THE SEPARABLE CONVEX QUADRATIC KNAPSACK PROBLEM *

TIMOTHY A. DAVIS†, WILLIAM W. HAGER‡, AND JAMES T. HUNGERFORD§

Abstract. This paper considers the problem of minimizing a convex, separable quadratic function subject to a knapsack constraint and a box constraint. An algorithm called NAPHEAP is developed for solving this problem. The algorithm solves the Karush-Kuhn-Tucker system using a starting guess to the optimal Lagrange multiplier and updating the guess monotonically in the direction of the solution. The starting guess is either computed using a Newton-type method (variable fixing method, secant method, or Newton's method) or is supplied by the user. A key innovation in our algorithm is the implementation of a heap data structure for storing the break points of the derivative of the dual function. Given a starting guess, the heap is built with about n comparisons, where n is the problem dimension. Each subsequent iteration amounts to an update of the heap, which can be done in about $\log_2 n$ comparisons. Hence, when the starting guess is sufficiently good, the cost of the iteration updates can be a small multiple of $\log_2 n$ comparisons. In contrast, Newton-type methods can require $O(n)$ operations per iteration, even when the starting guess is near the solution. Hence, a hybrid algorithm that uses a Newton-type method to generate a starting guess, followed by the heap-based monotone break point searching scheme, can be faster than a Newton-type method by itself. Numerical results are presented.

Key words. continuous quadratic knapsack, nonlinear programming, convex programming, quadratic programming, separable programming, singly-constrained quadratic program, heap, support vector machines

AMS subject classifications. 90-08, 90C20, 90C25, 90C35, 90A80

1. Introduction.

We consider the following *separable convex quadratic knapsack problem*:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{x} - \mathbf{y}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u} \quad \text{and} \quad r \le \mathbf{a}^\mathsf{T}\mathbf{x} \le s, \end{aligned} \tag{1.1}$$

where $\mathbf{a}, \mathbf{y} \in \mathbb{R}^n$, $\boldsymbol{\ell} \in (\mathbb{R} \cup \{-\infty\})^n$, $\mathbf{u} \in (\mathbb{R} \cup \{\infty\})^n$, $r, s \in \mathbb{R}$, and $\mathbf{D} \in \mathbb{R}^{n\times n}$ is a positive semidefinite diagonal matrix with diagonal $\mathbf{d}$. Without loss of generality, we assume that $\boldsymbol{\ell} < \mathbf{u}$. Problem (1.1) has applications in quadratic resource allocation [1, 2, 9, 17], quasi-Newton updates with bounds [5], multi-capacity network flow problems [16, 22, 26], continuous formulations of graph partitioning problems [14, 13], and support vector machines [10].

By introducing an auxiliary variable $b \in \mathbb{R}$, we may reformulate (1.1) as:

$$\min_{\mathbf{x}\in\mathbb{R}^n,\ b\in\mathbb{R}} \{q(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad r \le b \le s, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\}. \tag{1.2}$$

*October 3, 2013. The authors gratefully acknowledge support by the Office of Naval Research under grant N00014-11-1-0068 and by the Defense Advanced Research Project Agency under contract HR0011-12-C-0011. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

†davis@cise.ufl.edu, http://www.cise.ufl.edu/∼davis, PO Box 116120, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611-6120. Phone (352) 505-1546. Fax (352) 392-1220.

‡hager@ufl.edu, http://www.math.ufl.edu/∼hager, PO Box 118105, Department of Mathematics, University of Florida, Gainesville, FL 32611-8105. Phone (352) 392-0281. Fax (352) 392-8357.

§freerad@ufl.edu, http://www.math.ufl.edu/∼freerad, PO Box 118105, Department of Mathematics, University of Florida, Gainesville, FL 32611-8105. Phone (352) 392-0281. Fax (352) 392-8357.

Making the substitutions

$$\mathbf{x} \leftarrow \begin{pmatrix} \mathbf{x} \\ b \end{pmatrix}, \quad \boldsymbol{\ell} \leftarrow \begin{pmatrix} \boldsymbol{\ell} \\ r \end{pmatrix}, \quad \mathbf{u} \leftarrow \begin{pmatrix} \mathbf{u} \\ s \end{pmatrix}, \quad \mathbf{a} \leftarrow \begin{pmatrix} \mathbf{a} \\ -1 \end{pmatrix}, \quad n \leftarrow n+1,$$

and augmenting $\mathbf{y}$ and $\mathbf{d}$ by an additional zero entry, (1.2) is transformed into the following problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \{q(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = 0\}.$$

Hence, without loss of generality, we may restrict our study to an equality-constrained version of (1.1):

$$\min_{\mathbf{x}\in\mathbb{R}^n} \{q(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\}, \tag{1.3}$$

where $b \in \mathbb{R}$ is given. The constraints of (1.3) are continuous analogues of the constraints that arise in the discrete knapsack problem (see [3]).

In quadratic resource allocation, problem (1.3) arises with $\mathbf{a} = \mathbf{1}$ and $\boldsymbol{\ell} = \mathbf{0}$, where $\mathbf{1}$ and $\mathbf{0}$ are the vectors whose entries are all 1 and 0 respectively. The objective is to minimize the total cost of allocating b resources to n projects, where $\frac{1}{2}d_i x_i^2 - y_i x_i$ is the cost function for project i. The amount of resources allocated to project i is constrained to lie between ℓ_i and u_i.

In other applications [5, 10, 13, 14, 16, 22, 26], an objective function F is minimized over a feasible set described by bound constraints and a single linear equality constraint:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \{F(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\} \tag{1.4}$$

The gradient projection algorithm as formulated in [15] starts with an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$ to a solution of (1.4), and for each $k \ge 0$, the $(k+1)$st iterate is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k, \quad \text{where } \mathbf{p}_k = \text{proj}(\mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)).$$

Here $s_k \ge 0$ is the stepsize, $\alpha_k \mathbf{I}$ is an approximation to the inverse Hessian of F, and $\text{proj}(\mathbf{z})$ is the unique projection, relative to the 2-norm, of $\mathbf{z}$ onto the feasible set of (1.4). That is, $\text{proj}(\mathbf{z})$ is the solution of the problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \left\{ \frac{1}{2}\|\mathbf{x} - \mathbf{z}\| : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \qquad \mathbf{a}^\mathsf{T}\mathbf{x} = b \right\}. \tag{1.5}$$

This projection problem is a special case of Problem (1.3) in which $\mathbf{D}$ is the identity matrix and $\mathbf{y} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$.

Specialized algorithms for solving (1.3) typically assume $\mathbf{D}$ is positive definite and search for a root of the derivative of the dual function, a continuous piecewise linear, monotonic function with at most $2n$ break points ("kinks" where the slope could change). The first algorithm [16] was based on sorting all the break points and then sequentially marching through the break points until the optimal multiplier was found. The worst case complexity of this algorithm is $O(n \log_2 n)$ due to the sort. Starting with Brucker [4], linear time algorithms were proposed based on a median search rather than a sort. Subsequent developments of the median search method include those in [5, 20, 23].

In [1], Bitran and Hax proposed a method for solving a generalization of (1.3) in which the objective function is convex and separable, but not necessarily quadratic. In their algorithm, which has come to be known as the variable fixing method, each iteration implicitly computes an estimate for the optimal Lagrange multiplier by solving a subproblem in which the box constraints are ignored. Based on the sign of the derivative of the dual function at the multiplier estimate, a non-empty subset of indices is identified for which x_i can be optimally fixed at an upper or lower bound. The fixed variables are removed from the problem and the process repeats until all the bound components of an optimal solution have been determined. Subsequent developments of this approach in the context of (1.3) can be found in [3, 18, 21, 24, 27, 28]. An efficient and reliable implementation of the variable fixing method for (1.3), and a thorough convergence analysis, is given by Kiwiel in [18].

In [10], Dai and Fletcher develop a method in which each multiplier estimate is the root of a secant approximation to the derivative of the dual function (with additional modifications for speeding up convergence). In [7], a method is proposed by Cominetti, Mascarenhas, and Silva which uses semi-smooth Newton steps for updating multiplier estimates, with a secant safeguard. Numerical experiments showed that the semi-smooth Newton method was faster than the variable fixing method, the secant method, and a median based method on a standard test set of randomly generated problems. As we explain in Section 3, Newton's method is not very well suited for problems where $\mathbf{d}$ is small relative to $\mathbf{y}$ since the derivative of the dual function is nearly piecewise constant and the Newton iterates near a flat piece jump towards $\pm\infty$. As shown in [7], if $\mathbf{a} > \mathbf{0}$ and $\mathbf{u} = \infty$ or $\boldsymbol{\ell} = -\infty$, the variable fixing method and Newton's method can generate identical iterates. This result is generalized in Section 2.

In this paper we develop an algorithm called NAPHEAP built around a monotone break point searching algorithm implemented using a heap data structure. Given a starting guess λ for an optimal dual multiplier λ^* associated with the knapsack constraint, the break points on the side of λ containing λ^* are arranged in a heap. Building the heap requires about n comparisons, while updating the heap after removing a break point takes about $\log_2 n$ comparisons. Hence, if λ is separated from λ^* by m break points, then λ^* can be found using about $n + m \log_2 n$ comparisons. If m is small, then the heap-based algorithm is fast. One situation where a good starting guess is often available is when the gradient projection algorithm is applied to (1.4). In this situation, we compute a series of projections, and the multipliers associated with the linear constraint approach a limit. Hence, the multiplier arising in one projection is usually a good starting guess for the multiplier associated with the next projection. Currently the only algorithms which are able to take advantage of such a starting guess are the secant-based algorithm and the semi-smooth Newton method. In [7], numerical comparisons were made between algorithms for solving sequences of projection problems arising in Support Vector Machine applications. When hot starts are allowed, the Newton method was shown to be faster than the other methods.

We use the expression "Newton-type method" to refer to the class of methods which includes the variable fixing algorithm, Newton's method, and the secant method. If the knapsack problem is not connected with a convergent algorithm like the gradient projection algorithm, a few iterations of a Newton-type method may yield a good starting guess. Let $\mathbf{x}^*$ denote an optimal solution of (1.1), assuming it exists. When $O(n)$ components of $\mathbf{x}^*$ satisfy $\ell_i < x_i^* < u_i$, each iteration of a Newton-type method requires $O(n)$ operations. Even if the multiplier iterates are very close to λ^*,

the time for each iteration is still $O(n)$. On the other hand, NAPHEAP only requires $\log n$ comparisons as it passes by each break point on the path to λ^*. As we will show in the numerical experiments, a hybrid algorithm in which we perform between 1 and 4 iterations of a Newton-type method followed by the heap-based method (until convergence) can be much faster than a Newton-type method by itself.

Our paper is organized as follows. In Section 2 we give an overview of algorithms for solving (1.3) when $\mathbf{D}$ is positive definite, and we identify the elements these algorithms have in common. Section 3 studies the complexity of Newton-type methods, both in theory and in experiments. An example is constructed that shows the worst-case running time of a variable fixing algorithm could grow like n^2. Section 4 presents our heap-based algorithm for solving (1.3) in the case where the diagonal matrix $\mathbf{D}$ is positive semidefinite. Finally, Section 5 compares the performance of our hybrid NAPHEAP algorithm to Newton-type methods.

Notation: $\mathbf{0}$ and $\mathbf{1}$ denote vectors whose entries are all 0 and all 1 respectively, the dimensions should be clear from context. If $\mathcal{S}$ is a set, then $|\mathcal{S}|$ denotes the number of elements in $\mathcal{S}$, and $\mathcal{S}^c$ is the complement of $\mathcal{S}$. A subscript k is often used to denote the iteration number. Thus $\mathbf{x}_k$ is the k-th iterate and x_{ki} is the i-th component of the k-th iterate.

2. Overview of algorithms. For ease of exposition, it is assumed throughout the paper that $\mathbf{a} > \mathbf{0}$. Note that if $a_i = 0$, then x_i does not appear in the knapsack constraint and the optimal x_i is any solution of

$$\min\{.5x_i^2 d_i - y_i x_i : \ell_i \le x_i \le u_i\},$$

and if $a_i < 0$, then we can make the change of variables $z_i = -x_i$ to obtain an equivalent problem with $a_i > 0$. In this section, we also assume that $\mathbf{d} > \mathbf{0}$, while in the next section we take into account vanishing diagonal elements.

The common approach to (1.3) is to solve the dual problem. Let $\mathcal{L}$ denote the Lagrangian defined by

$$\mathcal{L}(\mathbf{x}, \lambda) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{x} - \mathbf{y}^\mathsf{T}\mathbf{x} + \lambda(\mathbf{a}^\mathsf{T}\mathbf{x} - b),$$

and let L denote the dual function

$$L(\lambda) = \min\{\mathcal{L}(\mathbf{x}, \lambda) : \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}\}. \tag{2.1}$$

The dual problem is

$$\max_{\lambda \in \mathbb{R}} \quad L(\lambda). \tag{2.2}$$

Since $\mathbf{d} > \mathbf{0}$, it follows that L is differentiable (see [6, Thm. 2.1] or [11]), and

$$L'(\lambda) = \mathbf{a}^\mathsf{T}\mathbf{x}(\lambda) - b,$$

where $\mathbf{x}(\lambda)$ is the unique minimizer in (2.1) given by

$$x_i(\lambda) = \mathrm{mid}(\ell_i, (y_i - \lambda a_i)/d_i, u_i),\ i = 1, 2, \ldots, n, \tag{2.3}$$

where $\mathrm{mid}(a, b, c)$ denotes the median of a, b, and c. The following result is well-known (for example, see [4]):

PROPOSITION 2.1. *Suppose* $\mathbf{d} > \mathbf{0}$ *and* (1.3) *is feasible.*

1. *L is concave and L' is non-increasing, continuous, and piecewise linear with break points given by the set*

$$\mathbf{\Lambda} := \bigcup_{1\le i\le n} \left\{ \frac{y_i - \ell_i d_i}{a_i}, \frac{y_i - u_i d_i}{a_i} \right\}. \tag{2.4}$$

2. *(2.2) has a solution $\lambda^* \in \mathbb{R}$ and $L'(\lambda^*) = 0$.*
3. *If $L'(\lambda^*) = 0$, then $\mathbf{x}(\lambda^*)$ defined in (2.3) is the unique solution of (1.3).*

Even in the case where λ is not an optimal multiplier, one can still use the sign of $L'(\lambda)$ to determine some bound components of the optimal solution to (1.3) (for example, see [28, Thm. 6]):

PROPOSITION 2.2. *If $\mathbf{d} > \mathbf{0}$, $\mathbf{a} > \mathbf{0}$, and $\mathbf{x}^*$ is a solution of (1.3), then for any $\lambda \in \mathbb{R}$, we have the following:*

1. *If $L'(\lambda) \ge 0$, then $x_i^* = \ell_i$ for every i such that $x_i(\lambda) = \ell_i$.*
2. *If $L'(\lambda) \le 0$, then $x_i^* = u_i$ for every i such that $x_i(\lambda) = u_i$.*

Proof. If $L'(\lambda) \ge 0$, then since L' is non-increasing, it follows that $\lambda \le \lambda^*$ for any λ^* which satisfies $L'(\lambda^*) = 0$. If $x_i(\lambda) = \ell_i$, then by the definition of $\mathbf{x}(\lambda)$, we conclude that $(y_i - \lambda a_i)/d_i \le \ell_i$. And since $a_i > 0$ and $\lambda^* \ge \lambda$,

$$\frac{y_i - \lambda^* a_i}{d_i} \le \frac{y_i - \lambda a_i}{d_i} \le \ell_i.$$

This implies that

$$x_i^* = x_i(\lambda^*) = \mathrm{mid}(\ell_i, (y_i - \lambda^* a_i)/d_i, u_i) = \ell_i.$$

The case where $L'(\lambda) \le 0$ is treated similarly. ∎

The specialized algorithms that have been developed for (1.3) have the generic form shown in Figure 2.1. The algorithms start with an initial guess λ_1 for the optimal

---

Step 0. *Initialization*: Set $k = 1$ and $\mathcal{B}_1 = \emptyset$.
Step 1. *Iteration*: Generate λ_k.
Step 2. *Stopping Criterion*: If $L'(\lambda_k) = 0$, set $\mathbf{x}^* = \mathbf{x}(\lambda_k)$ and stop.
Step 3. *Variable fixing* (optional): If $L'(\lambda_k) > 0$, then

$$\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{i \in \mathcal{B}_k^c : x_i(\lambda_k) = \ell_i\} \text{ and } x_i^* = \ell_i \text{ if } i \in \mathcal{B}_{k+1} \setminus \mathcal{B}_k. \tag{2.5}$$

If $L'(\lambda_k) < 0$, then

$$\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{i \in \mathcal{B}_k^c : x_i(\lambda_k) = u_i\} \text{ and } x_i^* = u_i \text{ if } i \in \mathcal{B}_{k+1} \setminus \mathcal{B}_k. \tag{2.6}$$

Step 4. *Update*: Set $k = k + 1$ and go to Step 1.

FIG. 2.1. *Generic algorithm for the separable quadratic knapsack problem with $\mathbf{a} > 0$ (1.3)*

dual multiplier and update λ_k until $L'(\lambda_k) = 0$. The solution $\mathbf{x}^*$ to (1.3) is the vector $\mathbf{x}(\lambda_k)$ constructed using (2.3). In Step 3, which is optional, Proposition 2.2 may be used to "fix" the values of some components of $\mathbf{x}^*$ at each iteration.

Algorithms for solving (1.3) differ primarily in how they choose λ_k in Step 1. For example, in a break point search [16], λ_k is the next break point between λ_{k-1} and

λ^* if such a break point exists. Otherwise, $\lambda_k = \lambda^*$ is found by linear interpolation. In the median search methods of [4, 5, 20, 23], the iteration amounts to selecting λ_k as the median of the remaining break points. Based on the sign of L' at the median, half of the remaining break points can be discarded. In the secant-based algorithm of Dai and Fletcher [10], λ_k is the root of a secant based on the value of L' at two previous iterates (with additional modifications for speeding up convergence).

In the variable fixing methods [1, 3, 18, 21, 24, 27, 28], each iteration $k \geq 2$ solves a subproblem over the remaining unfixed variables:

$$\min \left\{ \sum_{i \in \mathcal{F}_k} \frac{1}{2} d_i x_i^2 - y_i x_i : \sum_{i \in \mathcal{F}_k} a_i x_i = b_k \right\}, \tag{2.7}$$

where $\mathcal{F}_k = \mathcal{B}_k^c$ and

$$b_k = b - \sum_{i \in \mathcal{B}_k} a_i x_i^*. \tag{2.8}$$

The iterate λ_k is the optimal multiplier associated with the linear constraint in (2.7), and is given by

$$\lambda_k^F = \frac{-b_k + \sum_{i \in \mathcal{F}_k} a_i y_i / d_i}{\sum_{i \in \mathcal{F}_k} a_i^2 / d_i}. \tag{2.9}$$

While the method may begin with an arbitrary guess $\lambda_1 \in \mathbb{R}$, many implementations compute λ_1 using the formula (2.9) with $\mathcal{F}_1 = \{1, 2, \ldots, n\}$.

The recent method of Cominetti, Mascarenhas, and Silva [7] updates the multiplier by taking a semi-smooth Newton step in the direction of λ^*:

$$\lambda_k^N = \lambda_{k-1} - \frac{L'(\lambda_{k-1})}{L''_{\pm}(\lambda_{k-1})}, \; k \geq 2, \tag{2.10}$$

for some starting guess $\lambda_1 \in \mathbb{R}$. Here, $L''_{\pm}(\lambda_{k-1})$ denotes either the right or left side derivative of L' at λ_{k-1} (the side is chosen according to the sign of L'). Equivalently, λ_k^N is the maximizer of the second-order Taylor series approximation to L at λ_{k-1} in the direction of λ^* (in the case where λ_{k-1} is a break point, the Taylor series approximation depends on the direction in which we expand). If the Newton step is unacceptably large, then λ_k is either computed using a secant approximation or by simply moving to the next break point in the direction of λ^*. These modifications also prevent the iterates from cycling.

The Newton algorithm [7] and the variable fixing algorithm [18] are closely related. Both algorithms employ the variable fixing operation, and the iterate λ_k has the property that $\lambda_k \in (\alpha_{k-1}, \beta_{k-1})$, where

$$\begin{aligned} \alpha_j &= \sup\{\lambda_i : L'(\lambda_i) > 0 \text{ and } i \leq j\} \quad \text{and} \\ \beta_j &= \inf\{\lambda_i : L'(\lambda_i) < 0 \text{ and } i \leq j\}, \end{aligned} \tag{2.11}$$

where we use the convention that $\sup \emptyset = \infty$ and $\inf \emptyset = -\infty$. For the variable fixing algorithm, this is a consequence of the formula (2.9) – see Lemma 4.1 in [18]. For Newton scheme in [7], this property is ensured by replacing the Newton iterate by a secant iterate when the Newton iterate is not contained in $(\alpha_{k-1}, \beta_{k-1})$.

Any algorithm of the form of the generic algorithm in Figure 2.1 that includes the variable fixing step and which produces iterates $\lambda_k \in (\alpha_{k-1}, \beta_{k-1})$, also has the property that

$$x_i(\lambda_k) = x_i^* \text{ for all } i \in \mathcal{B}_{k+1}. \tag{2.12}$$

In particular, if $i \in \mathcal{B}_{k+1} \setminus \mathcal{B}_k$, then by (2.5)–(2.6), $x_i(\lambda_k) = x_i^*$. If $i \in \mathcal{B}_k$, then it entered $\mathcal{B}_k$ at an earlier iteration from either of the updates (2.5) or (2.6). To be specific, suppose that for some $j < k$, we have $L'(\lambda_j) > 0$ and $x_i(\lambda_j) = \ell_i = x_i^*$. Since α_{k-1} is expressed as a maximum in (2.11), it follows that $\lambda_j \le \alpha_{k-1}$. Since $a_i > 0$, $x_i(\lambda)$ is a decreasing function of λ with $x_i(\lambda) \ge \ell_i$ for all λ. It follows that

$$x_i^* = \ell_i = x_i(\lambda_j) \ge x_i(\alpha_{k-1}) \ge x_i(\lambda_k) \ge \ell_i,$$

which yields (2.12). The case $L'(\lambda_j) < 0$ and $x_i(\lambda_j) = u_i = x_i^*$ is similar.

Another interesting connection between these two algorithms, established below, is that both λ_k^F and λ_k^N maximize a *quadratic* approximation to L of the form

$$L_{\mathcal{B}}(\lambda) = \min\{\mathcal{L}(\mathbf{z}, \lambda) : \mathbf{z} \in \mathbb{R}^n, z_i = x_i(\lambda_{k-1}) \text{ for all } i \in \mathcal{B}\}, \tag{2.13}$$

for some choice of $\mathcal{B} \subset \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$, where $\mathcal{A}(\mathbf{x})$ denotes the set of active indices:

$$\mathcal{A}(\mathbf{x}) = \{i : x_i = \ell_i \text{ or } x_i = u_i\}$$

LEMMA 2.3. *Assume either that $k = 1$, or that $k \ge 2$ and $L'(\lambda_{k-1}) \ne 0$. Then the variable fixing iterate λ_k^F in (2.9) is the unique maximizer of $L_{\mathcal{B}_k}(\lambda)$.*

Proof. By (2.12), $x_i(\lambda_{k-1}) = x_i^*$ for all $i \in \mathcal{B}_k$ (if $k = 1$, then $\mathcal{B}_k = \emptyset$, so this is vacuously true). Hence, $L_{\mathcal{B}_k}$ can be expressed

$$L_{\mathcal{B}_k}(\lambda) = \min\{\mathcal{L}(\mathbf{z}, \lambda) : \mathbf{z} \in \mathbb{R}^n, z_i = x_i^* \text{ for all } i \in \mathcal{B}_k\}.$$

Consequently, $L_{\mathcal{B}_k}$ is the dual function associated with the optimization problem

$$\min\left\{\frac{1}{2}\mathbf{z}^\mathsf{T}\mathbf{D}\mathbf{z} - \mathbf{y}^\mathsf{T}\mathbf{z} : \mathbf{a}^\mathsf{T}\mathbf{z} = b, \quad z_i = x_i^* \text{ for all } i \in \mathcal{B}_k\right\}. \tag{2.14}$$

Since either $k = 1$ or $L'(\lambda_{k-1}) \ne 0$, we have that $\mathcal{F}_k$ is nonempty; hence, the maximizer of $L_{\mathcal{B}_k}$ is unique since it is strongly convex. Since the optimization problem (2.7) is the same as (2.14), the maximizer of $L_{\mathcal{B}_k}$ is the same as the optimal multiplier associated with the linear constraint of (2.14), which is the same as the optimal multiplier associated with the linear constraint of (2.7). □

Next, let us relate the Newton iterate to the maximizer of a dual function $L_{\mathcal{B}}$ for some choice of $\mathcal{B}$.

LEMMA 2.4. *Let $k \ge 2$ and suppose that $L'(\lambda_{k-1}) \ne 0$. Let $\mathcal{A}_0$ be defined by*

$$\mathcal{A}_0 := \left\{i : \frac{y_i - \lambda_{k-1}a_i}{d_i} = \ell_i \text{ if } L'(\lambda_{k-1}) < 0 \text{ and } \frac{y_i - \lambda_{k-1}a_i}{d_i} = u_i \text{ if } L'(\lambda_{k-1}) > 0\right\}.$$

Then λ_k^N is the unique maximizer of $L_{\mathcal{B}}(\lambda)$, where $\mathcal{B} = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))\setminus\mathcal{A}_0$.

Proof. Assume without loss of generality that $L'(\lambda_{k-1}) > 0$. By definition, λ_k^N is the maximizer of the second-order Taylor series

$$L(\lambda_{k-1}) + L'(\lambda_{k-1})(\lambda - \lambda_{k-1}) + \frac{1}{2}L''_+(\lambda_{k-1})(\lambda - \lambda_{k-1})^2. \tag{2.15}$$

Hence, we need only show that for $\mathcal{B} = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))\backslash\mathcal{A}_0$, $L_{\mathcal{B}}(\lambda)$ is equivalent to the expression (2.15). Since $L_{\mathcal{B}}(\lambda)$ is a quadratic function of λ, we need only show that $L_{\mathcal{B}}(\lambda_{k-1}) = L(\lambda_{k-1})$, $L'_{\mathcal{B}}(\lambda_{k-1}) = L'(\lambda_{k-1})$, and $L''_{\mathcal{B}}(\lambda_{k-1}) = L''_{+}(\lambda_{k-1})$.

For each λ, let $\mathbf{z}(\lambda)$ denote the unique solution to (2.13). We claim that $\mathbf{z}(\lambda_{k-1}) = \mathbf{x}(\lambda_{k-1})$. If $i \in \mathcal{B}$, then $z_i(\lambda_{k-1}) = x_i(\lambda_{k-1})$ by the constraint in (2.13). If $i \in \mathcal{B}^c$, then either $i \in \mathcal{A}(\mathbf{x}(\lambda_{k-1}))^c$ or $i \in \mathcal{A}_0$. In either case, it follows from (2.3) that

$$x_i(\lambda_{k-1}) = \frac{y_i - \lambda_{k-1}a_i}{d_i}.$$

By direct substitution, we obtain

$$\frac{\partial}{\partial x_i}\mathcal{L}(\mathbf{x}(\lambda_{k-1}), \lambda_{k-1}) = 0.$$

Hence, $\mathbf{x}(\lambda_{k-1})$ satisfies the first-order optimality conditions for (2.13), and by the strong convexity of the objective function $\mathbf{z}(\lambda_{k-1}) = \mathbf{x}(\lambda_{k-1})$. It follows immediately that $L_{\mathcal{B}}(\lambda_{k-1}) = L(\lambda_{k-1})$. Moreover, by [6, Thm. 2.1], we have

$$L'_{\mathcal{B}}(\lambda_{k-1}) = \mathbf{a}^{\mathsf{T}}\mathbf{z}(\lambda_{k-1}) - b = \mathbf{a}^{\mathsf{T}}\mathbf{x}(\lambda_{k-1}) - b = L'(\lambda_{k-1}).$$

Next, let us consider the second derivative of $L_{\mathcal{B}}$. We claim that for each i,

$$z'_i(\lambda_{k-1}) = x'_i(\lambda^+_{k-1}). \tag{2.16}$$

Indeed, if $i \in \mathcal{B}$, then $i \in \mathcal{A}(\mathbf{x}(\lambda_{k-1}))\backslash\mathcal{A}_0$, so by definition of $\mathcal{A}_0$, we have either

$$\frac{y_i - \lambda_{k-1}a_i}{d_i} \leq \ell_i \;\text{ or }\; \frac{y_i - \lambda_{k-1}a_i}{d_i} > u_i.$$

So, $x_i(\lambda) = u_i$ or ℓ_i if $\lambda \in [\lambda_k, \lambda_k + \epsilon]$ with $\epsilon > 0$ sufficiently small. Hence, in the case $i \in \mathcal{B}$, $z_i(\lambda)$ is constant on $\mathbb{R}$, and $z'_i(\lambda_{k-1}) = 0 = x'_i(\lambda^+_{k-1})$. On the other hand, if $i \in \mathcal{B}^c$, then either $i \in \mathcal{A}_0$ or $\ell_i < x_i(\lambda_{k-1}) < u_i$. In either case, we have

$$\ell_i < x_i(\lambda) = \frac{y_i - \lambda a_i}{d_i} \leq u_i$$

for $\lambda \in [\lambda_{k-1}, \lambda_{k-1} + \epsilon]$ and $\epsilon > 0$ Consequently, $z'_i(\lambda_{k-1}) = -a_i/d_i = x'_i(\lambda^+_{k-1})$. This completes the proof of the claim (2.16). Thus, we have

$$L''_{\mathcal{B}}(\lambda_{k-1}) = [\mathbf{a}^{\mathsf{T}}\mathbf{z}(\lambda_{k-1}) - b]' = [\mathbf{a}^{\mathsf{T}}\mathbf{x}(\lambda^+_{k-1}) - b]' = L''_{+}(\lambda_{k-1}),$$

which completes the proof. □

The next proposition shows that the iterates λ^F_k and λ^N_k coincide whenever the active components of $\mathbf{x}(\lambda_{k-1})$ coincide with $\mathcal{B}_k$.

THEOREM 2.5. *Let $k \geq 2$ and assume that $L'(\lambda_{k-1}) \neq 0$. If $\mathcal{B}_k = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$, then $\lambda^N_k = \lambda^F_k$.*

Proof. Assume without loss of generality that $L'(\lambda_{k-1}) > 0$. We prove that $\mathcal{A}_0 = \emptyset$ by contradiction, where $\mathcal{A}_0$ is defined in Lemma 2.4. If $i \in \mathcal{A}_0$, then $i \in \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$ and

$$x_i(\lambda_{k-1}) = \frac{y_i - \lambda_{k-1}a_i}{d_i} = u_i.$$

Since $a_i > 0$, it follows that $x_i(\lambda) < x_i(\lambda_{k-1}) = u_i$ for every $\lambda > \lambda_{k-1}$. Hence, x_i^* cannot have been fixed by the end of iteration $k-1$, and $i \notin \mathcal{B}_k$. Since $i \in \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$ but $i \notin \mathcal{B}_k$, we contradict the assumption that $\mathcal{B}_k = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$. Therefore, $\mathcal{A}_0 = \emptyset$, and by Lemma 2.4, λ_k^N is the unique maximizer of $\mathcal{L}_\mathcal{B}$ for $\mathcal{B} = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$. By Lemma 2.3, λ_k^F maximizes $\mathcal{L}_\mathcal{B}$ for $\mathcal{B} = \mathcal{B}_k$. So since $\mathcal{B}_k = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$, $\lambda_k^N = \lambda_k^F$. □

Remark. Theorem 2.5 generalizes Proposition 5.1 in [7] in which the semi-smooth Newton iterates are shown to be equivalent to the variable fixing iterates in the case where λ_1 satisfies (2.9), $\mathbf{a} > \mathbf{0}$, and $\mathbf{u} = \infty$. In this case, it can be shown that the sequence λ_k is monotonically increasing towards λ^*, $x_i(\lambda_k)$ is monotonically decreasing, and whenever a variable reaches a lower bound, it is fixed; that is, at each iteration k, we have $\mathcal{B}_k = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))$. By Theorem 2.5, $\lambda_k^F = \lambda_k^N$.

3. Worst case performance of Newton-type methods. In this section, we examine the worst case performance of the Newton-type methods such as the semi-smooth Newton method, the variable fixing method, and the secant method. All of these methods require the computation of $L'(\lambda_k)$ in each iteration. Since this computation involves a sum over the free indices, it follows that if $\Omega(n)$ components of an optimal solution are free, then each iteration of a Newton-type method requires $\Omega(n)$ flops. Here $\Omega(n)$ denotes a number bounded from below by cn for some $c > 0$.

Table 3.1 shows the performance of the variable fixing algorithm for a randomly selected example from Problem Set 1 of Section 5 of size $n = 3,000,000$. For each iteration, we give the CPU time for that iteration in seconds, the value of λ_k, and the size of the sets $\mathcal{F}_k = \mathcal{B}_k^c$ and $\mathcal{B}_{k+1} \setminus \mathcal{B}_k$. The algorithm converges after 10 iterations. However, after only 5 iterations, the relative error between λ_k and λ^* is already within 0.7%. The time for iteration 10 is smaller than the rest since the stopping condition was satisfied before completing the iteration. For iterations 5 through 9, the time per iteration is about 0.03 s and the number of free variables is on the order of 1 million. In Table 3.2 we solve the same problem associated with Table 3.1, but with the good

k	CPU (s)	λ_k	$\lvert\mathcal{F}_k\rvert$	$\lvert\mathcal{B}_{k+1}\setminus\mathcal{B}_k\rvert$
1	.190	0.2560789	2153491	846509
2	.066	1.5665073	1895702	257789
3	.066	3.3274053	1608655	287047
4	.054	4.4810929	1248067	360588
5	.032	3.8210237	1179116	68951
6	.030	3.8630132	1146762	32354
7	.029	3.8475087	1143152	3610
8	.029	3.8476129	1142346	806
9	.028	3.8476022	1142331	15
10	.014	3.8476022	1142331	0

TABLE 3.1
Statistics for the variable fixing algorithm applied to an example from Problem Set 1

starting guess $\lambda_1 = 3.84760$, which agrees with the exact multiplier to 6 significant digits. This starting guess is so good that all the components of the optimal solution that are at the upper bound can be fixed in the first iteration. Nonetheless, the variable fixing algorithm still took 8 iterations to reach the optimal solution, and the time for the trailing iterations is still around 0.03 s when the number of free variables is on the order of 1 million.

k	CPU (s)	λ_k	$\lvert\mathcal{F}_k\rvert$	$\lvert\mathcal{B}_{k+1}\backslash\mathcal{B}_k\rvert$
1	.127	3.8476000	2606250	393750
2	.096	0.1763559	1775119	831131
3	.051	1.3610706	1540751	234368
4	.045	2.6162104	1321817	218934
5	.036	3.5998977	1176249	145568
6	.030	3.8380911	1143634	32615
7	.029	3.8475885	1142330	1304
8	.014	3.8476022	1142330	0

TABLE 3.2

Statistics for the variable fixing algorithm applied to the same example shown in Table 3.1 but with a very good starting guess

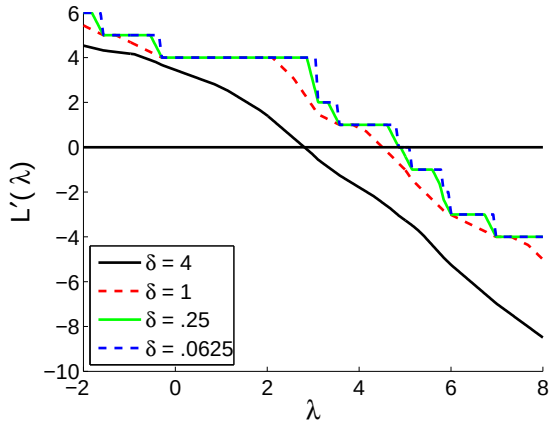

FIG. 3.1. *A plot of L' for the problems described in* (3.1) *with four different values for δ.*

Although Newton's method, starting from the good guess, would converge in 1 iteration on the problem of Table 3.1, it still requires $\Omega(n)$ flops per iteration. The good starting guess helps Newton's method by reducing the number of iterations, but not the time per iteration. Newton's method may also encounter convergence problems when there are small diagonal elements. To illustrate the effect of small diagonal elements, we consider a series of knapsack problems of the following form:

$$d_i = \delta, \quad a_i = 1, \quad y_i = \text{rand } [-10, 10], \quad \ell_i = 0, \quad u_i = 1. \tag{3.1}$$

Here rand $[-10, 10]$ denotes a random number between -10 and 10. The series of problems depends on the parameter δ. In Figure 3.1 we plot L' for four different values of δ. When $\delta = 4$, the plot is approximately linear, and Newton's method should find the root quickly. However, when δ decreases to 1, the plot develops a flat spot, and any Newton iterate landing on this flat spot would be kicked out toward $\pm\infty$. When δ reaches 0.0625, the graph is essentially piecewise constant, and in this case, Newton's method would not work well.

To correct for this poor performance, the authors of [7] implement a safeguard; whenever the Newton iterate lies outside the interval $(\alpha_{k-1}, \beta_{k-1})$, the multiplier update is computed by either moving to the next break point in the direction of λ^*, or using a secant step. In either case, convergence may be quite slow. For example,

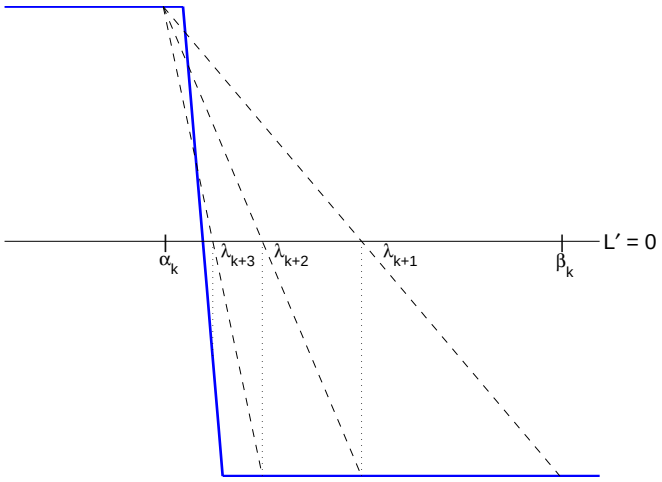

FIG. 3.2. *Potential secant iterates when L' is essentially piecewise constant*

Figure 3.2 shows that when the graph of L' is essentially piecewise constant, the convergence of a secant iteration based on the function values at α_{k-1} and β_{k-1} can be slow. Similar difficulties may be encountered when using a secant method like the one given by Dai and Fletcher [10]. In their algorithm, significant modifications had to be introduced in order to speed up convergence, particularly for problems where L' was nearly piecewise constant.

As the data in Table 3.1 indicates, for certain randomly generated problems, Newton-type methods require a small number of iterations to reach the solution. On the other hand, the worst case complexity could be $O(n^2)$ if the iteration is performed in exact arithmetic. An example demonstrating this worst case complexity for both Newton's algorithm and the variable fixing algorithm was given in [7], where the authors provide an L' for which n Newton iterations are needed to find the root. Here we provide another example that is expressed in terms of the knapsack problem itself. Let us consider the following special case of problem (1.3):

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{x} - \mathbf{1}^{\mathsf{T}}\mathbf{x} \quad \text{subject to} \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{1} \text{ and } \mathbf{1}^{\mathsf{T}}\mathbf{x} = 0. \tag{3.2}$$

Consider any lower bound $\boldsymbol{\ell}$ of the following form:

$$\ell_1 = \epsilon_1, \;\; \ell_i = \epsilon_1 + \sum_{j=2}^{i} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)} \;\; \text{for all } i \ge 2, \tag{3.3}$$

where ϵ_i is an arbitrary sequence that satisfies $1 > \epsilon_1 > \epsilon_2 > \ldots > \epsilon_n = 0$.

LEMMA 3.1. *The components of $\boldsymbol{\ell}$ given in* (3.3) *satisfy the following:*

1. $1 > \ell_1 > \ell_2 > \ldots > \ell_n$
2. *For each $i \ge 2$, the following relation holds:*

$$\ell_i = \frac{-\sum_{m=1}^{i-1} \ell_m}{n-i+1} + \frac{\epsilon_i}{\prod_{m=1}^{i-1}(n-m)}. \tag{3.4}$$

Proof.

Part 1: Since the sequence ϵ_i is strictly decreasing and non-negative, it follows that for any $2 \leq j \leq n$, we have

$$\epsilon_j - (n-j+2)\epsilon_{j-1} < \epsilon_j - \epsilon_{j-1} < 0.$$

Hence, the terms in the sum in (3.3) are all negative, which implies that the sequence ℓ_i is strictly decreasing and $\ell_1 = \epsilon_1 < 1$.

Part 2: Substituting ℓ_m using (3.3), we obtain

$$\sum_{m=1}^{i-1} \ell_m = (i-1)\epsilon_1 + \sum_{m=1}^{i-1} \sum_{j=2}^{m} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)}. \tag{3.5}$$

Notice that for each $2 \leq j \leq n$, the term $\frac{\epsilon_j-(n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)}$ appears exactly $(i-j)$ times in the sum of (3.5). Hence, equation (3.5) simplifies to

$$\sum_{m=1}^{i-1} \ell_m = (i-1)\epsilon_1 + \sum_{j=2}^{i-1} \left(\frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)} \right) (i-j).$$

We make this substitution on the right side of (3.4), the substitution (3.3) on the left, and multiply by $n-i+1$ to obtain

$$\epsilon_1(n-i+1) + \sum_{j=2}^{i} \left(\frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)} \right) (n-i+1) =$$
$$-\epsilon_1(i-1) - \sum_{j=2}^{i-1} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)}(i-j) + \frac{\epsilon_i}{\prod_{k=1}^{i-2}(n-k)}.$$

In the special case $i = 2$, this relation remains valid if the products are treated as 1 and the sums are treated as 0 when the lower limit exceeds the upper limit. We rearrange this relation to get

$$n\epsilon_1 + \sum_{j=2}^{i} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-2}(n-k)} = \frac{\epsilon_i}{\prod_{k=1}^{i-2}(n-k)}. \tag{3.6}$$

Hence, (3.4) is equivalent to (3.6). We prove (3.6) by induction.

If $i = 2$, then the left hand side of (3.6) is $n\epsilon_1 + \epsilon_2 - n\epsilon_1 = \epsilon_2$, which equals the right hand side. Now suppose that (3.6) holds for some $i \geq 2$. By the induction hypothesis, we have

$$n\epsilon_1 + \sum_{j=2}^{i+1} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-2}(n-k)} = \frac{\epsilon_{i+1} - (n-i+1)\epsilon_i}{\prod_{k=1}^{i-1}(n-k)} + \frac{\epsilon_i}{\prod_{k=1}^{i-2}(n-k)}$$
$$= \frac{\epsilon_{i+1}}{\prod_{k=1}^{i-1}(n-k)}.$$

But this is exactly the assertion made by (3.6) for $i+1$. Hence, the induction step has been established. □

Inserting $i = n$ in (3.4), we conclude that $\mathbf{1}^\mathsf{T}\boldsymbol{\ell} = 0$ since $\epsilon_n = 0$. Hence, $\mathbf{x} = \boldsymbol{\ell}$ is the only feasible point for (3.2). We now show that if the variable fixing algorithm is

applied to the problem (3.2) with $\boldsymbol{\ell}$ chosen according to (3.3), then only one variable is fixed at a lower bound in each iteration. Since the optimal solution is $\mathbf{x}^* = \boldsymbol{\ell}$, n iterations are required. Since the time to perform iteration k is $\Omega(n-k)$, the total running time is $\Omega(n^2)$.

PROPOSITION 3.2. *At iteration k of the variable fixing algorithm applied to* (3.2) *with $\boldsymbol{\ell}$ chosen according to* (3.3)*, exactly one component of the optimal solution $\mathbf{x}^*$ is fixed, namely $x_k^* = \ell_k$.*

Proof. The proof is by induction on k. For $k = 1$, $\mathcal{F}_k = \{1, 2, \ldots, n\}$, $b_k = 0$, and the solution of the equality constrained reduced problem (2.7) is $\mathbf{x}_1 = \mathbf{0}$. Since $\ell_1 = \epsilon_1 > 0$, the first component of $\mathbf{x}_1$ violates the lower bound: $x_{11} = 0 < \ell_1$. On the other hand, we now show that $\ell_i < x_{1i} < u_i$ for $i \geq 2$. Since $\epsilon_2 < \epsilon_1$, we have

$$l_2 = \epsilon_1 + \frac{\epsilon_2 - n\epsilon_1}{n-1} < \epsilon_1 + \frac{\epsilon_1 - n\epsilon_1}{n-1} = \epsilon_1 - \epsilon_1 = 0 = x_{12}.$$

By part 1 of Lemma 3.1, ℓ_i is a strictly decreasing function of i. Hence, for all $i \geq 2$,

$$\ell_i \leq \ell_2 < 0 = x_{1i} < u_i = 1.$$

This implies that the first component of $\mathbf{x}_1$ is the only component that is fixed at iteration 1; moreover, it is fixed at the lower bound. So $\mathcal{B}_2 = \{1\}$ and $\mathbf{x}_1^* = \ell_1$. This completes the base case.

Proceeding by induction, suppose that for some $k \geq 2$, $\mathcal{B}_k = \{1, 2, \ldots, k-1\}$ and $x_i^* = \ell_i$ for all $i < k$. We will show that $\mathcal{B}_{k+1} = \{1, 2, \ldots, k\}$ and $x_k^* = \ell_k$. At iteration k, the reduced problem is

$$\min \left\{ \sum_{i=k}^{n} \frac{1}{2} x_i^2 - x_i : \sum_{i=k}^{n} x_i = -\sum_{i=1}^{k-1} \ell_i \right\} \tag{3.7}$$

The solution is

$$x_{ki} = \frac{-\sum_{m=1}^{k-1} \ell_m}{n-k+1}, \quad i \geq k. \tag{3.8}$$

By (3.4),

$$x_{ki} = \ell_k - \frac{\epsilon_k}{\prod_{m=1}^{k-1}(n-m)}, \quad i \geq k. \tag{3.9}$$

In particular, $x_{kk} < \ell_k$.

By the definition (3.3) of $\boldsymbol{\ell}$, we have

$$\ell_{k+1} = \ell_k + \frac{\epsilon_{k+1} - (n-k+1)\epsilon_k}{\prod_{j=1}^{k}(n-j)}.$$

Substituting for ℓ_k using (3.9) yields

$$\ell_{k+1} = x_{ki} + \frac{\epsilon_{k+1} - \epsilon_k}{\prod_{j=1}^{k}(n-j)} < x_{ki} \text{ for } i > k.$$

By Lemma 3.1, ℓ_i is strictly decreasing. Hence, for every $i > k$ we have

$$\ell_i \leq \ell_{k+1} < x_{ki} = x_{kk} < \ell_k < 1.$$

It follows that $x_k^* = \ell_k$ while $\ell_i < x_{ki} < u_i$ for $i > k$. So $\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{k\}$. This completes the induction step. □

Remark. Proposition 3.2 is a theoretical result in the sense that the computations must be performed with exact arithmetic; in finite precision arithmetic, the ℓ_i sequence quickly approaches a limit as is seen in Table 3.3.

i	$\ell_i \times 100$
1	99.000000000000000000
2	−0.010101010101010111
3	−1.000103071531643038
4	−1.010102072694119869
5	−1.010204092701331297
6	−1.010205144342275242
7	−1.010205155295680612
8	−1.010205155410966865
9	−1.010205155412193141
10	−1.010205155412206325
11	−1.010205155412206499
12	−1.010205155412206499

TABLE 3.3
Values of ℓ_i in double precision arithmetic when $n = 100$ and $\epsilon_i = 1 - (i/100)$.

Remark. If Newton's method is applied to (3.2) starting from the λ_1 iterate of the variable fixing algorithm, then it will generate exactly the same iterates as the variable fixing algorithm. The equivalence between Newton's method and the variable fixing method is based on Theorem 2.5. During the proof of Proposition 3.2, we showed that the solution $\mathbf{x}_k$ of the subproblem (3.7) possessed exactly one component x_{kk} that violated the lower bound constraint $x_{kk} \geq \ell_k$. If λ_k is the multiplier associated with the constraint in (3.7), then

$$x_i(\lambda_k) = x_k(\lambda_k) < \ell_k < \ell_i$$

for all $i < k$ since ℓ_i is a decreasing function of i by Lemma 3.1. Consequently, $\mathcal{A}(\lambda_k) = \mathcal{B}_{k+1}$, and by Theorem 2.5, Newton's method produces the same iterate as the variable fixing method. Hence, Newton's method has complexity $\Omega(n^2)$ when applied to (3.2).

4. NAPHEAP. This section develops our algorithm for solving the separable quadratic knapsack problem (1.3) in the case that $\mathbf{d} \geq \mathbf{0}$. Assuming $L(\lambda) > -\infty$ (that is, λ lies in the domain of L), the set of minimizers $\mathbf{X}(\lambda)$ for (2.1) is

$$\mathbf{X}(\lambda) = \arg\min\{\mathcal{L}(\mathbf{x}, \lambda) : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}. \tag{4.1}$$

Notice that if $d_i > 0$, then the i-th component of $\mathbf{x}(\lambda)$ is unique and is given by (2.3). If $d_i = 0$ and λ lies in the interior of the domain of the dual function, then

$$X_i(\lambda) = \arg\min\{(\lambda a_i - y_i)x_i : \ell_i \leq x_i \leq u_i\} = \begin{cases} \ell_i & \text{if} \quad a_i\lambda > y_i, \\ [\ell_i, u_i] & \text{if} \quad a_i\lambda = y_i, \\ u_i & \text{if} \quad a_i\lambda < y_i. \end{cases}$$

The interval $[\ell_i, u_i]$ corresponds to the break point $\lambda = y_i/a_i$. On either side of the break point, $X_i(\lambda)$ equals either ℓ_i or u_i. Hence, for any $\mathbf{d} \geq \mathbf{0}$, $X(\lambda)$ is a linear function of λ on the interior of an interval located between break points.

By [6, Thm. 2.1] or [11], the subdifferential of L can be expressed

$$\partial L(\lambda) = [L'(\lambda^+), L'(\lambda^-)] \tag{4.2}$$

where

$$L'(\lambda^+) = \min\{\mathbf{a}^\mathsf{T}\mathbf{x} - b : \mathbf{x} \in \mathbf{X}(\lambda)\} \quad \text{and} \quad L'(\lambda^-) = \max\{\mathbf{a}^\mathsf{T}\mathbf{x} - b : \mathbf{x} \in \mathbf{X}(\lambda)\}.$$

Since L is concave, it subdifferential is monotone; in particular, if $\lambda_1 < \lambda_2$, $g_1 \in \partial L(\lambda_1)$, and $g_2 \in \partial L(\lambda_2)$, then $g_1 \geq g_2$. The generalization of Proposition 2.1 is the following:

PROPOSITION 4.1. *If $\mathbf{d} \geq \mathbf{0}$ and there exists an optimal solution $\mathbf{x}^*$ of (1.3), then there exists a maximizer λ^* of the dual function, $0 \in \partial L(\lambda^*)$, and $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$.*

Proof. The existence of a maximizer λ^* of the dual function along with the optimality conditions for λ^* and $\mathbf{x}^*$ are well-known properties of a convex optimization problem (see [19, 25]). □

For any given λ and for all sufficiently small $\epsilon > 0$, the sets $\mathbf{X}(\lambda+\epsilon)$ and $\mathbf{X}(\lambda-\epsilon)$ are singletons. We define

$$\mathbf{X}(\lambda^+) := \lim_{\epsilon \to 0^+} \mathbf{X}(\lambda + \epsilon) \quad \text{and} \quad \mathbf{X}(\lambda^-) := \lim_{\epsilon \to 0^+} \mathbf{X}(\lambda + \epsilon).$$

The following proposition extends Proposition 2.2 to the case $\mathbf{d} \geq \mathbf{0}$:

PROPOSITION 4.2. *If $\mathbf{d} \geq \mathbf{0}$, $\mathbf{a} > \mathbf{0}$, and $\mathbf{x}^*$ is optimal in (1.3), then for any λ in the domain of L, we have the following:*

1. *If $L'(\lambda^+) \geq 0$, then $x_i^* = \ell_i$ for every i such that $X_i(\lambda^+) = \ell_i$.*
2. *If $L'(\lambda^-) \leq 0$, then $x_i^* = u_i$ for every i such that $X_i(\lambda^-) = u_i$.*

Proof. Since the proofs of parts 1 and 2 are similar, we only prove part 1. Let λ^* maximize L. By Proposition 4.1, $0 \in \partial L(\lambda^*)$. If $L'(\lambda^+) \geq 0$, it follows from the monotonicity of ∂L, that $\lambda \leq \lambda^*$.

Case 1. First suppose that $\lambda = \lambda^*$ and $X_i(\lambda^+) = \ell_i$. Since $0 \in \partial L(\lambda^*) = \partial L(\lambda) = [L'(\lambda^+), L'(\lambda^-)]$, we conclude that $L'(\lambda^+) \leq 0$. By assumption, $L'(\lambda^+) \geq 0$. So it follows that $L'(\lambda^+) = 0$. If $d_i > 0$, then $X_i(\lambda) = X_i(\lambda^+) = \ell_i$. So since $\lambda = \lambda^*$, $X_i(\lambda^*) = \ell_i$. And since $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$ by Proposition 4.1, $x_i^* = \ell_i$. If $d_i = 0$, then since $X_i(\lambda^+) = \ell_i$, it follows that $\lambda a_i - y_i = \lambda^* a_i - y_i \geq 0$. If $\lambda^* a_i - y_i > 0$, then $x_i^* = X_i(\lambda^*) = \ell_i$. If $\lambda a_i - y_i = 0$, then $X_i(\lambda) = X_i(\lambda^*) = [\ell_i, u_i]$. Since $\mathbf{x}^*$ is optimal in (1.3), we have $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$. If $x_i^* > \ell_i$ and $a_i > 0$, then $x_i^* - \epsilon > \ell_i$ for $\epsilon > 0$ sufficiently small. Hence, $\mathbf{x}^* - \epsilon\mathbf{e}_i \in \mathbf{X}(\lambda^*)$, where $\mathbf{e}_i$ is the i-th column of the identity. Since $a_i > 0$, $\mathbf{a}^\mathsf{T}(\mathbf{x}^* - \epsilon\mathbf{e}_i) - b = -a_i\epsilon < 0$; it follow from (4.2) that $-a_i\epsilon \in \partial L(\lambda^*)$. This contradicts the fact that $\partial L(\lambda^*) = [L'(\lambda^{*+}), L'(\lambda^{*-})]$ where $L'(\lambda^{*+}) = 0$. Consequently, $x_i^* = \ell_i$.

Case 2. Suppose that $\lambda < \lambda^*$ and $X_i(\lambda^+) = \ell_i$. If $d_i > 0$, then by (2.3), $\frac{y_i - \lambda a_i}{d_i} \leq \ell_i$. Since $a_i > 0$ and $\lambda < \lambda^*$, we have

$$\frac{y_i - \lambda^* a_i}{d_i} < \frac{y_i - \lambda a_i}{d_i} \leq \ell_i.$$

Hence, by (2.3), $x_i^* = \ell_i$. If $d_i = 0$, then since $X_i(\lambda^+) = \ell_i$ and $\lambda < \lambda^*$, we have

$$\lambda^* a_i - y_i > \lambda a_i - y_i \geq 0$$

since $a_i > 0$. Again, it follows that $X_i(\lambda^*) = \ell_i$. □

Note that when $\mathbf{d} = \mathbf{0}$, Newton's method is not defined since $L''_{\pm}(\lambda) = 0$. On the other hand, the variable fixing method does make sense when $\mathbf{d} = \mathbf{0}$ since we can set $d_i = \epsilon > 0$ in (2.9) and take the limit as ϵ tends to zero to obtain

$$\lambda_k = \frac{\sum_{i \in \mathcal{F}_k} a_i y_i}{\sum_{i \in \mathcal{F}_k} a_i^2}.$$

Hence, the variable fixing method can be used to estimate the optimal multiplier associated with the linear constraint even when $\mathbf{d} = \mathbf{0}$.

The NAPHEAP algorithm has three phases. In the initial phase, we check whether the problem is feasible, and if it is, then we compute an interval $[\alpha, \beta]$ containing an optimal multiplier λ^*. In the next phase, several iterations of a Newton-type method are used to shrink our interval $[\alpha, \beta]$ containing λ^*. In the final phase, the monotonic break point searching algorithm is used to find an optimal multiplier λ^*.

NAPHEAP phase 0. Evaluate the quantities

$$A = \inf\{\mathbf{a}^\mathsf{T}\mathbf{x} : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\} \quad \text{and} \quad B = \sup\{\mathbf{a}^\mathsf{T}\mathbf{x} : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}. \tag{4.3}$$

If $b \notin [A, B]$, then stop since the problem is infeasible. Otherwise, evaluate the quantities

$$\alpha = \max\left\{\frac{y_i}{a_i} : d_i = 0 \text{ and } u_i = \infty\right\},$$
$$\beta = \min\left\{\frac{y_i}{a_i} : d_i = 0 \text{ and } \ell_i = -\infty\right\},$$

where the maximum and the minimum are defined to be ∞ and $-\infty$ respectively when the arguments are empty. L is finite on the interval $[\alpha, \beta]$ and $\lambda^* \in [\alpha, \beta]$. Based on the value of $L'(0)$, and the value of $L'(\lambda_1)$ if a starting guess λ_1 is given, the interval containing λ^* may be reduced further.

NAPHEAP phase 1. Perform up to K iterations of a Newton-type method (variable fixing, Newton's method, or the secant method), updating the interval $[\alpha, \beta]$ after each iteration and fixing variables when possible in accordance with Proposition 4.2. Replace Newton's method by the secant method whenever the current iterate lies outside the interval containing λ^*.

NAPHEAP phase 2. Arrange the break points $\boldsymbol{\Lambda}$ of (2.4) that are contained in $[\alpha, \beta]$ in a heap. If $\partial L(\alpha)$ is closer to zero than $\partial L(\beta)$, then organize the heap so that the break points closest to α are at the top of the heap. Otherwise organize the heap so that the break points closest to β are at the top of the heap. March sequentially through the break points until finding λ^* for which $0 \in \partial L(\lambda^*)$. The solution of (1.3) is any $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$ for which $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$.

In our code, we employ binary heaps (a tree with two children per node). For a max heap, the break point associated with any node is greater than or equal to the break points of the children. The largest break point is the root of the tree. Similarly, for a min heap, the break point associated with any node is less than or equal to the break points of the children. The smallest break point is the root of the tree. Building a heap of size n requires about n comparisons. To update a heap by adding or deleting a new node takes about $\log_2 n$ comparisons. The running time of phase 0 and phase 1 is $O(n)$ since K is a fixed integer, independent of n. The running time of the updates in phase 2 is proportional to $m \log_2 n$, where m is the number of break points separating the starting point and λ^*. For details concerning the construction and updating of heaps, see [8].

5. Numerical results. We now investigate the performance of the NAPHEAP algorithm using the following set of test problems. In these test problems, a statement of the form $C \in [A, B]$ means that C is randomly chosen from the interval $[A, B]$.

1. $d_i \in (0, 25]$, a_i and $y_i \in [-25, 25]$, ℓ_i and $u_i \in [-15, 15]$.
2. $a_i \in [-25, 25]$; $y_i \in [a_i - 5, a_i + 5]$; $d_i \in [.5|a_i|, 1.5|a_i|]$; ℓ_i, $u_i \in [-15, 15]$.
3. $a_i \in [-25, 25]$, $y_i = a_i + 5$, $d_i = |a_i|$, ℓ_i, $u_i \in [-15, 15]$.
4. $a_i = 1$, $y_i \in [-10, 10]$, $d_i = 1$, $\ell_i = 0$, $u_i = 1$.
5. $a_i \in (0, 25] \cap \mathbb{Z}$, $y_i \in [-10, 10]$, $d_i = 1$, $\ell_i = 0$, $u_i = 1$.
6. $d_i \in (0, 25]$, $y_i \in [-25, 25]$, $a_i = 1$, $\ell_i = 0$, $u_i = \infty$.
7. $d_i \in (0, 10^{-6}]$, $y_i \in [-25, 25]$, $a_i = 1$, $\ell_i = 0$, $u_i = \infty$.
8. $d_i = 0$, $y_i \in [-25, 25]$, $a_i = 1$, $\ell_i = 0$, and $u_i \in [0, 1]$.

Problem sets 4, 5, and 6 arise in graph partitioning (see [12, 14]), in multilevel graph partitioning, and in quadratic resource allocation [1, 2, 9, 17], respectively. b in the constraint $\mathbf{a}^\mathsf{T}\mathbf{x} = b$ was chosen randomly in the interval $[A, B]$ given in (4.3) except when $A = 0$ and $B = \infty$. In this case, b was chosen randomly in $[1, 100]$. Since similar results were obtained for all problem dimensions, we focus on problems of dimension 6,250,000 for which the running times were on the order of 0.5 s. NAPHEAP was programmed in C and the experiments were performed on a Dell Precision T7500 with 96 GB memory and dual six core Intel Xeon Processors (3.46 GZ). Only one core was used for the experiments.

We first investigated the dependence of performance on the number K of Newton-type iterations. When K is 0, NAPHEAP is simply performing a monotone break point search starting from either α or β. As K becomes large, NAPHEAP is simply performing a Newton-type method. For each value of K, NAPHEAP solves the same set of 100 randomly generated problems. The CPU times are used to plot performance profiles. Figure 5.1 shows the plot corresponding to the variable fixing algorithm for the Newton-type method. The vertical axis gives the fraction P of problems for which any given method is within a factor τ of the best time. The percentage of the test problems for which a method is fastest is given on the left axis of the plot. As τ tends to infinity, the right side of the plot gives the percentage of the test problems that were successfully solved by each of the methods. In essence, the right side is a measure of an algorithm's robustness. Since the highest curve in a performance plot corresponds to the method giving the best performance, Figure 5.1 seems to indicate that $K = 4$ was the best choice. From $K = 0$ to $K = 4$, the performance improves, while for $K = 5$ and larger, the performance degrades.

The best choices of K are shown in Table 5.1 for each of the problem sets. For problems in set 6, the monotone break point search by itself was fastest; that is, $K = 0$ was optimal. In these problems, the optimal multiplier was near an extreme break point (largest or smallest). Hence, the monotone break point search starting from an end of the interval bracketing the optimal multiplier reached the solution quickly. A similar situation arises if a good starting guess λ_1 for the optimal multiplier is provided. In this case, phase 0 of NAPHEAP uses $L'(\lambda_1)$ to help determine the interval $[\alpha, \beta]$ containing the optimal multiplier. Typically, either $\alpha = \lambda_1$ or $\beta = \lambda_1$, and phase 2 of NAPHEAP starts from λ_1 and moves monotonically to the nearby optimal multiplier.

For problems in sets 1, 2, and 3, the variable fixing algorithm was slightly more effective than Newton's method. For problems 4 and 5, Newton's method was more effective. Observe that $K = 3$ or 4 seems most effective with the variable fixing algorithm, while $K = 2$ or 3 seems most effective for Newton's method. Note that

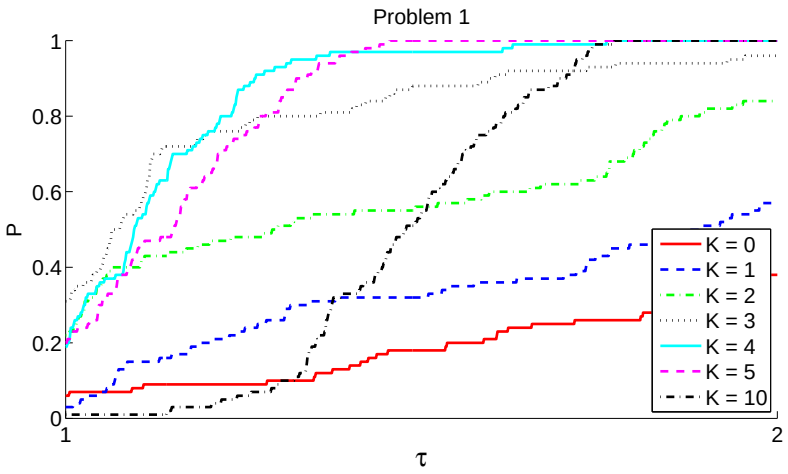

FIG. 5.1. *Performance profile based on time for NAPHEAP and Problem 1. K iterations of the variable fixing algorithm were performed before switching to the heap-based monotone break point search.*

Problem	Variable Fixing				Newton's Method			
Number	K	min	ave	max	K	min	ave	max
1	4	0.559	**0.659**	1.289	4	0.560	0.687	1.218
2	3	0.522	**0.560**	0.744	3	0.523	0.581	0.760
3	3	0.513	**0.540**	0.651	3	0.533	0.569	0.662
4	4	0.422	0.449	0.499	2	0.309	**0.350**	0.401
5	4	0.458	0.506	0.670	3	0.385	**0.463**	0.858
6	0	0.257	**0.263**	0.281	2	0.300	0.373	0.417
7	4	0.377	0.471	0.520	3	0.298	**0.369**	0.604
8	3	0.303	**0.347**	0.439				

TABLE 5.1

The best choice of K in NAPHEAP for each of the 8 test sets. For problem set 8, $\mathbf{d} = \mathbf{0}$ and Newton's method could not be applied. The minimum, maximum, and average CPU time in seconds is shown for 100 randomly generated problems of dimension $6,250,000$. The best average time for each problem set is in bold.

the starting point for Newton's method was generated by one iteration of the variable fixing algorithm. Hence, $K = 2$ or 3 in Newton's method really amounts to 3 or 4 iterations, the same as the optimal result for the variable fixing method. Although Table 5.1 indicates that Newton's method for problem 7 was most effective, it should be noted that problems in set 7 have small diagonal elements. In this case, it is easy for Newton iterates to go outside the interval $[\alpha, \beta]$, which causes the code to replace the Newton iterate by a secant iterate. The results for problem 7 essentially show that a few iterations of the secant method can be very effective, even when the diagonal elements are nearly 0.

6. Conclusion. We have presented a new hybrid algorithm NAPHEAP for the continuous quadratic knapsack problem (1.1) where $\mathbf{d} \geq \mathbf{0}$. The algorithm is based on maximizing the dual function L in (2.1), and computing the optimal multiplier associated with the linear constraint. The algorithm is developed in Section 4 where we are careful to take into account vanishing diagonal elements d_i. The algorithm is

based on an analysis of the break points; that is, points where L' is either discontinuous or has a discontinuous derivative. NAPHEAP arranges a subset of the break points surrounding the optimal multiplier in a heap, and then marches monotonically through the break points towards the optimal solution. If a good starting guess for the optimal multiplier is provided or if the optimal multiplier lies near one of the extreme break points, then the heap-based monotone break point search is very efficient by itself, as seen in Table 5.1, problem set 6. Otherwise, we observed in Section 5 (see Figure 5.1) that a good starting guess could be generated using 3 or 4 iterations of a Newton-type method.

The superior performance of NAPHEAP, when compared to a Newton-type method by itself, seems to be related to the fact that the operation count per iteration for a Newton-type method can be $\Omega(n)$, even when the iterates are very close to the optimum. On the other hand, a heap of size n can be built with about n comparisons, and it can be updated with about $\log_2 n$ comparisons. When there is a good starting guess, the heap size can be much smaller than n, and the number of heap updates can be small. The hybrid algorithm NAPHEAP can be a very efficient approach for solving the napsack problem (1.1), even in cases where some or all of the diagonal elements d_i are small or vanish.

REFERENCES

[1] G. Bitran and A. Hax, *Disaggregation and resource allocation using convex knapsack problems with bounded variables*, Manag. Sci., 27 (1981), pp. 431–441.

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

[3] K. Bretthauer, B. Shetty, and S. Syam, *A projection method for the integer quadratic knapsack problem*, J. Oper. Res. Soc., 47 (1996), pp. 457–462.

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

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

[6] F. H. Clarke, *Generalized gradients and applications*, Trans. Amer. Math. Soc., 205 (1975), pp. 247–262.

[7] R. Cominetti, W. F. Mascarenhas, and P. J. S. Silva, *A Newton's method for the continuous quadratic knapsack problem*, tech. rep., Optimization Online, August, 2012.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*, MIT Press, 2009.

[9] 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), pp. 94–111.

[10] Y. H. Dai and R. Fletcher, *New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds*, Math. Program., 106 (2006), pp. 403–421.

[11] J. M. Danskin, *The theory of max-min and its applications to weapons allocation problems*, Springer-Verlag, New York, 1967.

[12] W. W. Hager and J. T. Hungerford, *Optimality conditions for maximizing a function over a polyhedron, doi: 10.1007/s10107-013-0644-1*, Math. Program., (2013).

[13] ———, *A continuous quadratic programming formulation of the vertex separator problem*, European J. Oper. Res., (2013, submitted).

[14] W. W. Hager and Y. Krylyuk, *Graph partitioning and continuous quadratic programming*, SIAM J. Disc. Math., 12 (1999), pp. 500–523.

[15] W. W. Hager and H. Zhang, *A new active set algorithm for box constrained optimization*, SIAM J. Optim., 17 (2006), pp. 526–557.

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

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

[18] K. C. Kiwiel, *Variable fixing algorithms for the continuous quadratic knapsack problem*, J. Optim. Theory Appl., 136 (2008), pp. 445–458.

[19] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, Springer, Berlin, 2008.

[20] N. Maculan, C. Santiago, E. Macambira, and M. Jardim, *An O(n) algorithm for projecting a vector on the intersection of a hyperplane and a box in* $\mathbb{R}^n$, J. Optim. Theory Appl., 117 (2003), pp. 553–574.

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

[22] S. Nielsen and S. Zenios, *Massively parallel algorithms for singly-constrained convex programs*, ORSA J. Comput., 4 (1992), pp. 166–181.

[23] P. M. Pardalos and N. Kovoor, *An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds*, Math. Program., 46 (1990), pp. 321–328.

[24] A. G. Robinson, N. Jiang, and C. S. Lerme, *On the continuous quadratic knapsack problem*, Math. Program., 55 (1992), pp. 99–108.

[25] R. T. Rockafellar, *Convex analysis*, Princeton Univ. Press, 1970.

[26] B. Shetty and R. Muthukrishnan, *A parallel projection for the multicommodity network model*, J. Oper. Res. Soc., 41 (1990), pp. 837–842.

[27] N. Shor, *Minimization methods for nondifferentiable functions*, Springer-Verlag, New York, 1985.

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