42/2006

Raport Badawczy

Research Report

RB/56/2006

On linear time 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

TECHNICAL NOTE
On Linear Time Algorithms for the Continuous Quadratic Knapsack Problem[1]

K. C. Kiwiel[2]

Communicated by J. P. Crouzeix

[1]The author thanks the Associate Editor and an anonymous referee for their helpful comments.

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

Abstract. We give a linear time algorithm for the continuous quadratic knapsack problem which is simpler than the existing methods and competitive in practice. Encouraging computational results are presented 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

$$\begin{aligned} \text{P:} \qquad \min \qquad & f(x) := \tfrac{1}{2}x^T Dx - a^T x, && (1a)\\ \text{s.t.} \qquad & b^T x = r, && (1b)\\ & l \le x \le u, && (1c) \end{aligned}$$

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

Problem P has many applications; see e.g. Refs. 1–5 and references therein.

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. Section 2). To this end, the $O(n)$ algorithms of Refs. 1–4 use medians of breakpoint subsets. However, they are quite complicated, and the analysis of Refs. 3–4 has some gaps that are not easy to fix (cf. Remark 2.1(iv)).

In this paper we introduce a simpler $O(n)$ algorithm that is easier to analyze and competitive in practice with those in Refs. 1–4.

The paper is organized as follows. In Section 2 we review some properties of P and present our method. Additional constructions of Ref. 2 are discussed in Section 3. Finally, computational results for large-scale problems are reported in Section 4.

2 Breakpoint Searching Algorithm

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. $l \le x \le u$)

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

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

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

Solving P amounts to solving $g(t) = r$. Indeed, invoking the Karush–Kuhn–Tucker conditions for P as in Ref. 2, Theorem 2.1 and Ref. 4, Theorem 2.1 gives the following result.

Fact 2.1. $x^* = x(t)$ iff $g(t) = r$. Further, the set $T_* := \{ t : g(t) = r \}$ is nonempty.

As in Ref. 1, we assume for simplicity that $b > 0$, because if $b_i = 0$, x_i may be eliminated:

$$x_i^* = \min\left\{ \max\left[l_i, a_i/d_i \right], u_i \right\},$$

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), the function g has the following *breakpoints*

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

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

$$x_i(t) = \begin{cases} u_i & \text{if } t \le t_i^u, \\ (a_i - tb_i)/d_i & \text{if } t_i^u \le t \le t_i^l, \\ l_i & \text{if } t_i^l \le t. \end{cases} \tag{4}$$

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

To locate an optimal t_* in T_*, the algorithm below generates a *bracketing interval* $[t_L, t_U]$ that contains T_* by evaluating g at median breakpoints in (t_L, t_U) until (t_L, t_U) contains no breakpoints; then g is linear on $[t_L, t_U]$, and t_* is found by interpolation.

Algorithm 2.1.

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

$$t_* := t_L - [\, g(t_L) - r\,] \frac{t_U - t_L}{g(t_U) - g(t_L)}. \tag{5}$$

The following comments clarify the nature of the algorithm.

Remark 2.1.

(i) By the argument of Ref. 2 (p. 1438), Algorithm 2.1 requires only order n operations, since $|T|$ is originally $2n$, $\hat t :=$ median(T) can be obtained in order $|T|$ operations (Ref. 6, Section 5.3.3), the evaluation of $g(\hat t)$ requires order $|T|$ operations (see below), and each iteration reduces $|T|$ at least by half at Steps 4 or 5.

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

$$L := \{\, i : t_i^l \le t_L \,\}, \tag{6a}$$

$$M := \{\, i : t_L, t_U \in [\, t_i^u, t_i^l \,] \,\}, \tag{6b}$$

$$U := \{\, i : t_U \le t_i^u \,\}, \tag{6c}$$

$$I := \{\, i : t_i^l \in (t_L, t_U) \text{ or } t_i^u \in (t_L, t_U) \,\}; \tag{6d}$$

note that $|I| \le |T| \le 2|I|$. Thus, by (3), (4) and (6),

$$g(t) = \sum_{i\in I} b_i x_i(t) + (\, p - tq\,) + s \quad \forall t \in [\, t_L, t_U \,], \tag{7}$$

where
$$\sum_{i\in I} b_i x_i(t) = \sum_{i\in I: t\in[t_i^u, t_i^l]} b_i(a_i - tb_i)/d_i + \sum_{i\in I: t_i^l<t} b_i l_i + \sum_{i\in I: t<t_i^u} b_i u_i,$$
$$p := \sum_{i\in M} a_i b_i/d_i, \quad q := \sum_{i\in M} b_i^2/d_i \quad \text{and} \quad s := \sum_{i\in L} b_i l_i + \sum_{i\in U} b_i u_i.$$
Setting $I := N$, $p, q, s := 0$ at Step 0, at Step 6 we may update I, p, q and s as follows:

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

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

(iii) Upon termination, $x^* = x(t_*)$ is recovered via (2) in order n operations.

(iv) The algorithm of Ref. 4 is quite similar to ours, but it fails on simple examples (e.g., for $n = 2$, $d = b = (1, 1)$, $a = 0$, $r = -2$, $l = (-2, -2)$, $u = (-1, 0)$). The algorithm of Ref. 3 is much more complicated, and may also fail (e.g., on the example of Ref. 3, pp. 565–566).

3 Breakpoint Removal of Calamai and Moré

The original version of Algorithm 2.3 in Ref. 2 corresponds to replacing 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 $g(\check{t}) > r$, set $t_L := \check{t}$, $T := \{t \in T : \check{t} \le t\}$, else set $t_L := \hat{t}$, $t_U := \min\{t_U, \check{t}\}$ and stop with t_* given by (5).

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 $g(\check{t}) < r$, set $t_U := \check{t}$, $T := \{t \in T : t \le \check{t}\}$, else set $t_L := \max\{t_L, \check{t}\}$, $t_U := \hat{t}$ and stop with t_* given by (5).

By (4) and (7), because $\hat{t}$ and $\check{t}$ are *consecutive* breakpoints, we may compute
$$g(\check{t}) = g(\hat{t}) - (\,\check{t} - \hat{t}\,)\,(\,q + \check{q}\,)$$
with
$$\check{q} := \sum_{i\in I: \hat{t}, \check{t} \in [t_i^u, t_i^l]} b_i^2/d_i$$
in order $|I|$ operations. Yet this modification will typically remove only one more breakpoint, and this may not be worth the additional effort in finding $\check{t}$ and $g(\check{t})$.

4 Numerical Results

Algorithm 2.1 of this paper, the Calamai–Moré version of Section 3, and Brucker's method of Ref. 1 were programmed in Fortran 77 and run on a notebook PC (Pentium M 755 2 GHz, 1.5 GB RAM) under MS Windows XP. We used the median finding routine of Ref. 7, which permutes the list T to place elements $< \hat{t}$ first, then elements $= \hat{t}$, and finally elements $> \hat{t}$. Hence the updates of Steps 4 and 5 require only a change of one pointer.

Our test problems were randomly generated with n ranging between 50000 and 2000000. As in Ref. 5 (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.

Tables 1–3 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 Calmai-Moré algorithm and the Brucker algorithm one were slower than our method by about 21% and 23%, respectively.

References

1. Brucker, P., *An $O(n)$ Algorithm for Quadratic Knapsack Problems*, Operations Research Letters, Vol. 3, pp. 163–166, 1984.
2. Calamai, P. H., and Moré, J. J., *Quasi-Newton Updates with Bounds*, SIAM Journal on Numerical Analysis, Vol. 24, pp. 1434–1441, 1987.
3. 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.
4. 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.
5. 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.
6. Knuth, D. E., *The Art of Computer Programming. Volume III: Sorting and Searching*, 2nd Edition, Addison-Wesley, Reading, Massachussets, 1998.
7. 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 2.1 (sec).

	Uncorrelated			Weakly Correlated			Strongly Correlated			Overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.08	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

Table 2: Run times of the Calamai–Moré algorithm (sec).

	Uncorrelated			Weakly Correlated			Strongly Correlated			Overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.02	0.03	0.02	0.03	0.03	0.02	0.03	0.03	0.02	0.03	0.03	0.02
100000	0.06	0.07	0.05	0.06	0.07	0.06	0.06	0.07	0.06	0.06	0.07	0.05
500000	0.32	0.34	0.31	0.33	0.34	0.31	0.33	0.34	0.32	0.33	0.34	0.31
1000000	0.65	0.67	0.62	0.65	0.66	0.63	0.66	0.67	0.64	0.65	0.67	0.62
1500000	0.98	1.00	0.94	0.98	1.00	0.95	0.98	1.00	0.94	0.98	1.00	0.94
2000000	1.31	1.34	1.25	1.31	1.33	1.25	1.32	1.33	1.26	1.31	1.34	1.25

Table 3: Run times of the Brucker algorithm (sec).

	Uncorrelated			Weakly Correlated			Strongly Correlated			Overall		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
50000	0.03	0.03	0.01	0.03	0.06	0.02	0.03	0.06	0.02	0.03	0.06	0.01
100000	0.06	0.07	0.05	0.06	0.07	0.05	0.07	0.07	0.06	0.06	0.07	0.05
500000	0.33	0.38	0.24	0.34	0.37	0.28	0.33	0.36	0.26	0.33	0.38	0.24
1000000	0.66	0.76	0.49	0.69	0.79	0.52	0.65	0.72	0.52	0.67	0.79	0.49
1500000	1.01	1.13	0.85	1.01	1.17	0.84	0.99	1.07	0.80	1.00	1.17	0.80
2000000	1.35	1.51	1.10	1.31	1.45	1.00	1.33	1.44	1.08	1.33	1.51	1.00