Raport Badawczy

Research Report

RB/61/2006

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 2006

Variable Fixing Algorithms for the Continuous Quadratic Knapsack Problem[1]

K. C. KIWIEL[2]

Communicated by P. Tseng

[1]The author thanks the Associate Editor and an anonymous referee for their helpful comments, and A. G. Robinson for useful information.

[2]Professor, Systems Research Institute, Warsaw, Poland.

Abstract. We study several variations of the Bitran–Hax variable fixing method for the continuous quadratic knapsack problem. We close the gaps in the convergence analysis of several existing methods, and provide more efficient versions. Encouraging computational results are reported for large-scale problems.

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

1 Introduction

The *continuous quadratic knapsack problem* is defined by

$$\text{P:} \quad \min \quad f(x) := \tfrac{1}{2}x^T Dx - a^T x, \tag{1a}$$

$$\text{s.t.} \quad b^T x = r, \tag{1b}$$

$$l \le x \le u, \tag{1c}$$

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 strongly convex. Assuming P is feasible, let x^* denote its unique solution.

Problem P has applications in resource allocation (Refs. 1–3), hierarchical production planning (Ref. 1), network flows (Ref. 4), transportation problems (Ref. 5), multicommodity network flows (Refs. 6–8), constrained matrix problems (Ref. 9), integer quadratic knapsack problems (Refs. 10–11), integer and continuous quadratic optimization over submodular constraints (Ref. 3), Lagrangian relaxation via subgradient optimization (Ref. 12), and quasi-Newton updates with bounds (Ref. 13).

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. Section 2). The earliest $O(n \log n)$ methods of Refs. 6 and 12 sort the breakpoints initially, whereas the $O(n)$ algorithms of Refs. 3, 5 and 13–19 use medians of breakpoint subsets.

The variable fixing methods of Refs. 1, 4, 11 and 20–23, stemming from Ref. 24, 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 (Refs. 4 and 23), since they do not 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 Refs. 21–22 for a special case of P (cf. Section 5.8) have full proofs of convergence. We show that the algorithms of Refs. 11 and 23 fail on a simple counterexample (cf. Section 5.7). The method of Ref. 4 relies implicitly on the convergence framework of Ref. 1 (similar to Ref. 20). However, the proof of the main convergence result of Ref. 1 (Theorem 3) has a gap (cf. Section 5.5); we show how to fill this gap in the case of P (in the more general setting of Ref. 1 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 Refs. 4, 11 and 21–23, as well as ours, may be derived from Ref. 1 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. Example 5.2). 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 (Ref. 2).

The paper is organized as follows. Basic properties of P are reviewed in Section 2. In Section 3 we introduce a symmetric version of the method of Ref. 1. Its convergence is established in Section 4. Various modifications and relations with other methods are discussed in Section 5. Finally, encouraging computational results for large-scale problems are reported in Section 6.

2 Basic Properties of the Problem

Viewing $t \in \mathbb{R}$ as a multiplier for the equality constraint of P in (1), consider the *Lagrangian primal solution* (the minimizer of $f(x) + t(b^T x - r)$ s.t. $l \le x \le u$)

$$x(t) := \min\{\, \max[\, l, D^{-1}(a - tb)\,], u\,\} \tag{2}$$

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

$$g(t) := b^T x(t) \tag{3}$$

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

$$\mu(t) := \max\{\, Dl - a + tb, 0\,\} \quad \text{and} \quad \nu(t) := \max\{\, a - tb - Du, 0\,\}. \tag{4}$$

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

$$T_* := \{\, t : g(t) = r\,\}. \tag{5}$$

Indeed, invoking the Karush–Kuhn–Tucker conditions for P as in Ref. 6, Section 2, Ref. 7, Section 1.2, Ref. 13, Theorem 2.1 and Ref. 16, Theorem 2.1 gives the following result.

Theorem 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 Ref. 14, 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)–(3) and our assumption that $b > 0$, each $x_i(t)$ and $g(t)$ are continuous, piecewise linear and *nonincreasing* functions of t. Hence the set T_* of (5) has the form

$$T_* = [\, t_L^*, t_U^*\,] \cap \mathbb{R} \quad \text{with} \quad t_L^* := \inf\{\, t : g(t) = r\,\}, \quad t_U^* := \sup\{\, t : g(t) = r\,\}, \tag{6}$$

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 result (implicit in Ref. 1 and explicit in Ref. 4, Theorem 6). It states conditions under which some of the components of x can be optimally set to their bounds.

Theorem 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 $g(\hat{t}) \ge r$, then $x_i(t) = l_i = x_i^*$ for all $t \ge \hat{t}$ and $i \in \hat{I}^l$.
(b) If $g(\hat{t}) \le r$, then $x_i(t) = u_i = x_i^*$ for all $t \le \hat{t}$ and $i \in \hat{I}^u$.

3 Variable Fixing Algorithm

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

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 the set $N := \{1{:}\,n\}$ such that $x^*_{L_k} = l_{L_k}$, $x^*_{U_k} = u_{U_k}$. Thus, L_k is the set of variables that can be pegged to lower bound, U_k is the set of variables that can be pegged to upper bound, and I_k is the set of the remaining *free* variables. After fixing $x_{L_k} = l_{L_k}$, $x_{U_k} = u_{U_k}$, we need only consider the remaining free variables x_{I_k} in a *restricted* version of problem P, which is solvable in closed form. If its solution is feasible in P, termination occurs (in fact we use a more efficient stopping criterion based on infeasibilities with respect to lower and upper bounds). Otherwise, the partitioning of the index sets is updated, so that at least one more variable is pegged at the next iteration.

A formal statement of the algorithm is given below.

Algorithm 3.1.

Step 0. Initialization. Set $I_1 := N$, $L_1 := U_1 := \emptyset$, $k := 1$.

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

$$x^k := \arg\min\{\, f(x) : b^T x = r, x_{L_k} = l_{L_k}, x_{U_k} = u_{U_k} \,\}. \tag{7}$$

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

$$\nabla_k := \sum_{i \in I^l_k} b_i(l_i - x^k_i), \quad \text{where} \quad I^l_k := \{\, i \in I_k : x^k_i \le l_i \,\}, \tag{8a}$$

$$\Delta_k := \sum_{i \in I^u_k} b_i(x^k_i - u_i), \quad \text{where} \quad I^u_k := \{\, i \in I_k : x^k_i \ge u_i \,\}. \tag{8b}$$

Step 3. Stopping criterion. If $\nabla_k = \Delta_k$, *reset* $x^k_{I^l_k} := l_{I^l_k}$, $x^k_{I^u_k} := u_{I^u_k}$ and stop.

Step 4. Variable fixing. If $\nabla_k > \Delta_k$, set $I_{k+1} := I_k \setminus I^l_k$, $L_{k+1} := L_k \cup I^l_k$, $U_{k+1} := U_k$; if $\nabla_k < \Delta_k$, set $I_{k+1} := I_k \setminus I^u_k$, $L_{k+1} := L_k$, $U_{k+1} := U_k \cup I^u_k$.

Step 5. Loop. Increase k by 1 and go to Step 1.

Step 1 can be implemented as follows (more efficient implementations are discussed later). At Step 1, $x^k_{L_k} = l_{L_k}$, $x^k_{U_k} = u_{U_k}$. The remaining components may be computed as

$$x^k_i = (a_i - t_k b_i)/d_i, \quad i \in I_k \tag{9}$$

(by the form of f in (1)), where t_k is the *Lagrange multiplier* of (7) given by

$$t_k := \Big(\sum_{i \in I_k} a_i b_i/d_i - r_k\Big) \Big/ \sum_{i \in I_k} b_i^2/d_i \quad \text{with} \quad r_k := r - \sum_{i \in L_k} b_i l_i - \sum_{i \in U_k} b_i u_i; \tag{10}$$

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

$$x^k_{I_k} = \arg\min\Big\{ \sum\nolimits_{i \in I_k} \big(\tfrac{1}{2} d_i x_i^2 - a_i x_i\big) : b^T_{I_k} x_{I_k} = r_k \Big\}. \tag{11}$$

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 Ref. 1, Section 2, and *implicitly* in Ref. 23).

Consider the following estimates of the boundary multipliers t_L^* and t_U^* in (6):

$$t_L^k := \sup\{\, t_j : \nabla_j \geq \Delta_j, j \leq k \,\} \quad \text{and} \quad t_U^k := \inf\{\, t_j : \nabla_j \leq \Delta_j, j \leq k \,\}, \tag{12}$$

with $t_L^0 := -\infty$, $t_U^0 := \infty$. Define the *reduced constraint value* and its *linearization*

$$g_k(t) := b_{I_k}^T x_{I_k}(t) \quad \text{and} \quad \hat{g}_k(t) := b_{I_k}^T \hat{x}_{I_k}(t) \quad \text{with} \quad \hat{x}(t) := D^{-1}(a - tb); \tag{13}$$

note that

$$\hat{g}_k(t_k) = r_k \quad \text{and} \quad x_{I_k}^k = \hat{x}_{I_k}(t_k) \tag{14}$$

by (9) and (11). We shall show that at Step 2

$$x_{I_k}(t_L^{k-1}) = \min\{\, u_{I_k}, \hat{x}_{I_k}(t_L^{k-1}) \,\} \quad \text{and} \quad x_{I_k}(t_U^{k-1}) = \max\{\, l_{I_k}, \hat{x}_{I_k}(t_U^{k-1}) \,\}, \tag{15}$$

$$g_k(t_L^{k-1}) \geq r_k \geq g_k(t_U^{k-1}), \tag{16}$$

$$x_{L_k}(t_L^{k-1}) = l_{L_k} = x_{L_k}^* \quad \text{and} \quad x_{U_k}(t_U^{k-1}) = u_{U_k} = x_{U_k}^*, \tag{17}$$

$$x_{L_k}(t_k) = l_{L_k} = x_{L_k}^* \quad \text{and} \quad x_{U_k}(t_k) = u_{U_k} = x_{U_k}^*. \tag{18}$$

Lemma 4.1. Suppose $I_k \neq \emptyset$ and (15)–(17) hold at Step 2 for some k. Then:
(a) $t_L^{k-1} \leq t_k \leq t_U^{k-1}$.
(b) Condition (18) 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 (15)–(17) hold for k increased by 1, and $I_k^l \neq I_k$ at Step 4.
(f) If $\nabla_k < \Delta_k$, then (15)–(17) hold for k increased by 1, and $I_k^u \neq I_k$ at Step 4.

Proof. (a) Since by assumption $b, d > 0$ in (13), $\hat{g}_k(t)$ is a decreasing function of t. Hence $t_k \geq t_L^{k-1}$, since otherwise for $t_k < t_L^{k-1}$, (14), (15), (13) and (16) 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 we would have the contradiction

$$r_k \geq g_k(t_U^{k-1}) \geq \hat{g}_k(t_U^{k-1}) > \hat{g}_k(t_k) = r_k.$$

(b) This follows from (a) and (17): since $x(t)$ is nonincreasing in (2),

$$l_{L_k} = x_{L_k}(t_L^{k-1}) \geq x_{L_k}(t_k) \geq l_{L_k} \quad \text{and} \quad u_{U_k} \geq x_{U_k}(t_k) \geq x_{U_k}(t_U^{k-1}) = u_{U_k}.$$

(c) By (18) and (7), $x_{L_k \cup U_k}^k = x_{L_k \cup U_k}(t_k)$. Since $g(t_k) = b^T x(t_k)$ by (3), $b^T x^k = r$ by (7), 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), (8) and (9),

$$x_{I_k^l}(t_k) = l_{I_k^l}, \quad x_{I_k^u}(t_k) = u_{I_k^u}, \quad l_i < x_i(t_k) = x_i^k < u_i \quad \forall i \in I_k \setminus (I_k^l \cup I_k^u), \tag{19}$$

and $x_i^k = l_i = u_i \ \forall i \in I_k^l \cap I_k^u$. Using these relations and the definitions (8) 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 (5) and $x(t_k) = x^*$ by Theorem 2.1. By the proof of (c), at Step 2 we have $x_i^k = x_i(t_k)$ for all $i \in N \setminus (I_k^l \cup I_k^u)$ (cf. (19)), whereas 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 that $x^k = x(t_k)$.

(e) We have $t_L^k = t_k$, $t_U^k = t_U^{k-1}$ by (12). Since $I_{k+1} := I_k \setminus I_k^l$ at Step 4, (8a) with $x_{I_k}^k = \hat{x}_{I_k}(t_k)$ (cf. (14)) yields $\hat{x}_{I_{k+1}}(t_k) = x_{I_{k+1}}^k > l_{I_{k+1}}$; hence by (2) and (13),

$$x_{I_{k+1}}(t_k) := \min\{u_{I_{k+1}}, \max[\hat{x}_{I_{k+1}}(t_k), l_{I_{k+1}}]\} = \min\{u_{I_{k+1}}, \hat{x}_{I_{k+1}}(t_k)\}.$$

Next, since $x_{I_k^l}(t_k) = l_{I_k^l}$ by (19), combining (13) and (18) with (3) 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 (10). Thus $g_{k+1}(t_k) > r_{k+1}$. Similarly, using $t_U^k = t_U^{k-1}$ in (15)–(16) and then (10) 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 (15)–(16) 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. (18), (19)), whereas $g(t_k) > r$ by (c). Hence $x_{L_{k+1}}(t_k) = l_{L_{k+1}} = x_{L_{k+1}}^*$ by Theorem 2.2(a), i.e., (17) holds for k increased by 1. Since $x_{U_k}(t_k) = x_{U_k}^*$ by (18), if we had $I_k^l = I_k$, then $L_{k+1} \cup U_k = N$ and $x(t_k) = x^*$ combined with Theorem 2.1 and (5) 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 4.1. Algorithm 3.1 is well defined and terminates with $x^k = x^*$, $t_k \in T_*$.

Proof. Clearly, conditions (15)–(17) hold for $k = 1$. Indeed, since $I_k = N$, (15) means $\lim_{t \to -\infty} x(t) = u$, $\lim_{t \to \infty} x(t) = l$ (cf. (2)), whereas for $L_k = U_k = \emptyset$ and $r_k = r$, (16) reduces to $b^T u \geq r \geq b^T l$ (feasibility). The conclusion follows from Lemma 4.1 by induction, with parts (e) and (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 $x^k_{I_k}$ (cf. (9)). Now, by (10),

$$t_k = (p_k - r_k)/q_k \quad \text{with} \quad p_k := \sum_{i\in I_k} a_i b_i/d_i, \quad q_k := \sum_{i\in I_k} b_i^2/d_i. \tag{20}$$

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

$$p_{k+1} = p_k - \sum_{i\in I_k^-} a_i b_i/d_i, \quad q_{k+1} = q_k - \sum_{i\in I_k^-} b_i^2/d_i, \tag{21}$$

$$r_{k+1} = r_k - \sum_{i\in I_k^-} b_i \begin{cases} l_i & \text{if } I_k^- = I_k^l, \\ u_i & \text{if } I_k^- = I_k^u. \end{cases} \tag{22}$$

This updating technique of Refs. 11 and 23 may be improved as follows.

Relations (20)–(22), (9) and (8) yield the *incremental multiplier* formula

$$t_{k+1} = t_k + \frac{1}{q_{k+1}} \begin{cases} \nabla_k & \text{if } I_k^- = I_k^l, \\ -\Delta_k & \text{if } I_k^- = I_k^u. \end{cases} \tag{23}$$

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

$$\begin{aligned} q_{k+1}t_{k+1} &= p_{k+1} - r_{k+1} = p_k - r_k - \sum_{i\in I_k^l} a_i b_i/d_i + \sum_{i\in I_k^l} b_i l_i \\ &= q_k t_k - t_k \sum_{i\in I_k^l} b_i^2/d_i + \sum_{i\in I_k^l} b_i[l_i - (a_i - t_k b_i)/d_i] \\ &= q_{k+1}t_k + \sum_{i\in I_k^l} b_i(l_i - x_i^k) = q_{k+1}t_k + \nabla_k. \end{aligned} \tag{24}$$

Further, using the facts that $\sum_{i\in I_k} b_i x_i^k = r_k$, $I_k = I_{k+1} \cup I_k^-$ and (22) in (24) 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

$$t_{k+1} = t_k + \left(\sum\nolimits_{i\in I_{k+1}} b_i x_i^k - r_{k+1}\right)/q_{k+1}. \tag{25}$$

Of course, by (9), we may also update

$$x_i^{k+1} = x_i^k - (t_{k+1} - t_k)b_i/d_i, \quad i \in I_{k+1}. \tag{26}$$

The incremental formula (23) saves work by not requiring the updates of p_k and r_k. The second formula (25) and the update (26) are listed for comparisons (cf. Section 5.8).

5.2 Fixing Fewer Variables

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

$$I_k^< := \{ i \in I_k : x_i^k < l_i \} \quad \text{and} \quad I_k^> := \{ i \in I_k : x_i^k > u_i \}. \tag{27}$$

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

5.3 Stopping Criteria

Note that, by (27) and (8), we have

$$l_{I_k} \le x_{I_k}^k \le u_{I_k} \iff I_k^< = I_k^> = \emptyset \iff \nabla_k = \Delta_k = 0, \tag{28}$$

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 in (28), 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 the set I_k shrinks).

5.4 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 of Ref. 1 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. Section 5.3)

$$l_{I_k} \le x_{I_k}^k \le u_{I_k} \quad \text{or} \quad I_{k+1} = \emptyset \tag{29}$$

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 of $I_{k+1} = \emptyset$ is not covered by the main convergence proof of Ref. 1, Theorem 3; in our setting, the second condition of (29) is *redundant*, as shown below.

Lemma 5.1. If (18) 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 (8a) and (19), $x(t_k) = x^*$ by the proof of Lemma 4.1(e), whereas (7), (10) and (18) 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 $l_{I_k} \le x^k_{I_k} \le u_{I_k}$, stop ($x^k = x^*$).

Theorem 5.1. 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 Section 4. In view of (28), we have $\max\{\nabla_k, \Delta_k\} > 0$ at Step 4, and using the fact that $x^k_i = x_i(t_k)$ for all $i \in N \setminus (I^<_k \cup I^>_k)$, we may replace part (d) of Lemma 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, the condition $\nabla_k \le \Delta_k$ replaces $\nabla_k < \Delta_k$ in Lemma 4.1(f), with Lemma 5.1 showing that $I_{k+1} \ne \emptyset$. Consequently, Theorem 4.1 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 5.2. 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 that $l < u$, consider the following modification of Algorithm 3.1.

Replace Step 3 by Step 3' of Section 5.5. At Step 4, if $\nabla_k = \Delta_k$, set

$$I_{k+1} := I_k \setminus (I^l_k \cup I^u_k), \quad L_{k+1} := L_k \cup I^l_k, \quad U_{k+1} := U_k \cup I^u_k;$$

if $I_{k+1} = \emptyset$, 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 5.2. Suppose the above modification produces $\nabla_k = \Delta_k$ for some k. Then $t_k \in T_*$. If (29) 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 4.1 remains valid.

Proof. By Lemma 4.1(d), $t_k \in T_*$, and (29) 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 (19), and using (21) with $I^-_k = I^l_k \cup I^u_k$ and

$$r_{k+1} = r_k - \sum_{i \in I^k_l} b_i l_i - \sum_{i \in I^u_k} b_i u_i$$

as for (24) 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 (9). Since also $x^{k+1}_{L_{k+1}} = l_{L_{k+1}}$, $x^{k+1}_{U_{k+1}} = u_{U_{k+1}}$ by (7), 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 the condition $l < u$. The method of Ref. 4, Algorithm 3 is equivalent to this modification.

5.7 Projection Algorithm of Robinson, Jiang and Lerme

The algorithm of Ref. 23, Section 3, introduced for the special case of $d = e$, $a = 0$ and extended to the general case in Ref. 11, is related to Algorithm 3.1 as follows.

First, using the sets $I^<_k$ and $I^>_k$ (cf. (27)) instead of I^l_k and I^u_k, it may fix fewer variables per iteration. Second, its stopping criterion

$$I^<_k \cup I^>_k = \emptyset \quad \text{or} \quad \nabla_k = \Delta_k$$

is equivalent to $\nabla_k = \Delta_k$ (cf. Section 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 Refs. 2 and 11). Fourth, the analysis in Ref. 23 assumes implicitly that $I_k \neq \emptyset$ at Step 1, and doesn't show that the final $t_k \in T_*$.

5.8 Algorithms of Shor and Michelot

Consider the case where $d = b = e$, $r > 0$, $l = 0$, $u_i \equiv \infty$, in which the solution x^* of P is the Euclidean projection of a onto the canonical simplex $\{x \ge 0 : e^T x = r\}$. In this case streamlined versions of Algorithm 3.1 discussed below are more efficient than the algorithms of Shor (Ref. 21, Eq. (4.62)) and Michelot (Ref. 22, Section 4).

Starting with $I_1 = N$ and $t_1 = (\sum_{i=1}^n a_i - r)/n$ (cf. (10)), Algorithm 3.1 generates

$$x^k_{I_k} = a_{I_k} - t_k e_{I_k}, \quad x^k_{N \setminus I_k} = 0, \quad \nabla_k = -\sum_{i \in I_k : x^k_i \le 0} x^k_i, \quad \Delta_k = 0,$$

$$I_{k+1} = \{i \in I_k : x^k_i > 0\}, \quad t_{k+1} = t_k + \nabla_k / |I_{k+1}|$$

(cf. (23), (20)), until $\nabla_k = 0$. To avoid updating x^k, we may use the formulae

$$\nabla_k = \sum_{i \in I_k : t_k \ge a_i} (t_k - a_i), \quad I_{k+1} = \{i \in I_k : a_i > t_k\},$$

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 (Ref. 21, Eq. (4.62)) replaces ∇_k above by $g(t_k) - r$ ($= \nabla_k$ by Lemma 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

$$I_{k+1} = \{i \in I_k : x^k_i > 0\}, \quad t_{k+1} = t_k + \Big(\sum_{i \in I_{k+1}} x^k_i - r \Big) / |I_{k+1}|,$$

$$x^{k+1}_{I_{k+1}} = x^k_{I_{k+1}} - (t_{k+1} - t_k)e_{I_{k+1}}, \quad x^{k+1}_{N\setminus I_{k+1}} = 0$$

(cf. (25)–(26)) until $x^k_{I_k} \geq 0$ (i.e., $\nabla_k = 0$ by (8a)), we recover a more efficient version of Michelot's algorithm (Ref. 22, Section 4); the original version employs $I_{k+1} = \{i \in I_k : x^k_i \geq 0\}$ (i.e., $I^{\leq}_k$ instead of I^l_k; cf. Section 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)–(3), the function g has the following *breakpoints*

$$t^l_i := (a_i - l_i d_i)/b_i, \quad t^u_i := (a_i - u_i d_i)/b_i, \quad i = 1{:}n. \tag{30}$$

Lemma 5.3. Let

$$I_* := \{i : x^*_i \in (l_i, u_i)\}, \quad L_* := \{i : x^*_i = l_i\}, \quad U_* := \{i : x^*_i = u_i\}.$$

Then:

(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 $t^*_L = \max_{i \in L_* \setminus U_*} t^l_i$, $t^*_U = \min_{i \in U_* \setminus L_*} t^u_i$.

(c) Upon termination in Step 3, let

$$I^k_* := I_k \setminus (I^l_k \cup I^u_k), \quad L^k_* := L_k \cup I^l_k, \quad U^k_* := U_k \cup I^u_k.$$

Then

$$I^k_* = I_*, \quad L^k_* \subset L_*, \quad U^k_* \subset U_*,$$

$$L^k_* \cup U^k_* = L_* \cup U_*, \quad L^k_* \cap U^k_* \subset L_* \cap U_* = \{i : t^l_i = t^i_u\}.$$

(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) and the fact that $t \in T_*$ iff $x(t) = x^*$ (Theorem 2.1).

(c) We have $l_{I^k_*} < x^k_{I^k_*} < u_{I^k_*}$ by (8), whereas the proof of Lemma 4.1(c,d) yields $x^k = x^* = x(t_k)$, $x_{L^k_*}(t_k) = l_{L^k_*}$, $x_{U^k_*}(t_k) = u_{U^k_*}$, $I_k = N\setminus(L_k \cup U_k)$. Since $I^k_* = N\setminus(L^k_* \cup U^k_*)$, the conclusion follows from the fact that $\{i : l_i = u_i\} = \{i : t^l_i = t^i_u\}$ by (30).

(d) This follows from (4), Theorem 2.1 and the KKT conditions. □

Lemma 5.3(c) extends easily to all algorithms of Sections 5.5–5.8.

6 Numerical Results

We report here on our experience with Algorithm 3.1 and Kiwiel's breakpoint searching method of Ref. 19. Both methods were programmed in Fortran 77 and run on a notebook PC (Pentium M 755 2 GHz, 1.5 GB RAM) under MS Windows XP. For Algorithm 3.1, 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. For Kiwiel's method, we used the median finding routine of Ref. 25.

Our test problems were randomly generated with n ranging between 50000 and 2000000. As in Ref. 10 (Section 2), all parameters were distributed uniformly in the intervals of the following three problem classes:

(i) uncorrelated: $a_i, b_i, d_i \in [10, 25]$;
(ii) weakly correlated: $b_i \in [10, 25]$, $a_i, d_i \in [b_i - 5, b_i + 5]$;
(iii) 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 1 reports the average, maximum and minimum run times of Algorithm 3.1 in seconds over the 20 instances for each of the listed problem sizes and classes. The run times grow linearly with the problem size.

Table 2 gives the run times of Kiwiel's method. Again, the run times grow linearly with the problem size. Kiwiel's method is slower than Algorithm 3.1 by about 14% on average, but its run times are more stable. The relatively good performance of Kiwiel's method is due to the high efficiency of the median finding routine of Ref. 25.

To save space, we only add that Ref. 19 showed that the methods of Refs. 13–14 were slower than Kiwiel's method by about 21% and 23%, respectively; in other words, they were slower than Algorithm 3.1 by about 39%.

As for the earlier computational comparisons of Refs. 4 and 23, we recall from Sections 5.6–5.7 that the variable fixing methods of Ref. 4, Algorithm 3 and Ref. 23, Section 3 are close to Algorithm 3.1. On the other hand, the breakpoint searching methods of Ref. 4, Algorithm 3 and Ref. 16, tested in Ref. 23, are quite similar to Kiwiel's method, whereas the $O(n \log n)$ sorting-based method of Ref. 6 is clearly less efficient even for moderate values of n (see Refs. 4 and 11). With these similarities in mind, our results confirm the main finding of Refs. 4 and 23 that the variable fixing methods are faster than the breakpoint searching methods; however, the performance gap becomes quite small (14% on average) when a state-of-the-art median finding routine is used.

References

1. BITRAN, G. R., and HAX, A. C., *Disaggregation and Resource Allocation Using Convex Knapsack Problems with Bounded Variables*, Management Science, Vol. 27, pp. 431–441, 1981.
2. BRETTHAUER, K. M., and SHETTY, B., *Quadratic Resource Allocation with Generalized Upper Bounds*, Operational Research Letters, Vol. 20, pp. 51–57, 1997.
3. HOCHBAUM, D. S., and HONG, S. P., *About Strongly Polynomial Time Algorithms for Quadratic Optimization over Submodular Constraints*, Mathematical Programming, Vol. 69, pp. 269–309, 1995.
4. VENTURA, J. A., *Computational Development of a Lagrangian Dual Approach for Quadratic Networks*, Networks, Vol. 21, pp. 469–485, 1991.
5. COSARES, S., and HOCHBAUM, D. S., *Strongly Polynomial Algorithms for the Quadratic Transportation Problem with a Fixed Number of Sources*, Mathematics of Operations Research, Vol. 19, pp. 94–111, 1994.
6. HELGASON, K., KENNINGTON, J., and LALL, H., *A polynomially Bounded Algorithm for a Singly Constrained Quadratic Program*, Mathematical Programming, Vol. 18, pp. 338–343, 1980.
7. NIELSEN, S. S., and ZENIOS, S. A., *Massively Parallel Algorithms for Singly Constrained Convex Programs*, ORSA Journal on Computing, Vol. 4, pp. 166–181, 1992.
8. SHETTY, B., and MUTHUKRISHNAN, R., *A Parallel Projection for the Multicommodity Network Model*, Journal of the Operational Research Society, Vol. 41, pp. 837–842, 1990.
9. COTTLE, R. W., DUVALL, S. G., and ZIKAN, K., *A Lagrangean Relaxation Algorithm for the Constrained Matrix Problem*, Naval Research Logistics Quarterly, Vol. 33, pp. 55–76, 1986.
10. BRETTHAUER, K. M., SHETTY, B., and SYAM, S., *A Branch and Bound Algorithm for Integer Quadratic Knapsack Problems*, ORSA Journal on Computing, Vol. 7, pp. 109–116, 1995.
11. BRETTHAUER, K. M., SHETTY, B., and SYAM, S., *A Projection Method for the Integer Quadratic Knapsack Problem*, Journal of the Operational Research Society, Vol. 47, pp. 457–462, 1996.
12. HELD, M., WOLFE, P., and CROWDER, H. P., *Validation of Subgradient Optimization*, Mathematical Programming, Vol. 6, 62–88, 1974.
13. CALAMAI, P. H., and MORÉ, J. J., *Quasi-Newton Updates with Bounds*, SIAM Journal on Numerical Analysis, Vol. 24, pp. 1434–1441, 1987.
14. BRUCKER, P., *An $O(n)$ Algorithm for Quadratic Knapsack Problems*, Operations Research Letters, Vol. 3, pp. 163–166, 1984.

15. MACULAN, N., and DE PAULA, JR., G. G., *A Linear-time Median-finding Algorithm for Projecting a Vector on the Simplex of R^n*, Operations Research Letters, Vol. 8, pp. 219–222, 1989.

16. PARDALOS, P. M., and KOVOOR, N., *An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Upper and Lower Bounds*, Mathematical Programming, Vol. 46, pp. 321–328, 1990.

17. MACULAN, N., MINOUX, M., and PLATEAU, G., *An $O(n)$ Algorithm for Projecting a Vector on the Intersection of a Hyperplane and R^n_+*, RAIRO Recherche Opérationelle, Vol. 31, pp. 7–16, 1997.

18. MACULAN, N., SANTIAGO, C. P., MACAMBIRA, E. M., and JARDIM, M. H. C., *An $O(n)$ Algorithm for Projecting a Vector on the Intersection of a Hyperplane and a Box in R^n*, Journal of Optimization Theory and Applications, Vol. 117, pp. 553–574, 2003.

19. KIWIEL, K. C., *On Linear Time Algorithms for the Continuous Quadratic Knapsack Problem*, Journal of Optimization Theory and Applications, Vol. 133, 2007. To appear.

20. BITRAN, G. R., and HAX, A. C., *On the Solution of Convex Knapsack Problems with Bounded Variables*, Survey of Mathematical Programming, Vol. 1, Edited by A. Prékopa, North-Holland—Akadémiai Kiadó, Amsterdam—Budapest, pp. 357–367, 1979.

21. SHOR, N. Z., *Minimization Methods for Non-Differentiable Functions*, Naukova Dumka, Kiev, 1979 (in Russian); English translation Springer-Verlag, Berlin, 1985.

22. MICHELOT, C., *A Finite Algorithm for Finding the Projection of a Point Onto the Canonical Simplex of R^n*, Journal of Optimization Theory and Applications, Vol. 50, pp. 195–200, 1986.

23. ROBINSON, A. G., JIANG, N., and LERME, C. S., *On the Continuous Quadratic Knapsack Problem*, Mathematical Programming, Vol. 55, pp. 99–108, 1992.

24. LUSS, H., and GUPTA, S. K., *Allocation of Effort Resources among Competing Activities*, Operations Research, Vol. 23, pp. 360–366, 1975.

25. KIWIEL, K. C., *On Floyd and Rivest's* SELECT *Algorithm*, Theoretical Computer Science, Vol. 347, pp. 214–238, 2005.

List of Tables

Table 1: Run times of Algorithm 3.1 (sec).

	Uncorrelated			Weakly Correlated			Strongly Correlated			Overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.02	0.01	0.02	0.02	0.01	0.02	0.02	0.01	0.02	0.02	0.01
100000	0.05	0.05	0.04	0.05	0.05	0.04	0.05	0.05	0.04	0.05	0.05	0.04
500000	0.25	0.28	0.22	0.24	0.27	0.22	0.23	0.25	0.20	0.24	0.28	0.20
1000000	0.50	0.55	0.45	0.48	0.55	0.44	0.48	0.50	0.44	0.49	0.55	0.44
1500000	0.72	0.83	0.59	0.72	0.81	0.66	0.71	0.75	0.58	0.72	0.83	0.58
2000000	0.97	1.08	0.88	0.94	1.04	0.78	0.95	1.00	0.88	0.95	1.08	0.78

Table 2: Run times of the Kiwiel breakpoint searching algorithm (sec).

	Uncorrelated			Weakly Correlated			Strongly Correlated			Overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.08	0.02	0.02	0.03	0.02	0.03	0.05	0.02	0.02	0.08	0.02
100000	0.05	0.06	0.05	0.05	0.06	0.05	0.05	0.06	0.05	0.05	0.06	0.05
500000	0.27	0.28	0.25	0.27	0.28	0.26	0.27	0.28	0.26	0.27	0.28	0.25
1000000	0.53	0.55	0.51	0.54	0.55	0.51	0.54	0.55	0.52	0.54	0.55	0.51
1500000	0.80	0.82	0.76	0.80	0.82	0.77	0.80	0.82	0.77	0.80	0.82	0.76
2000000	1.08	1.09	1.02	1.08	1.10	1.02	1.08	1.09	1.03	1.08	1.10	1.02