Raport Badawczy
Research Report

RB/77/2002

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 2002

Breakpoint searching algorithms for the continuous quadratic knapsack problem*

Krzysztof C. Kiwiel†

December 30, 2002

Abstract

We give several linear time algorithms for the continuous quadratic knapsack problem. We show that, out of the seven existing linear time algorithms, two can be improved, and the remaining five fail on simple counterexamples.

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 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. §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] use medians of breakpoint subsets (see [BKP93, MeT93] for extensions); [PaK90] also proposed an approximate median version with an average-case

*Research supported by the State Committee for Scientific Research under Grant 4T11A00622.

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

performance of $O(n)$. Another class of methods with worst-case performance of $O(n^2)$ [BiH81, Zip80, Mic86, Ven91, RJL92, BSS96] employs variable fixing [LuG75].

This paper focuses on linear time algorithms for P. We introduce a breakpoint searching framework that is conceptually simpler than those in [Bru84, CaM87, MdP89, PaK90, CoH94, HoH95, MMP97]. Among other things, we give modifications of the first two methods [Bru84, CaM87] (with more efficient g-evaluations), and simple counterexamples for the remaining methods [MdP89, PaK90, CoH94, HoH95, MMP97], showing that they may deliver wrong solutions.

The paper is organized as follows. Basic properties of P are reviewed in §2. Our simplest algorithm is introduced in §3 together with the standard g-evaluation of [Bru84, CaM87]. A more refined g-evaluation is derived in §4, and a complementary one in §5 (the latter extends and corrects some ideas in [CoH94, HoH95]). To ease comparisons with related methods, in §6 we state simplifications for quadratic resource allocation. Extensions of the two median approach of [Bru84] and the additional breakpoint removal of [CaM87] are discussed in §§7 and 8, respectively. Section 9 presents our counterexamples for the methods of [MdP89, PaK90, CoH94, HoH95, MMP97]. Finally, preliminary computational results for large-scale problems are reported in §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 $f(x) + t(b^T x - r)$ s.t. $l \le x \le u$)

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

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

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

and the associated multipliers for the constraints $l - x \le 0$ and $x - u \le 0$, 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 [CaM87, Thm 2.1], [HKL80, §2], [NiZ92, §1.2], [PaK90, Thm 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 [Bru84], we assume for simplicity that $b > 0$, because if $b_i = 0$ then x_i may be eliminated ($x_i^* = \min\{\max[l_i, a_i/d_i], u_i\}$), whereas if $b_i < 0$ then 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*).

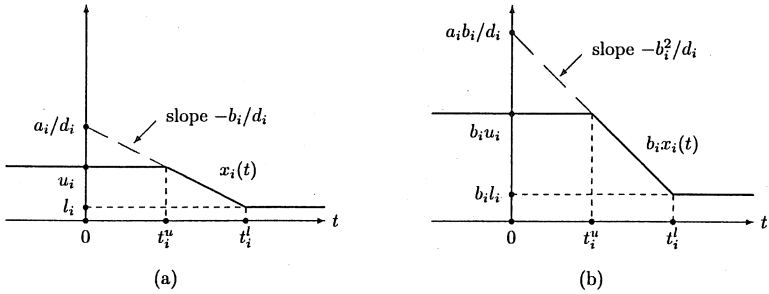

Figure 2.1: (a) Illustration of $x_i(t) := \min\{\max[l_i, (a_i - tb_i)/d_i], u_i\}$. (b) Illustration of $b_ix_i(t) = \min\{\max[b_il_i, (a_ib_i - tb_i^2)/d_i], b_iu_i\}$ (for $b_i > 0$).

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

$$t_i^l := (a_i - l_id_i)/b_i \quad \text{and} \quad t_i^u := (a_i - u_id_i)/b_i, \quad i = 1{:}n, \tag{2.5}$$

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

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

Thus $g(t)$ is a continuous, piecewise linear and *nonincreasing* function of t (cf. Fig. 2.1). Hence the optimal set T_* of (2.4) is an interval (possibly infinite) of the form

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

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 $g(t) = b^T u \ge r$ for all $t \le t_{\min}^u$, $g(t) = b^T l \le r$ for all $t \ge t_{\max}^l$.

3 The breakpoint searching algorithm

As shown in §2, the search for an optimal t_* in T_* can be restricted to the breakpoint interval $[t_{\min}^u, t_{\max}^l]$. The algorithm below generates successive nondecreasing underestimates t_L of t_L^* and nonincreasing overestimates t_U of t_U^* 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 $N := \{1{:}n\}$.

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 $\hat{t}$ in T.

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

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

Step 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then set $t_L := \hat{t}$, $T := \{t \in T : \hat{t} < t\}$.

Step 5 (*Upper breakpoint removal*). If $g(\hat{t}) < r$ then set $t_U := \hat{t}$, $T := \{t \in T : t < \hat{t}\}$.

Step 6 (*Stopping criterion*). If $T \neq \emptyset$ then 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.

Remarks 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 §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 $t_i^u \leq t_i^l$. 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 $T = \{t_i^l\}_{i \in I_l} \cup \{t_i^u\}_{i \in I_u}$; hence $|I| \leq |T|$. Thus, by (2.2), (2.6) and (3.2),

$$g(t) = \sum_{i \in I} b_i x_i(t) + (\,p - tq\,) + s \quad \forall t \in [\,t_L, t_U\,], \tag{3.4}$$

where

$$\sum_{i \in I} b_i x_i(t) = \sum_{i \in I : t \in [t_i^u, t_i^l]} b_i(a_i - tb_i)/d_i + \sum_{i \in I : t_i^l < t} b_i l_i + \sum_{i \in I : t < t_i^u} b_i u_i, \tag{3.5}$$

$$p := \sum_{i \in M} a_i b_i / d_i, \quad q := \sum_{i \in M} b_i^2 / d_i \quad \text{and} \quad s := \sum_{i \in L} b_i l_i + \sum_{i \in U} b_i u_i. \tag{3.6}$$

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

$$\begin{array}{l} \text{for } i \in I \text{ do} \\ \quad \text{if } t_i^l \leq t_L \text{ then } I := I \setminus \{i\},\ s := s + b_i l_i; \\ \quad \text{if } t_U \leq t_i^u \text{ then } I := I \setminus \{i\},\ s := s + b_i u_i; \\ \quad \text{if } t_L, t_U \in [t_i^u, t_i^l] \text{ then } I := I \setminus \{i\},\ p := p + a_i b_i / d_i,\ q := q + b_i^2 / d_i. \end{array} \tag{3.7}$$

This update and the calculation of $g(\hat{t})$ require order $|I| \leq |T|$ operations.

(c) When T becomes empty, then $I = \emptyset$ in (3.4), 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; indeed, $t_U = \infty$ would imply $t_L = t_{\max}^l$ and $g(t_L) = b^T l \leq r$, contradicting $g(t_L) > r$; similarly $t_L = -\infty$ would yield $t_U = t_{\min}^u$ and $g(t_U) = b^T u \geq r$, another contradiction. Alternatively, (3.4) with $I = \emptyset$ shows that (3.1) is equivalent to

$$t_* := (p + s - r)/q. \tag{3.8}$$

(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.

Remarks 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., [AHU74, §3.6], [CLR90, §10.3], [BFP+72, SPP76]. Thus the complexity of each iteration is $O(|T|)$. Since $|T|$ is originally $2n$ and is halved at each iteration, the algorithm makes $O(\log n)$ iterations in time $O(n)$.

(b) As suggested by [PaK90], 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 [AHU74, Thm 3.11], [CLR90, §10.2].

We now briefly describe several useful modifications.

Remarks 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 (3.8) if $T = \emptyset$.

(b) If the set of fixed variables $L^= := \{i : l_i = u_i\}$ is nonempty, at Step 0 we may set $I := N \setminus L^=$, $T := \{t^l_i, t^u_i\}_{i \in I}$, with L replaced by $L \cup L^=$ in (3.2) and (3.6) and M, U, I modified accordingly, terminating with t_* given by (3.8) 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 $u_i = \infty$. Starting with $t_L := -\infty$, $t_U := \infty$ (or $t_L := t^u_{\min}$, $t_U := t^l_{\max}$ as in (a)), 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 (3.8) if $I = \emptyset$. Thus infinite breakpoints are effectively ignored.

4 More refined updates

In a simple implementation based on (3.4)–(3.7), certain sums of (3.5) are repeated in (3.7). We now give a more refined version of Algorithm 3.1 that eliminates these redundancies.

Our refinement consists in using the following partition of I (cf. (3.3))

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

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

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, left and right 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

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

Algorithm 4.1.

Step 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$ (or $t_L := t^u_{\min}$, $t_U := t^l_{\max}$ as in Rems. 3.4(a,c)), $T := \{t^l_i\}_{i \in J_m \cup J_l} \cup \{t^u_i\}_{i \in J_m \cup J_u}$ 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). If $I = \emptyset$, stop with t_* given by (3.8).

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

Step 2 (*Computing* $g(\hat{t})$). Set $\hat{M}_m := \{i \in J_m : t_i^u \le \hat{t} \le t_i^l\}$, $\hat{M}_l := \{i \in J_l : \hat{t} \le t_i^l\}$, $\hat{M}_u := \{i \in J_u : t_i^u \le \hat{t}\}$, $\hat{L} := \{i \in I_l : t_i^l < \hat{t}\}$, $\hat{U} := \{i \in I_u : \hat{t} < t_i^u\}$, $\hat{p} := p + p(\hat{M}_m) + p(\hat{M}_l) + p(\hat{M}_u)$, $\hat{q} := q + q(\hat{M}_m) + q(\hat{M}_l) + q(\hat{M}_u)$, $\hat{s} := s + s_l(\hat{L}) + s_u(\hat{U})$, $g(\hat{t}) = (\hat{p} - \hat{t}\hat{q}) + \hat{s}$.

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

Step 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then 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 $g(\hat{t}) < r$ then 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 $T \neq \emptyset$ then go to Step 1, else stop with t_* given by (3.8).

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 (3.4)–(3.5); however, relative to (3.7), 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 Remarks 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 (3.6) holds (this is true initially; cf. Step 0). Then (3.3) and (4.1) 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 $\hat{U} = \{i \in I : \hat{t} < t_i^u\}$ (use $t_L \le \hat{t} \le t_U$) form a partition of I. Hence (3.5) and (4.2) yield

$$\begin{aligned}\sum_{i\in I} b_i x_i(\hat{t}) &= p(\hat{M}) - \hat{t}q(\hat{M}) + s_l(\hat{L}) + s_u(\hat{U}) \\ &= p(\hat{M}_m) + p(\hat{M}_l) + p(\hat{M}_u) - \hat{t}\left[q(\hat{M}_m) + q(\hat{M}_l) + q(\hat{M}_u)\right] + s_l(\hat{L}) + s_u(\hat{U}).\end{aligned}$$

Combining this with (3.4) and (3.6) shows that Step 2 computes $g(\hat{t})$ correctly.

Thus, as long as (3.6) holds, Algorithm 4.1 may be identified with Algorithm 3.1. We now show that (3.6) 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 doesn't 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 $\breve{L} := \{i \in I : t_i^l \le \hat{t}\}$ ($t_L^+ = \hat{t}$), with $\breve{L} = \{i \in I_l : t_i^l \le \hat{t}\}$ (using $\hat{t} < t_U$ in (3.3)). Thus $M^+ = M \cup \hat{M}_u$ with $M \cap \hat{M}_u = \emptyset$, $L^+ = L \cup \breve{L}$ with $L \cap \breve{L} = \emptyset$, $U^+ = U$. Further, $\breve{L} = \hat{L} \cup \hat{I}_l$ with $\hat{L} \cap \hat{I}_l = \emptyset$. Combining the preceding relations with (3.6) and (4.2) gives $p^+ = p(M) + p(\hat{M}_u) = p(M^+)$, $q^+ = q(M) + q(\hat{M}_u) = q(M^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\breve{L}) = s_l(L^+) + s_u(U^+)$. Thus (3.6) holds for the updated quantities.

Next, suppose $t_U^+ = \hat{t}$ at Step 5. Since $t_U^+ \le t_U$ and t_L doesn't 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 $\breve{U} := \{i \in I : \hat{t} \le t_i^u\}$ ($t_U^+ = \hat{t}$), with $\breve{U} = \{i \in I_u : \hat{t} \le t_i^u\}$ (using $t_L < \hat{t}$ in (3.3)). Thus $M^+ = M \cup \hat{M}_l$ with $M \cap \hat{M}_l = \emptyset$, $U^+ = U \cup \breve{U}$ with $U \cap \breve{U} = \emptyset$, $L^+ = L$. Further, $\breve{U} = \hat{U} \cup \hat{I}_u$ with $\hat{U} \cap \hat{I}_u = \emptyset$. Combining the preceding relations with (3.6) and (4.2) gives $p^+ = p(M) + p(\hat{M}_l) = p(M^+)$, $q^+ = q(M) + q(\hat{M}_l) = q(M^+)$, $s^+ = s_l(L) + s_u(U) + s_u(\breve{U}) = s_l(L^+) + s_u(U^+)$. Thus (3.6) holds for the updated quantities.

It follows by induction that (3.6) always holds at Steps 2 and 6.

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

5 Decrementing 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))

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

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

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

decrementing p and q when I_M shrinks.

Algorithm 5.1.

Step 0 (*Initiation*). Set $t_L := -\infty$, $t_U := \infty$ (or $t_L := t_{\min}^u$, $t_U := t_{\max}^l$ as in Rems. 3.4(a,c)), $T := \{t_i^l\}_{i \in I_l} \cup \{t_i^u\}_{i \in I_u}$ 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). If $I = \emptyset$ then stop with t_* given by (3.8).

Step 1 (*Breakpoint selection*). Choose $\hat{t}$ 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 $g(\hat{t}) = r$ then stop with $t_* := \hat{t}$.

Step 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then 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 $g(\hat{t}) < r$ then 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 $T \neq \emptyset$ then go to Step 1, else stop with t_* given by (3.8).

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 Thm 4.2). Hence again the efficiency estimates of Remarks 3.3 remain valid, and we only need to 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

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

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

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

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 $\breve{L} := \{i \in I_l : t_i^l \le \hat{t}\}$. Then $I_M = I_M^+ \cup \breve{L}$ with $I_M^+ = \{i : \hat{t} < t_i^l \text{ and } t_i^u < t_U\}$ and $I_M^+ \cap \breve{L} = \emptyset$ by (5.1) and (3.3), whereas the partition (3.2) yields $L \cup U = N \setminus I_M$ and $L^+ \cup U^+ = N \setminus I_M^+$ with $U^+ = U$ and $\breve{L} \cap U = \emptyset$, so $L^+ = L \cup \breve{L}$ with $L \cap \breve{L} = \emptyset$. Further, $\breve{L} = \hat{L} \cup \hat{I}_l$ 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 $p^+ = p(I_M) - p(\breve{L}) = p(I_M^+)$, $q^+ = q(I_M) - q(\breve{L}) = q(I_M^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\breve{L}) = s_l(L^+) + s_u(U^+)$. Thus (5.2) holds for the updated quantities.

Next, suppose $t_U^+ = \hat{t}$ at Step 5. Let $\breve{U} := \{i \in I_u : \hat{t} \le t_i^u\}$. Then $I_M = I_M^+ \cup \breve{U}$ with $I_M^+ = \{i : t_L < t_i^l \text{ and } t_i^u < \hat{t}\}$ and $I_M^+ \cap \breve{U} = \emptyset$ by (5.1) and (3.3), whereas the partition (3.2) yields $L \cup U = N \setminus I_M$ and $L^+ \cup U^+ = N \setminus I_M^+$ with $L^+ = L$ and $\breve{U} \cap L = \emptyset$, so $U^+ = U \cup \breve{U}$ with $U \cap \breve{U} = \emptyset$. Further, $\breve{U} = \hat{U} \cup \hat{I}_u$ 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 $p^+ = p(I_M) - p(\breve{U}) = p(I_M^+)$, $q^+ = q(I_M) - q(\breve{U}) = q(I_M^+)$, $s^+ = s_l(L) + s_u(U) + s_l(\breve{U}) = s_l(L^+) + s_u(U^+)$. 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 $I_M = M$ reduces to (3.6), 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 $\breve{L} := \{i \in I_l : t_i^l \le \hat{t}\}$ at Steps 2 and 4 with $\hat{I}_l$ omitted; alternatively we may replace $\hat{U}$ by $\breve{U} := \{i \in I_u : \hat{t} \le t_i^u\}$, 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 $\breve{L}$, $\breve{U}$ only needs $\breve{L} \cap \breve{U} = \emptyset$).

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. Rem. 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 $\hat{t}$ 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 $g(\hat{t}) = r$ then stop with $t_* := \hat{t}$.

Step 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then 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 $g(\hat{t}) < r$ then 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 $T \neq \emptyset$ then go to Step 1, else stop with t_* given by (3.8).

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 $\hat{t}$ 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 $g(\hat{t}) = r$ then stop with $t_* := \hat{t}$.

Step 4 (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then 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 $g(\hat{t}) < r$ then 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 $T \neq \emptyset$ then go to Step 1, else stop with t_* given by (3.8).

Note the complementary features of both algorithms, and their modifications discussed below.

Remarks 6.3. (a) For $\breve{M} := \{i \in I : \hat{t} < t_i^l\}$ and $\hat{I} := \{i \in I : t_i^l = \hat{t}\}$, we have $\hat{M} = \breve{M} \cup \hat{I}$ with $\breve{M} \cap \hat{I} = \emptyset$, 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 $p(\hat{M}) - \hat{t}q(\hat{M}) = p(\breve{M}) - \hat{t}q(\breve{M})$. Hence $\breve{M}$ 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 (cf. Rem. 5.3), $\breve{L} := \{i \in I : t_i^l \leq \hat{t}\}$ 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 [Bru84, §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 0: Set $\hat{t} := \text{median}\{t_i^l\}_{i\in I}$. If $t_U \le \hat{t}$ then go to Step 4.

Step 1: If $g(\hat{t}) = r$ then stop with $t_* := \hat{t}$.

Step 2: If $g(\hat{t}) > r$ then set $t_L := \hat{t}$ and exit, else set $t_U := \hat{t}$.

Step 3: Set $C := \{i \in I : t_i^l, t_i^u \notin (t_L, t_U)\}$. If $|C| \ge \frac{1}{4}|I|$ then exit.

Step 4: 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 5: If $g(\check{t}) = r$ then stop with $t_* := \check{t}$.

Step 6: If $g(\check{t}) > r$ then set $t_L := \check{t}$, else set $t_U := \check{t}$.

After t_L, t_U are updated to t_L^+, t_U^+, I is updated to I^+ via (3.3).

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 0, $t_L < \hat{t}$ because $t_L < t_i^l$ $\forall i \in I$ by (3.2). If Step 2 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 3 exits then $C = I \setminus I^+$. If Step 4 is entered from Step 3, 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 4 is entered from Step 1 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 6 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 $|I^+| \le \frac{3}{4}|I|$. □

Remarks 7.3. (a) The exits in Steps 2 and 3 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 4 when $\check{t} \notin (t_L, t_U)$, still ensuring $|I^+| \le \frac{3}{4}|I|$; this version corresponds to the algorithm in [Bru84, §3].

(b) Procedure 7.1 requires order $|I|$ operations for $g(\hat{t})$ and $g(\check{t})$ computed via (3.4)–(3.7) as in [Bru84, §3], or as in Algorithms 4.1 and 5.1. Of course, $I = \emptyset$ 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 [CaM87, Alg. 2.3]. Replace Steps 4 and 5 by

Step 4' (*Lower breakpoint removal*). If $g(\hat{t}) > r$ then find the right adjacent breakpoint $\check{t} := \min\{t \in T : \hat{t} < t\}$; if $\check{t} < \infty$ and $g(\check{t}) > r$ then 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 (3.8) if $t_U = \infty$.

Step 5' (*Upper breakpoint removal*). If $g(\hat{t}) < r$ then find the left adjacent breakpoint $\check{t} := \max\{t \in T : t < \hat{t}\}$; if $\check{t} > -\infty$ and $g(\check{t}) < r$ then 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 (3.8) if $t_L = -\infty$.

By (2.6) and (3.4), because $\hat{t}$ and $\check{t}$ are *consecutive* breakpoints, we may compute

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

in order $|I|$ operations. Thus the complexity estimates of Remarks 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 [CaM87, Alg. 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

9.1 Dangerous modifications

Steps 4 and 5 of Algorithm 3.1 reduce T *independently* of how $\hat{t} \in T$ is chosen. The following examples (cf. Figs. 9.1–9.2) illustrate the need for such reductions when $\hat{t} := \text{median}(T)$ at Step 1. Let $e := (1, \ldots, 1) \in \mathbb{R}^n$ 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 $\hat{t} \equiv 0$, $g(\hat{t}) = 0$.

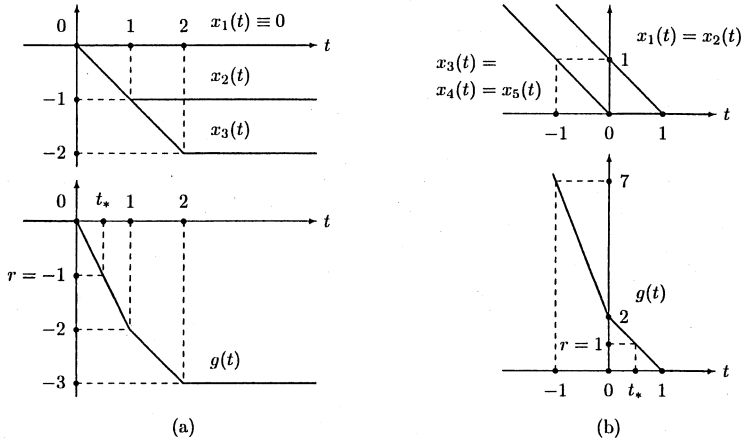

Figure 9.1: (a) Illustration of Example 9.1. (b) Illustration of Example 9.2.

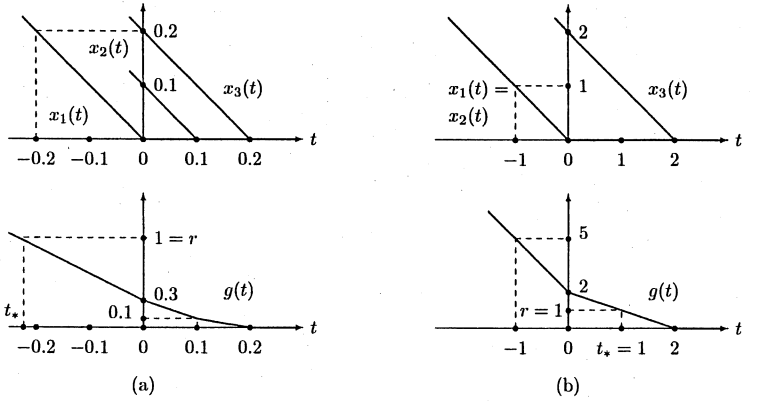

Figure 9.2: (a) Illustration of Example 9.3. (b) Illustration of Example 9.4.

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 [PaK90, §2], starting with $\tilde{T} := T_0 \cup \{-\infty, \infty\}$, sets $\hat{t} := \text{median}(\tilde{T})$, computes $g(\hat{t})$ via (3.4), 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 (3.7) until $I = \emptyset$. First, without reducing $\tilde{T}$, it loops on Example 9.1. Second, the updates of (3.7) are *not* valid when $t_L = t_U$; this makes it fail on the following example.

Example 9.5. For $n = 2$, let $d = b = e$, $a = 0$, $r = -2$, $l = (-2,-2)$, $u = (-1,0)$. Then $T_* = \{1\}$ (cf. Fig. 9.3) and $x^* = (-1,-1)$, but the algorithm of [PaK90, §2] delivers the wrong solution $(-0.5, -0.5)$.

9.3 The algorithm of Cosares and Hochbaum

In our notation, the algorithm of [CoH94, §1.2] differs from Algorithm 6.2 in two (crucial) aspects. First, assumming implicitly that $|\hat{I}| = 1$ in Step 4, it fails on Example 9.4 (producing $t_* = -0.5$). Second, it employs the modification of Example 9.2; hence it cycles on that example.

9.4 The algorithm of Maculan and de Paula

In our notation, the algorithm of [MdP89] differs from Algorithm 6.1 in two aspects (note that Step 3 in [MdP89, §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 $|T| \le 2$, or in Step 5 if $|T| \le 1$; hence it cycles on that example (assuming $\text{median}\{0, 0.1\} = 0.1$). Second, its calculation of $g(\hat{t})$ is wrong: it terminates on Example 9.4 with $t_* = -1$.

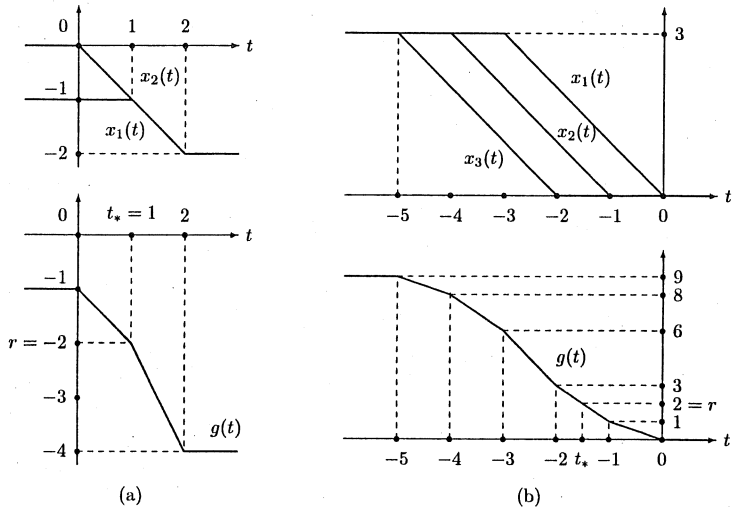

Figure 9.3: (a) Illustration of Example 9.5. (b) Illustration of Example 9.6.

9.5 The algorithm of Maculan, Minoux and Plateau

In our notation, the algorithm of [MMP97] differs from Algorithm 6.1 in three aspects (note that p^-, p_1^- should be swapped with q^-, q_1^- in calculating σ in [MMP97, §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 $n \leq 2$, it only yields $t_* = -\frac{r}{0}$.

9.6 The algorithm of Hochbaum and Hong

The algorithm of [HoH95, §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. However, its updates of p, q, s and the final formula for t_* are wrong; hence it fails on the following example.

Example 9.6. For $n = 3$, let $d = b = e$, $a = (0, -1, -2)$, $r = 2$, $l = 0$, $u = 3e$; then $T_* = \{-\frac{3}{2}\}$ (cf. Fig. 9.3). 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 [HoH95, §3] stops with $t_* = 1$ or $t_* = 0$, depending on how medians are chosen.

Table 10.1: Average, maximum and minimum run times in seconds for uncorrelated, weakly correlated and strongly correlated problems with exact medians.

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.22	0.28	0.16	0.21	0.22	0.16	0.21	0.33	0.16	0.21	0.33	0.16
100000	0.42	0.49	0.38	0.39	0.44	0.38	0.40	0.50	0.33	0.40	0.50	0.33
500000	1.79	1.87	1.75	1.81	1.87	1.75	1.82	1.92	1.75	1.81	1.92	1.75
1000000	3.57	3.79	3.40	3.59	3.84	3.40	3.58	3.85	3.40	3.58	3.85	3.40
1500000	5.35	5.66	5.11	5.39	5.72	5.16	5.37	5.76	5.11	5.37	5.76	5.11
2000000	7.24	9.23	6.81	7.23	7.85	6.86	7.17	7.75	6.86	7.22	9.23	6.81

Table 10.2: Average, maximum and minimum run times in seconds for uncorrelated, weakly correlated and strongly correlated problems with approximate medians.

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.29	0.39	0.16	0.26	0.39	0.16	0.24	0.44	0.11	0.26	0.44	0.11
100000	0.45	0.82	0.33	0.45	0.77	0.27	0.45	0.77	0.28	0.45	0.82	0.27
500000	1.60	2.20	0.99	2.04	3.40	0.99	1.96	3.07	1.21	1.87	3.40	0.99
1000000	3.82	5.83	1.98	3.83	6.26	1.65	3.55	4.94	1.70	3.73	6.26	1.65
1500000	5.47	8.57	3.08	5.47	7.74	2.80	5.91	9.77	2.42	5.62	9.77	2.42
2000000	7.20	14.88	3.73	7.24	11.21	3.57	7.47	11.64	3.90	7.30	14.88	3.57

10 Numerical results

Two versions of Algorithm 4.1 were programmed in Fortran 77 and run on a notebook PC (Pentium II 400 MHz, 256 MB RAM) under MS Windows 98. The first version computed exact medians of T via subroutine dsel.f of Lin and Moré (available as part of the incomplete Cholesky factorization code ICFS [LiM99] from www.mcs.anl.gov/~more/icfs). The second version chose $\hat{t}$ in T at random (cf. Rem. 3.3).

Our test problems were randomly generated with n ranging between 50000 and 2000000 (to avoid memory swapping). 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.

Tables 10.1 and 10.2 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. The average run times grow linearly with the problem size. The relatively good performance of the exact median version is due to the high efficiency of dsel.f.

More extensive numerical tests and comparisons with variable fixing methods [Kiw02] are given in [Kiw03].

Acknowledgment. I would like to thank Jorge Moré for providing subroutine dsel.f.

References

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BFP+72] M. R. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, *Time bounds for selection*, J. Comput. System Sci. **7** (1972) 448–461.

[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.

[BKP93] P. Berman, N. Kovoor and P. M. Pardalos, *Algorithms for the least-distance problem*, in Complexity in Numerical Optimization, P. M. Pardalos, ed., World Scientific, Singapore, 1993, pp. 33–56.

[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.

[CLR90] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[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.

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

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

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

[Kiw02] K. C. Kiwiel, *Variable fixing algorithms for the continuous quadratic knapsack problem*, Tech. report, Systems Research Institute, Warsaw, 2002.

[Kiw03] ———, *Bracketing methods for the continuous quadratic knapsack problem*, Tech. report, Systems Research Institute, Warsaw, 2003.

[LiM99] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Statist. Comput. **21** (1999) 24–45.

[LuG75] H. Luss and S. K. Gupta, *Allocation of effort resources among competing activities*, Oper. Res. **23** (1975) 360–366.

[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.

[MeT93] N. Megiddo and A. Tamir, *Linear time algorithms for some separable quadratic programming problems*, Oper. Res. Lett. **13** (1993) 203–211.

[Mic86] C. Michelot, *A finite algorithm for finding the projection of a point onto the canonical simplex of R^n*, J. Optim. Theory Appl. **50** (1986) 195–200.

[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.

[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. Programming **46** (1990) 321–328.

[RJL92] A. G. Robinson, N. Jiang and C. S. Lerme, *On the continuous quadratic knapsack problem*, Math. Programming **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.

[SPP76] A. Schönhage, M. Paterson and N. Pippenger, *Finding the median*, J. Comput. System Sci. **13** (1976) 184–199.

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

[Zip80] P. H. Zipkin, *Simple ranking methods for allocation of one resource*, Management Sci. **26** (1980) 34–43.