

In the spirit of [6, Sect. 3], we now consider a modification of Algorithm 3.1 in which Steps 1–5 are replaced by a call to the following procedure that may update both t_L and t_U.

Procedure 7.1
STEP P0: Set $\hat{t} := \text{median}\{t_i^l\}_{i \in I}$. If $t_U \le \hat{t}$, go to Step **P4**.
STEP P1: If $g(\hat{t}) = r$, stop with $t_* := \hat{t}$.
STEP P2: If $g(\hat{t}) > r$, set $t_L := \hat{t}$ and exit, else set $t_U := \hat{t}$.
STEP P3: Set $C := \{i \in I : t_i^l, t_i^u \notin (t_L, t_U)\}$. If $|C| \ge \frac{1}{4}|I|$, exit.
STEP P4: Set $\check{t} := \text{median}\{t_i^u\}_{i \in \hat{I}}$, where $\hat{I} := \{i \in I : \hat{t} \le t_i^l\}$.
STEP P5: If $g(\check{t}) = r$, stop with $t_* := \check{t}$.
STEP P6: If $g(\check{t}) > r$, set $t_L := \check{t}$, else set $t_U := \check{t}$.

After t_L, t_U are updated to t_L^+, t_U^+, I and T are updated to I^+ and T^+ via (3.3), (3.4).

Lemma 7.2 *Procedure* 7.1 *either terminates, or finds t_L^+, t_U^+ such that $|I^+| \le \frac{3}{4}|I|$.*

Proof At Step P0, $t_L < \hat{t}$ because $t_L < t_i^l$ $\forall i \in I$ by (3.3). If Step P2 exits with $t_L^+ = \hat{t}$, then $\{i \in I : t_i^l \le \hat{t}\} \subset L^+ \subset I \setminus I^+$; otherwise, t_U is decreased to $\hat{t}$. If Step P3 exits, then $C = I \setminus I^+$. If Step P4 is entered from Step P3, then $\check{t} \in (t_L, t_U)$. Indeed, $\check{t} \le t_L$ would imply $C_M := \{i \in \hat{I} : t_i^u \le \check{t}\} \subset C$ using $t_U = \hat{t}$, with $|C_M| \ge \frac{1}{2}|\hat{I}| \ge \frac{1}{4}|I|$, whereas $t_U \le \check{t}$ would yield $C_U := \{i \in \hat{I} : \check{t} \le t_i^u\} \subset C$ with $|C_U| \ge \frac{1}{2}|\hat{I}| \ge \frac{1}{4}|I|$, contradicting $|C| < \frac{1}{4}|I|$. Also $\check{t} \in (t_L, t_U)$ if Step P4 is entered from Step P0 with $t_U \le \hat{t}$, since by (3.3), $t_U \le \hat{t} \le t_i^l$ $\forall i \in \hat{I}$ implies $t_i^u \in (t_L, t_U)$ $\forall i \in \hat{I}$ and hence $\check{t} \in (t_L, t_U)$. If $t_L^+ = \check{t}$ at Step P6, then $C_M := \{i \in \hat{I} : t_i^u \le \check{t}\} \subset M^+$ from $\hat{I} \subset \{i \in I : t_U^+ \le t_i^l\}$, with $|C_M| \ge \frac{1}{4}|I|$. Otherwise $t_U^+ = \check{t}$ yields $C_U := \{i \in \hat{I} : \check{t} \le t_i^u\} \subset U^+$ with $|C_U| \ge \frac{1}{4}|I|$. In each case $I \setminus I^+$ contains a set of cardinality at least $\frac{1}{4}|I|$; hence $|I^+| \le \frac{3}{4}|I|$. □

Remark 7.3 (a) The exits in Steps P2 and P3 of Procedure 7.1 are intended to save work in finding $\check{t}$ and $g(\check{t})$. Note that $|C|$ is easily determined while computing $g(\hat{t})$. Both exits may be replaced by an exit at Step P4 when $\check{t} \notin (t_L, t_U)$, still ensuring $|I^+| \le \frac{3}{4}|I|$; this version corresponds to the algorithm in [6, Sect. 3].

(b) Procedure 7.1 requires order $|I|$ operations for $g(\hat{t})$ and $g(\check{t})$ computed via (3.5)–(3.8) as in [6, Sect. 3], or as in Algorithms 4.1 and 5.1. As before, the condition $T = \emptyset$ (or equivalently $I = \emptyset$) serves as the stopping criterion. Since $|I|$ is initially n and is reduced by at least a quarter at each iteration, the overall complexity is $O(n)$ as in the single median versions of Algorithms 3.1, 4.1 and 5.1.

8 Removing more breakpoints at each iteration

Consider the following modification of Algorithm 3.1 which removes more breakpoints from the set T as in [7, Algorithm 2.3]. Replace Steps 4 and 5 by

STEP 4' (*Lower breakpoint removal*). If $g(\hat{t}) > r$, find the right adjacent breakpoint $\check{t} := \min\{t \in T : \hat{t} < t\}$; if $\check{t} < \infty$ and $g(\check{t}) > r$, set $t_L := \check{t}$, $T := \{t \in T : \check{t} < t\}$, else set $t_L := \hat{t}$, $t_U := \min\{t_U, \check{t}\}$ and stop with t_* given by (3.1), or (3.9) if $t_U = \infty$.

STEP 5' (*Upper breakpoint removal*). If $g(\hat{t}) < r$, find the left adjacent breakpoint $\check{t} := \max\{t \in T : t < \hat{t}\}$; if $\check{t} > -\infty$ and $g(\check{t}) < r$, set $t_U := \check{t}$, $T := \{t \in T : t < \check{t}\}$, else set $t_L := \max\{t_L, \check{t}\}$, $t_U := \hat{t}$ and stop with t_* given by (3.1), or (3.9) if $t_L = -\infty$.

By (2.6) and (3.5), because $\hat{t}$ and $\check{t}$ are *consecutive* breakpoints, we may compute

$$g(\check{t}) = g(\hat{t}) - (\check{t} - \hat{t})\left[q + q(I_{\hat{t},\check{t}})\right] \quad \text{with} \quad I_{\hat{t},\check{t}} := \left\{i \in I : \hat{t}, \check{t} \in [t_i^u, t_i^l]\right\} \tag{8.1}$$

in order $|I|$ operations. Thus the complexity estimates of Remark 3.3 remain valid. Yet, relative to the original version, this modification will typically remove only one more breakpoint; it is not clear whether this is worth the additional effort in finding $\check{t}$ and $g(\check{t})$. The version of [7, Algorithm 2.3] is less aggressive, setting $T := \{t \in T : \check{t} \le t\}$ in Step 4' and $T := \{t \in T : t \le \check{t}\}$ in Step 5'.

Algorithms 4.1 and 5.1 may be modified similarly, using

$$g(\check{t}) = g(\hat{t}) - (\check{t} - \hat{t}) \begin{cases} [\hat{q} - q(\hat{I}_l)] & \text{if } \hat{t} < \check{t}, \\ [\hat{q} - q(\hat{I}_u)] & \text{if } \check{t} < \hat{t}, \end{cases} \tag{8.2}$$

for $\hat{q}$ available from Step 2. Of course, $\check{t}$ replaces $\hat{t}$ in Steps 4 and 5. More specifically, let $\check{I}_l := \{i \in I_l : t_i^l = \check{t}\}$, $\check{I}_u := \{i \in I_u : t_i^u = \check{t}\}$, $\check{J}_l := \{i \in J_l : t_i^l = \check{t}\}$, $\check{J}_u := \{i \in J_u : t_i^u = \check{t}\}$. In Algorithm 4.1, p, q, s increase by $p(\check{J}_u)$, $q(\check{J}_u)$, $s_l(\check{I}_l)$ in Step 4, and by $p(\check{J}_l)$, $q(\check{J}_l)$, $s_l(\check{I}_u)$ in Step 5, respectively. In Algorithm 5.1, subtract $p(\check{I}_l)$, $q(\check{I}_l)$ from p, q, and add $s_l(\check{I}_l)$ to s in Step 4, and do the same in Step 5 with $\check{I}_l$ replaced by $\check{I}_u$. The derivation of these updates and of (8.2) is quite long, and hence omitted.

9 Relations with other methods

The relations of Algorithm 3.1 and its modifications with the two earliest methods of [6,7] were discussed in Sects. 7, 8. In this section we highlight some features of the remaining $O(n)$ methods of [9,13,18,19,23]. First, we acknowledge that our framework employs several ideas introduced in these works. For instance, Algorithm 3.1 may be regarded as a simplified variant of the method of [23, Sect. 2], Algorithms 6.1 and 6.2 employ the updates of [18] and [9, Sect. 1.2], respectively, whereas Algorithm 5.1 was inspired by that in [13, Sect. 3]. On the other hand, the algorithms of [9,13,18,19,23] employ more complex choices of breakpoint subsets for which the median is found. Although their choices work on most problems, it turns out they may cycle on simple examples. To see this, we first describe some "dangerous" choices. Afterwards, we discuss each algorithm and suggest a "cure", i.e., a convergent modification; to save space, fairly obvious details are omitted.

9.1 Dangerous modifications

Steps 4 and 5 of Algorithm 3.1 reduce the set T *independently* of how we choose $\hat{t}$ in T. The following examples (see Figs. 2, 3) illustrate the need for such reductions when $\hat{t} := \text{median}(T)$ at Step 1. Let $e := (1, \dots, 1) \in \mathbb{R}^n$ denote the unit vector.

Example 9.1 Suppose Steps 4 and 5 of Algorithm 3.1 set $T := T \cap [t_L, t_U]$. For the problem with $n = 3$, $d = b = e$, $a = 0$, $r = -1$, $l = (0, -1, -2)$, $u = 0$, we have $T_* = \{0.5\}$ and $T_0 = \{0, 1, 2, 0, 0, 0\}$, but this version will loop infinitely with $\hat{t} \equiv 0$, $g(\hat{t}) = 0$.

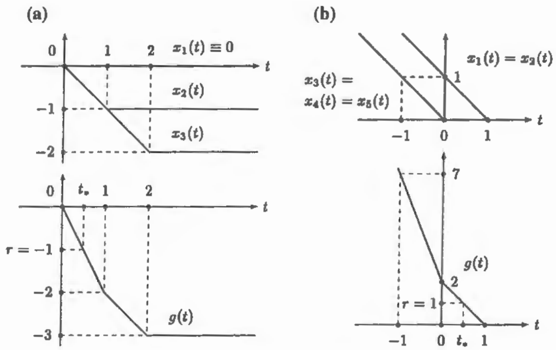

Fig. 2 **a** Illustration of Example 9.1. **b** Illustration of Example 9.2

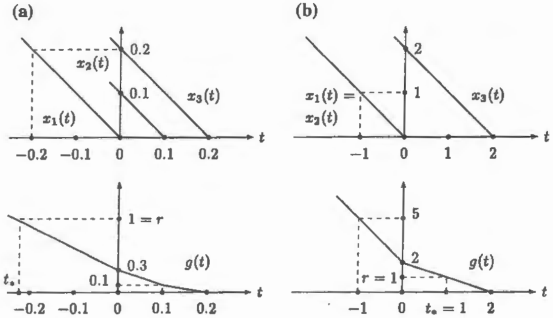

Fig. 3 a Illustration of Example 9.3. b Illustration of Example 9.4

Example 9.2 Consider QRA with $n = 5$, $d = b = e$, $a = (1, 1, 0, 0, 0)$, $r = 1$, $T_* = \{0.5\}$. Algorithm 6.2, starting with $T = \{1, 1, 0, 0, 0\}$, generates $t_L = \hat{t} = 0$, $T = \{1, 1\}$, $I = \{1, 2\}$, then $t_U = \hat{t} = 1$, $T = \emptyset$, terminating with $t_* = 0.5$. Now, suppose Step 5 sets $T := \{t \in T : t \le \hat{t}\}$, $I := \{i \in I : t_i^l \le \hat{t}\}$, and Step 6 stops if $|T| \le 1$. This version loops infinitely with $t_U = \hat{t} = 1$, $T = \{1, 1\}$.

Example 9.3 Consider QRA with $n = 3$, $d = b = e$, $a = (0, 0.1, 0.2)$, $r = 1$, $T_* = \{-\frac{7}{30}\}$. Algorithm 6.1, starting with $T = \{0, 0.1, 0.2\}$, generates $t_U = \hat{t} = 0.1$, $T = \{0\}$, $p = 0.3$, $q = 2$, then $t_U = \hat{t} = 0$, $T = \emptyset$, $p = 0.3$, $q = 3$, terminating with $t_* = -\frac{7}{30}$. Now, suppose that when $\hat{t} = t_m^l$ at Step 1 for some $m \in I$, Step 4 sets $T := \{t \in T : \hat{t} < t\} \cup \{\hat{t}\}$, $I := \{i \in I : \hat{t} < t_i^l\} \cup \{m\}$, Step 5 sets $T := \{t \in T : t < \hat{t}\} \cup \{\hat{t}\}$, $I := \{i \in I : t_i^l < \hat{t}\} \cup \{m\}$, and Step 6 stops if $|T| \le 2$. Then the first iteration terminates with $t_* = -\frac{7}{20}$.

As will be seen, several methods fail on the following simple example.

Example 9.4 Consider QRA with $n = 3$, $d = b = e$, $a = (0, 0, 2)$, $r = 1$, $T_* = \{1\}$. Algorithms 6.1 and 6.2, starting with $T = \{0, 0, 2\}$, generate $t_L = \hat{t} = 0$, $T = \{2\}$, then $t_U = \hat{t} = 2$, $T = \emptyset$, terminating with $t_* = 1$.

9.2 The algorithm of Pardalos and Kovoor

In our notation, the algorithm of [23, Sect. 2], starting with $\tilde{T} := T_0 \cup \{-\infty, \infty\}$, sets $\hat{t} := \mathrm{median}(\tilde{T})$, computes $g(\hat{t})$ via (3.5), sets $t_L := \hat{t}$ if $g(\hat{t}) \ge r$, $t_U := \hat{t}$ if $g(\hat{t}) \le r$, $\tilde{T} := \tilde{T} \cap [t_L, t_U]$, updating p, q, s as in (3.8) until $I = \emptyset$. First, without reducing $\tilde{T}$, it loops on Example 9.1. Second, the updates of (3.8) are *not* valid when $t_L = t_U$; this makes it fail on the following example.

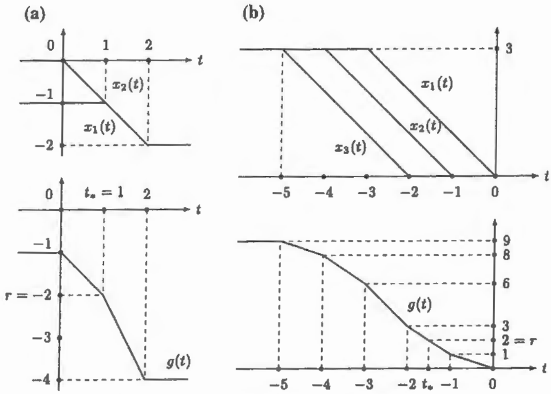

Fig. 4 a Illustration of Example 9.5. b Illustration of Example 9.7

Example 9.5 For $n = 2$, let $d = b = e$, $a = 0$, $r = -2$, $l = (-2, -2)$, $u = (-1, 0)$. Then $T_* = \{1\}$ (see Fig. 4) and $x^* = (-1, -1)$, but the algorithm of [23, Sect. 2] delivers the wrong solution $(-0.5, -0.5)$.

A simple cure is to make the algorithm of [23, Sect. 2] fit the pattern of Algorithm 3.1 by replacing some of its nonstrict inequalities by strict ones.

9.3 The algorithm of Cosares and Hochbaum

In our notation, the algorithm of [9, Sect. 1.2] differs from Algorithm 6.2 in two aspects. First, it employs the modification of Example 9.2; hence it cycles on that example. Second, assumming implicitly that $|\hat{I}| = 1$ in Step 4, it fails on Example 9.4 (producing $t_* = -0.5$). The cure is simple: in Step 3 of Routine Q-Alloc [9, p. 99], replace $a_m/b_m, 1/b_m, a_i \geq \delta$ by $\sum_{i:a_i=a_m} a_i/b_i, \sum_{i:a_i=a_m} 1/b_i$, $a_i > \delta$, and in Step 4, $|L| \geq 2$ by $|L| \geq 1$.

9.4 The algorithm of Maculan and de Paula

In our notation, the algorithm of [19] differs from Algorithm 6.1 in two aspects (note that Step 3 in [19, Sect. 3] should set $S := \{\bar{x}_j | j \in J\}$). First, it employs the modification of Example 9.3, but only stopping in Step 4 if $|T| \leq 2$, or in Step 5 if $|T| \leq 1$; hence it cycles on that example (assuming median$\{0, 0.1\} = 0.1$). Second, its calculation of $g(\hat{t})$ is wrong: it terminates on Example 9.4 with $t_* = -1$. A natural cure is to simplify the modification of Remark 6.3(a) for $d = b = e$, using $t_i^l = a_i$, $p(\check{M}) = \sum_{i \in \check{M}} a_i$, $q(\check{M}) = |\check{M}|$.

9.5 The algorithm of Maculan, Minoux and Plateau

In our notation, the algorithm of [18] differs from Algorithm 6.1 in three aspects (note that p^-, p_1^- should be swapped with q^-, q_1^- in calculating σ in [18, Sect. 3]). First, employing the modification of Example 9.3, it fails on that example (producing $t_* = -\frac{7}{20}$). Second, it fails on instances where $g(\hat{t}) < r$ never occurs, such as Example 9.4 (producing $t_* = -\frac{r}{0}$). Third, for $n \le 2$, it only yields $t_* = -\frac{r}{0}$. The cure is to reorganize its main loop to fit the pattern of Algorithm 6.1, modified as in Remark 6.3(a).

9.6 The algorithm of Hochbaum and Hong

The algorithm of [13, Sect. 3] is close in spirit to the asymmetric version of Algorithm 5.1 of Remark 5.3 (with $\hat{L}$ replaced by $\check{L}$), modified as in Example 9.2; hence it may cycle.

Example 9.6 For $n = 1$, let $d = b = u = e$, $a = 2$, $r = 1$, $l = 0$; then $T_* = \left\{\frac{3}{2}\right\}$. Assuming median$\{-2, -1\} = -2$, the algorithm of [13, Sect. 3] cycles on this example.

Moreover, its updates of p, q, s and the final formula for t_* are wrong.

Example 9.7 For $n = 3$, let $d = b = e$, $a = (0, -1, -2)$, $r = 2$, $l = 0$, $u = 3e$; then $T_* = \left\{-\frac{3}{2}\right\}$ (see Fig. 4). The asymmetric version of Algorithm 5.1, starting with $T = \{0, -1, -2, -3, -4, -5\}$, generates $t_L = \hat{t} = -2$, $T = \{0, -1\}$, then $t_U = \hat{t} = -1$, $T = \emptyset$, terminating with $t_* = -\frac{3}{2}$. The algorithm of [13, Sect. 3] stops with $t_* = 1$ or $t_* = 0$, depending on whether lower or upper medians are chosen.

A natural cure is to simplify the modification of Remark 5.3 (with $\hat{L}$ replaced by $\check{L}$) for $b = e$, $l = 0$.

10 Numerical results

Two versions of Algorithm 3.1 were programmed in Fortran 77 and run on a notebook PC (Pentium M 755 2 GHz, 1.5 GB RAM) under MS Windows XP. The first version computed exact medians of T via the method of [14]. The second version chose $\hat{t}$ in T at random as in Remark 3.3(b). We also give results for the modified Brucker method (Procedure 7.1), the original Brucker method [cf. Remark 7.3(a)], and the modified Calamai–Moré version of Sect. 8 (which behaved like the original version [7, Algorithm 2.3] in our tests).

Our test problems were randomly generated with n ranging between 50,000 and 2,000,000. As in [4, Sect. 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.

Tables 1, 2, 3, 4 and 5 report the average, maximum and minimum run times over the 20 instances for each of the listed problem sizes and classes, as well as overall statistics.

Table 6 reports the final iteration numbers for the tested methods.

As expected, the average run times grew linearly with the problem size.

Algorithm 3.1 with exact medians was faster than the other versions by about 20%.

The relatively good performance of the exact median versions was due to the high efficiency of the median finding routine of [14]. Random median selections performed quite well on average, but exhibited much larger variations in run times.

Table 1 Run times of Algorithm 3.1 with exact medians (in seconds)

n	Uncorrelated			Weakly correlated			Strongly correlated			Overall		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
50,000	0.02	0.08	0.02	0.02	0.03	0.02	0.03	0.05	0.02	0.02	0.08	0.02
100,000	0.05	0.06	0.05	0.05	0.06	0.05	0.05	0.06	0.05	0.05	0.06	0.05
500,000	0.27	0.28	0.25	0.27	0.28	0.26	0.27	0.28	0.26	0.27	0.28	0.25
1,000,000	0.53	0.55	0.51	0.54	0.55	0.51	0.54	0.55	0.52	0.54	0.55	0.51
1,500,000	0.80	0.82	0.76	0.80	0.82	0.77	0.80	0.82	0.77	0.80	0.82	0.76
2,000,000	1.08	1.09	1.02	1.08	1.10	1.02	1.08	1.09	1.03	1.08	1.10	1.02

Table 2 Run times of Algorithm 3.1 with approximate medians

n	Uncorrelated			Weakly correlated			Strongly correlated			Overall		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
50,000	0.03	0.05	0.01	0.03	0.06	0.01	0.03	0.06	0.02	0.03	0.06	0.01
100,000	0.05	0.08	0.03	0.06	0.10	0.04	0.05	0.08	0.04	0.05	0.10	0.03
500,000	0.30	0.38	0.17	0.31	0.42	0.18	0.30	0.50	0.17	0.30	0.50	0.17
1,000,000	0.63	0.97	0.35	0.60	0.77	0.31	0.60	0.95	0.26	0.61	0.97	0.26
1,500,000	0.90	1.35	0.52	0.86	1.26	0.58	0.95	1.44	0.41	0.90	1.44	0.41
2,000,000	1.18	1.77	0.86	1.38	2.10	0.76	1.30	2.13	0.64	1.29	2.13	0.64

Table 3 Run times of the modified Brucker algorithm

n	Uncorrelated			Weakly correlated			Strongly correlated			Overall		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
50,000	0.02	0.05	0.01	0.03	0.06	0.02	0.03	0.07	0.02	0.03	0.07	0.01
100,000	0.06	0.07	0.04	0.06	0.07	0.05	0.07	0.09	0.05	0.06	0.09	0.04
500,000	0.31	0.38	0.23	0.33	0.35	0.24	0.33	0.36	0.24	0.32	0.38	0.23
1,000,000	0.64	0.74	0.47	0.66	0.71	0.47	0.62	0.71	0.49	0.64	0.74	0.47
1,500,000	0.98	1.11	0.73	0.93	1.06	0.73	0.93	1.05	0.72	0.95	1.11	0.72
2,000,000	1.31	1.48	0.96	1.26	1.40	0.99	1.27	1.43	0.97	1.28	1.48	0.96

Table 4 Run times of Brucker's algorithm

n	Uncorrelated			Weakly correlated			Strongly correlated			Overall		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
50,000	0.03	0.03	0.01	0.03	0.06	0.02	0.03	0.06	0.02	0.03	0.06	0.01
100,000	0.06	0.07	0.05	0.06	0.07	0.05	0.07	0.07	0.06	0.06	0.07	0.05
500,000	0.33	0.38	0.24	0.34	0.37	0.28	0.33	0.36	0.26	0.33	0.38	0.24
1,000,000	0.66	0.76	0.49	0.69	0.79	0.52	0.65	0.72	0.52	0.67	0.79	0.49
1,500,000	1.01	1.13	0.85	1.01	1.17	0.84	0.99	1.07	0.80	1.00	1.17	0.80
2,000,000	1.35	1.51	1.10	1.31	1.45	1.00	1.33	1.44	1.08	1.33	1.51	1.00

Table 5 Run times of the Calamai–Moré algorithm

n	Uncorrelated			Weakly correlated			Strongly correlated			Overall		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
50,000	0.02	0.03	0.02	0.03	0.03	0.02	0.03	0.03	0.02	0.03	0.03	0.02
100,000	0.06	0.07	0.05	0.06	0.07	0.06	0.06	0.07	0.06	0.06	0.07	0.05
500,000	0.32	0.34	0.31	0.33	0.34	0.31	0.33	0.34	0.32	0.33	0.34	0.31
1,000,000	0.65	0.67	0.62	0.65	0.66	0.63	0.66	0.67	0.64	0.65	0.67	0.62
1,500,000	0.98	1.00	0.94	0.98	1.00	0.95	0.98	1.00	0.94	0.98	1.00	0.94
2,000,000	1.31	1.34	1.25	1.31	1.33	1.25	1.32	1.33	1.26	1.31	1.34	1.25

Table 6 Iteration numbers for the tested algorithms

Method	n	Uncorrelated			Weakly correlated			Strongly correlated		
		Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
Alg. 3.1 exact	1,000,000	20	21	20	21	21	21	20	21	20
	2,000,000	21	22	21	22	22	22	21	22	21
Alg. 3.1 appr.	1,000,000	30	43	19	29	37	20	27	39	16
	2,000,000	29	40	20	31	41	24	30	43	19
Mod. Brucker	1,000,000	19	25	11	19	23	13	19	25	15
	2,000,000	20	23	16	19	26	13	20	24	15
Orig. Brucker	1,000,000	18	22	11	18	22	13	18	22	12
	2,000,000	19	24	13	19	24	13	19	24	13
Calamai–Moré	1,000,000	18	20	16	19	20	17	18	20	16
	2,000,000	19	21	13	20	21	16	20	21	17

For exact medians, the modified Brucker method was slightly faster than the original Brucker method, and their run times were less stable than those of Algorithm 3.1. The Calamai–Moré method performed similarly to the Brucker variants on average, but its run times were more stable. Relative to Algorithm 3.1, the extra complications of the Brucker and Calamai–Moré variants did not pay in practice.

More extensive numerical tests of Algorithms 3.1, 4.1 and 5.1, and comparisons with variable fixing methods [16] will be given elsewhere [15].

Acknowledgments I would like to thank the Associate Editor and the three anonymous referees for their helpful comments.

References

1. Berman, P., Kovoor, N., Pardalos, P.M.: Algorithms for the least-distance problem. In: Pardalos, P.M. (ed.) Complexity in Numerical Optimization, pp. 33–56. World Scientific, Singapore (1993)
2. Bitran, G.R., Hax, A.C.: Disaggregation and resource allocation using convex knapsack problems with bounded variables. Manage. Sci. **27**, 431–441 (1981)
3. Bretthauer, K.M., Shetty, B.: Quadratic resource allocation with generalized upper bounds. Oper. Res. Lett. **20**, 51–57 (1997)
4. Bretthauer, K.M., Shetty, B., Syam, S.: A branch and bound algorithm for integer quadratic knapsack problems. ORSA J. Comput. **7**, 109–116 (1995)
5. Bretthauer, K.M., Shetty, B., Syam, S.: A projection method for the integer quadratic knapsack problem. J. Oper. Res. Soc. **47**, 457–462 (1996)
6. Brucker, P.: An $O(n)$ algorithm for quadratic knapsack problems. Oper. Res. Lett. **3**, 163–166 (1984)
7. Calamai, P.H., Moré, J.J.: Quasi-Newton updates with bounds. SIAM J. Numer. Anal. **24**, 1434–1441 (1987)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
9. Cosares, S., Hochbaum, D.S.: Strongly polynomial algorithms for the quadratic transportation problem with a fixed number of sources. Math. Oper. Res. **19**, 94–111 (1994)
10. Cottle, R.W., Duvall, S.G., Zikan, K.: A Lagrangean relaxation algorithm for the constrained matrix problem. Naval Res. Logist. Quart. **33**, 55–76 (1986)
11. Held, M., Wolfe, P., Crowder, H.P.: Validation of subgradient optimization. Math. Program. **6**, 62–88 (1974)
12. Helgason, K., Kennington, J., Lall, H.: A polynomially bounded algorithm for a singly constrained quadratic program. Math. Program. **18**, 338–343 (1980)
13. Hochbaum, D.S., Hong, S.P.: About strongly polynomial time algorithms for quadratic optimization over submodular constraints. Math. Program. **69**, 269–309 (1995)
14. Kiwiel, K.C.: On Floyd and Rivest's select algorithm. Theor. Comput. Sci. **347**, 214–238 (2005)
15. Kiwiel, K.C.: Bracketing methods for the continuous quadratic knapsack problem. Technical report. Systems Research Institute, Warsaw (2006) (in preparation)
16. Kiwiel, K.C.: Variable fixing algorithms for the continuous quadratic knapsack problem. Technical report. Systems Research Institute, Warsaw (2006) (in preparation)
17. Luss, H., Gupta, S.K.: Allocation of effort resources among competing activities. Oper. Res. **23**, 360–366 (1975)
18. Maculan, N., Minoux, M., Plateau, G.: An $O(n)$ algorithm for projecting a vector on the intersection of a hyperplane and R^n_+. RAIRO Rech. Opér. **31**, 7–16 (1997)
19. Maculan, N., de Paula, G.G., Jr.: A linear-time median-finding algorithm for projecting a vector on the simplex of R^n. Oper. Res. Lett. **8**, 219–222 (1989)
20. Megiddo, N., Tamir, A.: Linear time algorithms for some separable quadratic programming problems. Oper. Res. Lett. **13**, 203–211 (1993)
21. Michelot, C.: A finite algorithm for finding the projection of a point onto the canonical simplex of R^n. J. Optim. Theory Appl. **50**, 195–200 (1986)
22. Nielsen, S.S., Zenios, S.A.: Massively parallel algorithms for singly constrained convex programs. ORSA J. Comput. **4**, 166–181 (1992)
23. Pardalos, P.M., Kovoor, N.: An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. Math. Program. **46**, 321–328 (1990)
24. Robinson, A.G., Jiang, N., Lerme, C.S.: On the continuous quadratic knapsack problem. Math. Program. **55**, 99–108 (1992)
25. Shetty, B., Muthukrishnan, R.: A parallel projection for the multicommodity network model. J. Oper. Res. Soc. **41**, 837–842 (1990)
26. Ventura, J.A.: Computational development of a Lagrangian dual approach for quadratic networks. Networks **21**, 469–485 (1991)
27. Zipkin, P.H.: Simple ranking methods for allocation of one resource. Manage. Sci. **26**, 34–43 (1980)