Raport Badawczy
Research Report

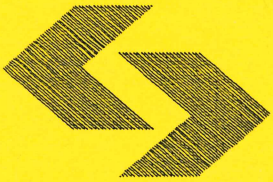
RB/78/2002

Variable Fixing 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

Variable fixing algorithms for the continuous quadratic knapsack problem*

Krzysztof C. Kiwiel†

December 30, 2002

Abstract

We study several variations of the Bitran-Hax method for the continuous quadratic knapsack problem. We close the gaps in the convergence analysis of several existing methods, and provide more efficient versions. Computational results are reported for problems with up to two million variables solved on a notebook PC.

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

1 Introduction

The continuous quadratic knapsack problem is defined by

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

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

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

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

*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)

The variable fixing methods of [BiH79, BiH81, Sho79, Mic86, Ven91, RJL92, BSS96], stemming from [LuG75], determine at each iteration the optimal value of at least one variable; such variables are fixed and hence effectively removed for the next iteration. Although these methods have worst-case performance of $O(n^2)$, they may be competitive in practice [Ven91, RJL92], since they don't need sorting or median calculations.

The first aim of this paper is to clarify certain convergence issues of the variable fixing methods. Only the methods of [Sho79, Mic86] for a special case of P (cf. §5.8) have full proofs of convergence. We show that the algorithms of [RJL92, BSS96] fail on a simple counterexample (cf. §5.7). The method of [Ven91] relies implicitly on the convergence framework of [BiH81] ($\approx$ [BiH79]). However, the proof of the main convergence result of [BiH81, Thm 3] has a gap (cf. §5.5); we show how to fill this gap in the case of P (in the more general setting of [BiH81] where f is merely separable and convex, our proof technique could close the gap when f is strictly convex).

Second, we provide more efficient versions of the variable fixing methods. This is quite suprising, since the methods of [Sho79, Mic86, Ven91, RJL92, BSS96], as well as ours, may be derived from [BiH81] by replacing certain nonstrict inequalities by strict ones and using slightly different stopping criteria (these tight relationships have not been noticed so far). Yet in practice such "tiny" differences can be significant (cf. Ex. 5.4). We also discuss updating techniques which reduce work per iteration.

Third, we show how suitable modifications of the variable fixing methods may find the Lagrange multipliers of P; this is useful in certain applications [BrS97].

The paper is organized as follows. Basic properties of P are reviewed in §2. In §3 we introduce a symmetric version of the method of [BiH81]. Its convergence is established in §4. Various modifications and relations with other methods are discussed in §5. Finally, computational results for large-scale problems are reported in §6.

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. $l \le x \le u$)

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

(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*).

By (2.1)–(2.2) and our assumption $b > 0$, each $x_i(t)$ and $g(t)$ are continuous, piecewise linear and *nonincreasing* functions of t. Hence the set T_* of (2.4) has the form

$$T_* = [\, t_L^*, t_U^* \,] \cap \mathbb{R} \quad \text{with} \quad t_L^* := \inf\{\, t : g(t) = r \,\}, \quad t_U^* := \sup\{\, t : g(t) = r \,\}, \tag{2.5}$$

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 $t_U^* < t$. Further, since $g(t)$ and $x(t)$ are nonincreasing, and $x^* = x(t_*)$ for any $t_* \in T_*$, we have the following useful property (implicit in [BiH81] and explicit in [Ven91, Thm 6]).

Fact 2.2. *Let $\hat{t} \in \mathbb{R}$, $\hat{I}^l := \{i : x_i(\hat{t}) = l_i\}$, $\hat{I}^u := \{i : x_i(\hat{t}) = u_i\}$. Then:*
(a) *If $g(\hat{t}) \geq r$, then $x_i(t) = l_i = x_i^*$ for all $t \geq \hat{t}$ and $i \in \hat{I}^l$.*
(b) *If $g(\hat{t}) \leq r$, then $x_i(t) = u_i = x_i^*$ for all $t \leq \hat{t}$ and $i \in \hat{I}^u$.*

3 The variable fixing algorithm

At each iteration k, our algorithm partitions the variables as $x = (x_{I_k}, x_{L_k}, x_{U_k})$, where (I_k, L_k, U_k) is a partition of $N := \{1{:}n\}$ such that $x_{L_k}^* = l_{L_k}$, $x_{U_k}^* = u_{U_k}$; thus, fixing $x_{L_k} = l_{L_k}$, $x_{U_k} = u_{U_k}$, we only need to consider the remaining *free* variables x_{I_k}.

Algorithm 3.1.
Step 0 (*Initiation*). Set $I_1 := N$, $L_1 := U_1 := \emptyset$, $k := 1$.

Step 1 (*Restricted subproblem solution*). Find the *restricted minimizer*

$$x^k := \arg\min \left\{\, f(x) : b^T x = r, x_{L_k} = l_{L_k}, x_{U_k} = u_{U_k} \,\right\}. \tag{3.1}$$

Step 2 (*Feasibility check*). Compute the *infeasibility indicators*

$$\nabla_k := \sum_{i \in I_k^l} b_i(l_i - x_i^k), \quad \text{where} \quad I_k^l := \left\{\, i \in I_k : x_i^k \leq l_i \,\right\}, \tag{3.2a}$$

$$\Delta_k := \sum_{i \in I_k^u} b_i(x_i^k - u_i), \quad \text{where} \quad I_k^u := \left\{\, i \in I_k : x_i^k \geq u_i \,\right\}. \tag{3.2b}$$

Step 3 (*Stopping criterion*). If $\nabla_k = \Delta_k$ then *reset* $x_{I_k^l}^k := l_{I_k^l}$, $x_{I_k^u}^k := u_{I_k^u}$ and stop.

Step 4 (*Variable fixing*). If $\nabla_k > \Delta_k$ then set $I_{k+1} := I_k \setminus I_k^l$, $L_{k+1} := L_k \cup I_k^l$, $U_{k+1} := U_k$; if $\nabla_k < \Delta_k$ then set $I_{k+1} := I_k \setminus I_k^u$, $L_{k+1} := L_k$, $U_{k+1} := U_k \cup I_k^u$.

Step 5 (*Loop*). Increase k by 1 and go to Step 1.

At Step 1, $x^k_{L_k} = l_{L_k}$, $x^k_{U_k} = u_{U_k}$. The remaining components may be computed as

$$x^k_i = (a_i - t_k b_i)/d_i, \quad i \in I_k \tag{3.3}$$

(by the form of f in (1.1)), where t_k is the Lagrange multiplier of (3.1) given by

$$t_k := \left(\sum_{i \in I_k} a_i b_i / d_i - r_k \right) / \sum_{i \in I_k} b_i^2 / d_i \quad \text{with} \quad r_k := r - \sum_{i \in L_k} b_i l_i - \sum_{i \in U_k} b_i u_i; \tag{3.4}$$

in other words, t_k is the Lagrange multiplier of the *reduced subproblem*

$$x^k_{I_k} = \arg\min \left\{ \textstyle\sum_{i \in I_k} \left(\frac{1}{2} d_i x_i^2 - a_i x_i \right) : b^T_{I_k} x_{I_k} = r_k \right\}. \tag{3.5}$$

4 Convergence of the variable fixing algorithm

Since each iteration reduces the set I_k, Algorithm 3.1 is finite. However, before showing that the final $x^k = x^*$, we must prove that the algorithm is well defined, i.e., $I_k \neq \emptyset$ at Step 1 for all k (this condition is assumed in [BiH81, §2], and *implicitly* in [RJL92]).

Consider the following estimates of t^*_L and t^*_U in (2.5):

$$t^k_L := \sup \{ t_j : \nabla_j \geq \Delta_j, j \leq k \} \quad \text{and} \quad t^k_U := \inf \{ t_j : \nabla_j \leq \Delta_j, j \leq k \}, \tag{4.1}$$

with $t^0_L := -\infty$, $t^0_U := \infty$. Define the reduced constraint value and its linearization

$$g_k(t) := b^T_{I_k} x_{I_k}(t) \quad \text{and} \quad \hat{g}_k(t) := b^T_{I_k} \hat{x}_{I_k}(t) \quad \text{with} \quad \hat{x}_i(t) := (a_i - t b_i)/d_i, \tag{4.2}$$

so that $x^k_{I_k} = \hat{x}_{I_k}(t_k)$, $\hat{g}_k(t_k) = r_k$; cf. (3.3)–(3.5). We shall show that at Step 2

$$x_{I_k}(t^{k-1}_L) = \min \left\{ u_{I_k}, \hat{x}_{I_k}(t^{k-1}_L) \right\} \quad \text{and} \quad x_{I_k}(t^{k-1}_U) = \max \left\{ l_{I_k}, \hat{x}_{I_k}(t^{k-1}_U) \right\}, \tag{4.3}$$

$$g_k(t^{k-1}_L) \geq r_k \geq g_k(t^{k-1}_U), \tag{4.4}$$

$$x_{L_k}(t^{k-1}_L) = l_{L_k} = x^*_{L_k} \quad \text{and} \quad x_{U_k}(t^{k-1}_U) = u_{U_k} = x^*_{U_k}, \tag{4.5}$$

$$x_{L_k}(t_k) = l_{L_k} = x^*_{L_k} \quad \text{and} \quad x_{U_k}(t_k) = u_{U_k} = x^*_{U_k}. \tag{4.6}$$

Lemma 4.1. *Suppose $I_k \neq \emptyset$ and (4.3)–(4.5) hold at Step 2 for some k. Then:*

(a) $t^{k-1}_L \leq t_k \leq t^{k-1}_U$.

(b) *Condition (4.6) holds.*

(c) $g(t_k) - r = \nabla_k - \Delta_k$.

(d) *If $\nabla_k = \Delta_k$, then $t_k \in T_*$ and $x^k = x^*$ after the reset of Step 3.*

(e) *If $\nabla_k > \Delta_k$, then (4.3)–(4.5) hold for k increased by 1, and $I^l_k \neq I_k$ at Step 4.*

(f) *If $\nabla_k < \Delta_k$, then (4.3)–(4.5) hold for k increased by 1, and $I^u_k \neq I_k$ at Step 4.*

Proof. (a) $\hat{g}_k(t)$ is a decreasing function of t, since by assumption $b_i, d_i > 0$ in (4.2). Hence $t_k \geq t_L^{k-1}$, since otherwise (4.2)–(4.4) would yield $r_k = \hat{g}_k(t_k) > \hat{g}_k(t_L^{k-1}) \geq g_k(t_L^{k-1}) \geq r_k$, a contradiction. Similarly $t_k \leq t_U^{k-1}$, since otherwise $r_k \geq g_k(t_U^{k-1}) \geq \hat{g}_k(t_U^{k-1}) > \hat{g}_k(t_k) = r_k$.

(b) This follows from (a) and (4.5), since $x(t)$ is nondecreasing (cf. (2.1)).

(c) By (4.6) and (3.1), $x^k_{L_k \cup U_k} = x_{L_k \cup U_k}(t_k)$. Since $g(t_k) = b^T x(t_k)$ (cf. (2.2)), $b^T x^k = r$ (cf. (3.1)) and $I_k = N \setminus (L_k \cup U_k)$ (cf. Steps 0 and 4), we have

$$g(t_k) - r = \sum_{i \in I_k} b_i [x_i(t_k) - x_i^k] + \sum_{i \in L_k \cup U_k} b_i [x_i(t_k) - x_i^k] = \sum_{i \in I_k} b_i [x_i(t_k) - x_i^k].$$

Now, by (2.1), (3.2) and (3.3),

$$x_{I_k^l}(t_k) = l_{I_k^l}, \quad x_{I_k^u}(t_k) = u_{I_k^u}, \quad l_i < x_i(t_k) = x_i^k < u_i \quad \forall i \in I_k \setminus (I_k^l \cup I_k^u), \tag{4.7}$$

and $x_i^k = l_i = u_i \; \forall i \in I_k^l \cap I_k^u$. Using these relations and the definitions (3.2) gives

$$g(t_k) - r = \sum_{i \in I_k^l} b_i (l_i - x_i^k) + \sum_{i \in I_k^u} b_i (u_i - x_i^k) + \sum_{i \in I_k \setminus (I_k^l \cup I_k^u)} b_i [x_i(t_k) - x_i^k] = \nabla_k - \Delta_k.$$

(d) Since $g(t_k) - r = 0$ by (c), we have $t_k \in T_*$ by (2.4) and $x(t_k) = x^*$ by Fact 2.1. By the proof of (c) (cf. (4.7)), $x_i^k = x_i(t_k)$ for all $i \in N \setminus (I_k^l \cup I_k^u)$ at Step 2, and Step 3 resets $x_i^k := x_i(t_k)$ for the remaining $i \in I_k^l \cup I_k^u$ (if any), so $x^k = x(t_k)$.

(e) We have $t_L^k := t_k$, $t_U^k := t_U^{k-1}$ by (4.1). Since $I_{k+1} := I_k \setminus I_k^l$ at Step 4, (4.7) with $x_{I_k}^k = \hat{x}_{I_k}(t_k)$ yields $x_{I_{k+1}}(t_k) = \min\{u_{I_{k+1}}, \hat{x}_{I_{k+1}}(t_k)\}$. On the other hand, since $x_{I_k^l}(t_k) = l_{I_k^l}$ by (4.7), combining (4.2) and (4.6) with (2.2) and (c) gives

$$g_{k+1}(t_k) + \sum_{i \in I_k^l} b_i l_i + \sum_{i \in L_k} b_i l_i + \sum_{i \in U_k} b_i u_i = g(t_k) = r + \nabla_k - \Delta_k,$$

which implies $g_{k+1}(t_k) = r_{k+1} + \nabla_k - \Delta_k$, using $L_{k+1} = L_k \cup I_k^l$, $U_{k+1} = U_k$ in (3.4). Thus $g_{k+1}(t_k) > r_{k+1}$. Similarly, using $t_U^k = t_U^{k-1}$ in (4.3)–(4.4) and then (3.4) gives

$$g_{k+1}(t_U^k) = g_k(t_U^k) - \sum_{i \in I_k^l} b_i \max\left\{ l_i, \hat{x}_i(t_U^k) \right\} \leq r_k - \sum_{i \in I_k^l} b_i l_i = r_{k+1}.$$

Combining the preceding relations, we obtain (4.3)–(4.4) for k increased by 1.

Next, we have $x_i(t_k) = l_i$ for all i in $L_{k+1} = L_k \cup I_k^l$ (cf. (4.6), (4.7)), whereas $g(t_k) > r$ by (c). Hence $x_{L_{k+1}}(t_k) = l_{L_{k+1}} = x^*_{L_{k+1}}$ by Fact 2.2(a), i.e., (4.5) holds for k increased by 1. Since $x_{U_k}(t_k) = x^*_{U_k}$ by (4.6), if we had $I_k^l = I_k$ then $L_{k+1} \cup U_k = N$ and $x(t_k) = x^*$ combined with Fact 2.1 and (2.4) would give $g(t_k) = r$, a contradiction.

(f) The argument is symmetric to that of part (e). □

We may now state and prove our principal convergence result.

Theorem 4.2. *Algorithm* 3.1 *is well defined and terminates with* $x^k = x^*$, $t_k \in T_*$.

Proof. Clearly, conditions (4.3)–(4.5) hold for $k = 1$. Indeed, since $I_k = N$, (4.3) means $\lim_{t \to -\infty} x(t) = u$, $\lim_{t \to \infty} x(t) = l$ (cf. (2.1)), whereas for $L_k = U_k = \emptyset$ and $r_k = r$, (4.4) reduces to $b^T u \geq r \geq b^T l$ (feasibility). The conclusion follows from Lemma 4.1 by induction, with parts (e,f) ensuring that $I_{k+1} \neq \emptyset$ at Step 4. □

5 Modifications and relations with other methods

5.1 Updating and incremental forms

Each iteration of Algorithm 3.1 requires finding t_k and $x_{I_k}^k$ (cf. (3.3)). Now, by (3.4),

$$t_k = (p_k - r_k)/q_k \quad \text{with} \quad p_k := \sum_{i\in I_k} a_i b_i/d_i, \quad q_k := \sum_{i\in I_k} b_i^2/d_i. \tag{5.1}$$

To save work, we may update p_k, q_k and r_k by using the "fixed" set $I_k^- := I_k \setminus I_{k+1}$ in

$$p_{k+1} = p_k - \sum_{i\in I_k^-} a_i b_i/d_i, \quad q_{k+1} = q_k - \sum_{i\in I_k^-} b_i^2/d_i, \tag{5.2}$$

$$r_{k+1} = r_k - \sum_{i\in I_k^-} b_i \begin{cases} l_i & \text{if } I_k^- = I_k^l, \\ u_i & \text{if } I_k^- = I_k^u. \end{cases} \tag{5.3}$$

This updating technique of [RJL92, BSS96] may be improved as follows.

Relations (5.1)–(5.3), (3.3) and (3.2) yield the *incremental multiplier* formula

$$t_{k+1} = t_k + \frac{1}{q_{k+1}} \begin{cases} \nabla_k & \text{if } I_k^- = I_k^l, \\ -\Delta_k & \text{if } I_k^- = I_k^u. \end{cases} \tag{5.4}$$

Indeed, for $\nabla_k > \Delta_k$ (the opposite case is similar), we have $I_k^- = I_k^l$ and

$$\begin{aligned} q_{k+1}t_{k+1} &= p_{k+1} - r_{k+1} = p_k - r_k - \sum_{i\in I_k^l} a_i b_i/d_i + \sum_{i\in I_k^l} b_i l_i \\ &= q_k t_k - t_k \sum_{i\in I_k^l} b_i^2/d_i + \sum_{i\in I_k^l} b_i[l_i - (a_i - t_k b_i)/d_i] \\ &= q_{k+1}t_k + \sum_{i\in I_k^l} b_i(l_i - x_i^k) = q_{k+1}t_k + \nabla_k. \end{aligned} \tag{5.5}$$

Further, using the facts $\sum_{i\in I_k} b_i x_i^k = r_k$, $I_k = I_{k+1} \cup I_k^-$ and (5.3) in (5.5) yields

$$q_{k+1}(t_{k+1} - t_k) = \sum_{i\in I_k^l} b_i(l_i - x_i^k) = \sum_{i\in I_k^l} b_i l_i - r_k + \sum_{i\in I_{k+1}} b_i x_i^k = -r_{k+1} + \sum_{i\in I_{k+1}} b_i x_i^k$$

for $\nabla_k > \Delta_k$, with l replaced by u for $\nabla_k < \Delta_k$. Hence we also have

$$t_{k+1} = t_k + \left(\textstyle\sum_{i\in I_{k+1}} b_i x_i^k - r_{k+1}\right)/q_{k+1}. \tag{5.6}$$

Of course, by (3.3), we may also update

$$x_i^{k+1} = x_i^k - (t_{k+1} - t_k)b_i/d_i, \quad i \in I_{k+1}. \tag{5.7}$$

The incremental formula (5.4) saves work by not requiring the updates of p_k and r_k. The second formula (5.6) and the update (5.7) are listed for comparisons (cf. §5.8).

5.2 Fixing fewer variables

The convergence results of §§3–4 hold for I_k^l and I_k^u replaced by their subsets

$$I_k^< := \left\{ i \in I_k : x_i^k < l_i \right\} \quad \text{and} \quad I_k^> := \left\{ i \in I_k : x_i^k > u_i \right\}. \tag{5.8}$$

However, this version is less efficient, since it may fix fewer variables per iteration.

5.3 Stopping criteria

Note that $l_{I_k} \le x_{I_k}^k \le u_{I_k}$ iff $I_k^< = I_k^> = \emptyset$ (by (5.8)) iff $\nabla_k = \Delta_k = 0$ (by (3.2)), in which case Step 3 needn't reset x^k. Further, by the proof of Lemma 4.1(e,f), if $I_{k+1} = \emptyset$ then $\nabla_k = \Delta_k$. Thus our stopping criterion $\nabla_k = \Delta_k$ *subsumes* the criteria $l_{I_k} \le x_{I_k}^k \le u_{I_k}$, as well as $I_{k+1} = \emptyset$. In practice, choosing a *feasibility tolerance* $\epsilon_{\text{tol}} \ge 0$, we may use the stopping criterion:

$$|\nabla_k - \Delta_k| \le \epsilon_{\text{tol}} \max\{ 1, |r| \} \quad \text{or} \quad I_{k+1} = \emptyset;$$

it will *guarantee* termination even under roundoff error (since I_k shrinks).

5.4 An illustrative example

For future comparisons, consider the following example. Let $e := (1, \ldots, 1) \in \mathbb{R}^n$.

Example 5.1. For $n = 2$, let $d = b = e$, $a = 0$, $r = 1$, $l = (1, -1)$, $u = (2, 0)$, so that $x^* = (1, 0)$ and $T_* = [-1, 0]$. Then $r_1 = 1$, $t_1 = -0.5$, $x^1 = (0.5, 0.5)$, $\nabla_1 = \Delta_1 = 0.5$, $I_1^l = \{1\}$, $I_1^u = \{2\}$ and Step 3 of Algorithm 3.1 resets x^1 to $x(t_1) = x^*$ before terminating.

5.5 Revisiting the Bitran-Hax algorithm

The Bitran-Hax (BH for short) algorithm [BiH81] differs from Algorithm 3.1 in two aspects. First, at Step 4 it replaces the condition $\nabla_k < \Delta_k$ by $\nabla_k \le \Delta_k$; our version is symmetric. Second, its stopping criterion (cf. §5.3)

$$l_{I_k} \le x_{I_k}^k \le u_{I_k} \quad \text{or} \quad I_{k+1} = \emptyset \tag{5.9}$$

may be less efficient than our criterion $\nabla_k = \Delta_k$; e.g., the BH algorithm solves Example 5.1 in two iterations (with $U_2 = \{2\}$, $I_2 = \{1\}$, $t_2 = -1$, $x^2 = x^*$). However, the case $I_{k+1} = \emptyset$ is not covered by the main convergence proof of [BiH81, Thm 3]; in our setting, the second condition of (5.9) is *redundant*, as shown below.

Lemma 5.2. *If* (4.6) *holds and either* $I_k^l = I_k$ *or* $I_k^u = I_k$, *then* $l_{I_k} \le x_{I_k}^k \le u_{I_k}$.

Proof. Suppose $I_k^l = I_k$ (the case $I_k^u = I_k$ is similar). Then $x_{I_k}^k \le l_{I_k} = x_{I_k}(t_k)$ by (3.2a) and (4.7), $x(t_k) = x^*$ by the proof of Lem. 4.1(e), whereas (3.1), (3.4) and (4.6) yield $b_{I_k}^T x_{I_k}^k = r_k = b_{I_k}^T x_{I_k}^*$. Since $b > 0$, $x_{I_k}^k \le x_{I_k}^*$ and $b_{I_k}^T x_{I_k}^k = b_{I_k}^T x_{I_k}^*$ give $x_{I_k}^k = x_{I_k}^*$. □

Consequently, an equivalent version of the BH algorithm is obtained from Algorithm 3.1 by replacing the condition $\nabla_k < \Delta_k$ by $\nabla_k \le \Delta_k$ in Step 4, and Step 3 by

Step 3' (*Stopping criterion*). If $l_{I_k} \le x^k_{I_k} \le u_{I_k}$ then stop ($x^k = x^*$).

Theorem 5.3. *The BH algorithm is well defined and terminates with* $x^k = x^*$, $t_k \in T_*$.

Proof. We only show how to modify the analysis of §4. Since $l_{I_k} \le x^k_{I_k} \le u_{I_k}$ iff $I^<_k = I^>_k = \emptyset$ iff $\nabla_k = \Delta_k = 0$ (cf. §5.3), we have $\max\{\nabla_k, \Delta_k\} > 0$ at Step 4, and using $x^k_i = x_i(t_k)$ for all $i \in N \setminus (I^<_k \cup I^>_k)$, we may replace (d) of Lem. 4.1 by

(d') *If* $l_{I_k} \le x^k_{I_k} \le u_{I_k}$, *then* $t_k \in T_*$ *and* $x^k = x^*$.

Next, $\nabla_k \le \Delta_k$ replaces $\nabla_k < \Delta_k$ in Lem. 4.1(f), with Lem. 5.2 showing that $I_{k+1} \ne \emptyset$. Thus Theorem 4.2 holds for the BH algorithm. □

The BH algorithm coincides with Algorithm 3.1 until $\nabla_k = \Delta_k$ occurs; then Algorithm 3.1 terminates, but the BH algorithm may go on. The following example shows that the number of additional iterations of the BH algorithm may be quite large.

Example 5.4. For $n = 2m + 1$ with $m \ge 1$, let $d = b = e$, $a = 0$, $r = 0$, $l_i = i$ and $u_i = \infty$ for $i = 1{:}\,m$, $l_{m+1} = -1$, $u_{m+1} = 1$, $l_i = -\infty$ and $u_i = m + 1 - i$ for $i = m + 2{:}\,n$, so that $T_* = \{0\}$, $x^*_i = l_i$ for $i = 1{:}\,m$, $x^*_{m+1} = 0$, $x^*_i = u_i$ for $i = m + 2{:}\,n$. Algorithm 3.1 generates $t_1 = 0$, $x^1 = 0$, $I^l_1 = \{1{:}\,m\}$, $I^u_1 = \{m + 2{:}\,n\}$, $\nabla_1 = \Delta_1 = \frac{m(m+1)}{2}$, terminating with x^1 reset to x^*. In contrast, the BH algorithm continues with $I_2 = \{1{:}\,m + 1\}$ and $r_2 = \frac{m(m+1)}{2}$, bisecting I_k and decreasing r_k until $I_k = \{m+1\}$ and $r_k = 0$. Our experiments with instances having up to twenty million variables show that the BH algorithm makes $k = \lfloor \log_2(n + 1) \rfloor + 1$ iterations; e.g., $k = 20$ for $n = 10^6 + 1$.

5.6 Ventura's modification of the Bitran-Hax algorithm

Assuming $l < u$, consider the following modification of Algorithm 3.1.

Replace Step 3 by Step 3' of §5.5. At Step 4, if $\nabla_k = \Delta_k$ then set $I_{k+1} = I_k \setminus (I^l_k \cup I^u_k)$, $L_{k+1} = L_k \cup I^l_k$, $U_{k+1} = U_k \cup I^u_k$. At Step 5, if $I_{k+1} = \emptyset$ then reset $x^k_{I^l_k} := l_{I^l_k}$, $x^k_{I^u_k} := u_{I^u_k}$ and stop.

Clearly, this modification behaves like the original version until $\nabla_k = \Delta_k$ occurs.

Lemma 5.5. *Suppose the above modification produces* $\nabla_k = \Delta_k$ *for some* k. *Then* $t_k \in T_*$. *If* (5.9) *holds, then* $x^k = x^*$ *upon termination; otherwise, the next iteration terminates with* $t_{k+1} = t_k$ *and* $x^{k+1} = x^*$. *Consequently, Theorem* 4.2 *remains valid.*

Proof. By Lem. 4.1(d), $t_k \in T_*$, and (5.9) implies $x^k = x^*$ after the final reset, if any. If no termination occurs, then $x_{L_{k+1}}(t_k) = l_{L_{k+1}}$, $x_{U_{k+1}}(t_k) = u_{U_{k+1}}$, $x_{I_{k+1}}(t_k) = x^k_{I_{k+1}}$ by (4.7), and using (5.2) with $I^-_k = I^l_k \cup I^u_k$ and $r_{k+1} = r_k - \sum_{i \in I^l_k} b_i l_i - \sum_{i \in I^u_k} b_i u_i$ as for (5.5) yields $q_{k+1} t_{k+1} = q_{k+1} t_k + \nabla_k - \Delta_k$ and hence $t_{k+1} = t_k$. Thus $x^{k+1}_{I_{k+1}} = x^k_{I_{k+1}}$ by (3.3). Since also $x^{k+1}_{L_{k+1}} = l_{L_{k+1}}$, $x^{k+1}_{U_{k+1}} = u_{U_{k+1}}$ by (3.1), we get $x^{k+1} = x(t_k)$. Then $x(t_k) = x^*$ ($t_k \in T_*$) implies termination due to $l_{I_{k+1}} \le x^{k+1}_{I_{k+1}} \le u_{I_{k+1}}$. □

In effect, this modification may only add one (spurious) final iteration, and it needs $l < u$. The method of [Ven91, Alg. 3] is equivalent to this modification.

5.7 The projection algorithm of Robinson, Jiang and Lerme

The algorithm of [RJL92, §3], introduced for the special case of $d = e$, $a = 0$ and extended to the general case in [BSS96], is related to Algorithm 3.1 as follows.

First, using $I_k^<$ and $I_k^>$ (cf. (5.8)) instead of I_k^l and I_k^u, it may fix fewer variables per iteration. Second, its stopping criterion "$I_k^< \cup I_k^> = \emptyset$ or $\nabla_k = \Delta_k$" is equivalent to $\nabla_k = \Delta_k$ (cf. §5.3). Third, omitting the reset of Step 3, it may produce wrong solutions; e.g., $x^1 = (0.5, 0.5)$ in Example 5.1 (in its original notation, the final step should replace I by $I \setminus (L' \cup U')$; similarly in [BSS96, BrS97]). Fourth, the analysis in [RJL92] assumes implicitly that $I_k \neq \emptyset$ at Step 1, and doesn't show that the final $t_k \in T_*$.

5.8 The algorithms of Shor and Michelot

Consider the case where $d = b = e$, $r > 0$, $l = 0$, $u_i \equiv \infty$, in which x^* is the Euclidean projection of a onto the canonical simplex $\{x \geq 0 : e^T x = r\}$. In this case streamlined versions of Algorithm 3.1 discussed below are more efficient than the algorithms of Shor [Sho79, Eq. (4.62)] and Michelot [Mic86, §4].

Starting with $I_1 = N$ and $t_1 = (\sum_{i=1}^n a_i - r)/n$ (cf. (3.4)), Algorithm 3.1 generates $x^k_{I_k} = a_{I_k} - t_k e_{I_k}$, $x^k_{N\setminus I_k} = 0$, $\nabla_k = -\sum_{i\in I_k : x_i^k \leq 0} x_i^k$, $\Delta_k = 0$, $I_{k+1} = \{i \in I_k : x_i^k > 0\}$, $t_{k+1} = t_k + \nabla_k/|I_{k+1}|$ (cf. (5.4), (5.1)), until $\nabla_k = 0$. To avoid updating x^k, we may use $\nabla_k = \sum_{i\in I_k : t_k \geq a_i}(t_k - a_i)$ and $I_{k+1} = \{i \in I_k : a_i > t_k\}$, setting $x^k_{I_k} = a_{I_k} - t_k e_{I_k}$, $x^k_{N\setminus I_k} = 0$ upon termination. Shor's algorithm [Sho79, Eq. (4.62)] replaces ∇_k by $g(t_k) - r$ ($= \nabla_k$ by Lem. 4.1(c)); note that ∇_k is cheaper to compute than $g(t_k)$.

Alternatively, starting with $I_1 = N$, $t_1 = (\sum_{i=1}^n a_i - r)/n$, $x^1 = a - t_1 e$ and using $I_{k+1} = \{i \in I_k : x_i^k \geq 0\}$ (i.e., $I_k^\leq$ instead of I_k^l; cf. §5.3), $t_{k+1} = t_k + (\sum_{i\in I_{k+1}} x_i^k - r)/|I_{k+1}|$, $x^{k+1}_{I_{k+1}} = x^k_{I_{k+1}} - (t_{k+1} - t_k)e_{I_{k+1}}$ (cf. (5.6)–(5.7)), $x^{k+1}_{N\setminus I_{k+1}} = 0$ until $x^k_{I_k} \geq 0$ (i.e., $\nabla_k = 0$ by (3.2a)), we recover Michelot's algorithm [Mic86, §4]. A more efficient version may use $I_{k+1} = \{i \in I_k : x_i^k > 0\}$ (i.e., I_k^l instead of $I_k^<$; cf. §5.3).

5.9 Recovering all Lagrange multipliers

Once Algorithm 3.1 terminates, the following results may be used for recovering *all* Lagrange multipliers of P. By (2.1)–(2.2), the function g has the following *breakpoints*

$$t_i^l := (a_i - l_i d_i)/b_i, \quad t_i^u := (a_i - u_i d_i)/b_i, \quad i = 1{:}n. \tag{5.10}$$

Lemma 5.6. *Let* $I_* := \{i : x_i^* \in (l_i, u_i)\}$, $L_* := \{i : x_i^* = l_i\}$, $U_* := \{i : x_i^* = u_i\}$.

(a) *If* $I_* \neq \emptyset$, *then* $T_* = \{t_*\}$, *where* $t_* = (a_i - d_i x_i^*)/b_i \ \forall i \in I_*$.

(b) *If* $I_* = \emptyset$, *then* $T_* = [t_L^*, t_U^*] \cap \mathbb{R}$, *where* $t_L^* = \max_{i\in L_*\setminus U_*} t_i^l$, $t_U^* = \min_{i\in U_*\setminus L_*} t_i^u$.

(c) *Upon termination in Step 3, let* $I_*^k := I_k \setminus (I_k^l \cup I_k^u)$, $L_*^k := L_k \cup I_k^l$, $U_*^k := U_k \cup I_k^u$. *Then* $I_*^k = I_*$, $L_*^k \subset L_*$, $U_*^k \subset U_*$, $L_*^k \cup U_*^k = L_* \cup U_*$, $L_*^k \cap U_*^k \subset L_* \cap U_* = \{i : t_i^l = t_u^i\}$.

(d) t, μ, ν *are Lagrange multipliers of* P *iff* $t \in T_*$, $\mu = \mu(t) + \lambda$, $\nu = \nu(t) + \lambda$ *for some* $\lambda \geq 0$ *with* $\lambda^T(u - l) = 0$; *in particular,* $\lambda = 0$ *if* $l < u$.

Proof. (a,b) These follow from (2.1) and the fact that $t \in T_*$ iff $x(t) = x^*$ (Fact 2.1).

Table 6.1: Average, maximum and minimum run times in seconds for uncorrelated, weakly correlated and strongly correlated problems.

	uncorrelated			weakly correl.			strongly correl.			overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.18	0.27	0.11	0.21	0.28	0.16	0.19	0.22	0.16	0.19	0.28	0.11
100000	0.38	0.44	0.28	0.38	0.44	0.27	0.38	0.44	0.33	0.38	0.44	0.27
500000	1.57	1.76	1.32	1.58	1.75	1.32	1.60	1.76	1.38	1.58	1.76	1.32
1000000	3.13	3.40	2.53	3.11	3.41	2.42	3.19	3.46	2.80	3.14	3.46	2.42
1500000	4.48	5.00	3.79	4.63	5.05	3.85	4.60	5.17	3.63	4.57	5.17	3.63
2000000	6.14	6.75	5.05	6.11	6.86	4.55	6.28	6.87	4.89	6.17	6.87	4.55

(c) We have $l_{I_*^k} < x^k_{I_*^k} < u_{I_*^k}$ by (3.2), whereas the proof of Lem. 4.1(c,d) yields $x^k = x^* = x(t_k)$, $x_{L_*^k}(t_k) = l_{L_*^k}$, $x_{U_*^k}(t_k) = u_{U_*^k}$, $I_k = N\backslash(L_k \cup U_k)$. Since $I_*^k = N\backslash(L_*^k \cup U_*^k)$, the conclusion follows from the fact that $\{i : l_i = u_i\} = \{i : t_i^l = t_u^i\}$ by (5.10).

(d) This follows from (2.3), Fact 2.1 and the KKT conditions. □

Lemma 5.6(c) extends easily to all algorithms of §§5.5–5.8.

6 Numerical results

Algorithm 3.1 was programmed in Fortran 77 and run on a notebook PC (Pentium II 400 MHz, 256 MB RAM) under MS Windows 98. The set I_k was maintained as a linked list; instead of maintaining L_k and U_k, the final $x(t_k)$ and $g(t_k)$ were computed directly.

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.

Table 6.1 reports the average, maximum and minimum run times over the 20 instances for each of the listed problem sizes and classes. The run times grow linearly with the problem size.

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

Acknowledgment. I would like to thank A.G. Robinson for useful information.

References

[BiH79] G. R. Bitran and A. C. Hax, *On the solution of convex knapsack problems with bounded variables*, in Survey of Mathematical Programming, A. Prékopa, ed., vol. 1, North-Holland—Akadémiai Kiadó, Amsterdam—Budapest, 1979, pp. 357–367.

[BiH81] ———, *Disaggregation and resource allocation using convex knapsack problems with bounded variables*, Management Sci. **27** (1981) 431–441.

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

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

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

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

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

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

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

[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, *Breakpoint searching 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.

[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.

[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.

[Sho79] N. Z. Shor, *Minimization Methods for Non-Differentiable Functions*, Naukova Dumka, Kiev, 1979 (Russian). English transl., Springer-Verlag, Berlin, 1985.

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