An Efficient Hybrid Algorithm for the Separable Convex Quadratic Knapsack Problem

TIMOTHY A. DAVIS, University of Florida
WILLIAM W. HAGER, University of Florida
JAMES T. HUNGERFORD, University of Florida

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 computed using the variable fixing 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 dual function, and computing the solution of the dual problem. Also, a new version of the variable fixing algorithm is developed that is convergent even when the objective Hessian is not strictly positive definite. The hybrid algorithm NAPHEAP that uses a Newton-type method (variable fixing method, secant method, or Newton's method) to bracket a root, followed by a heap-based monotone break point search, can be faster than a Newton-type method by itself, as demonstrated in the numerical experiments.

Categories and Subject Descriptors: Mathematics of Computing [**Continuous optimization**]: Quadratic programming

General Terms: Algorithms; Theory

Additional Key Words and Phrases: continuous quadratic knapsack, nonlinear programming, convex programming, quadratic programming, separable programming, heap

ACM Reference Format:
Timothy A. Davis, William W. Hager, and James T. Hungerford, 2015. *ACM Trans. Math. Softw.* 9, 4, Article 39 (March 2010), 25 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

The authors gratefully acknowledge support by the Office of Naval Research under grants N00014-11-1-0068 and N00014-15-1-2048, by the National Science Foundation under grants 0620286 and 1522629, and by the Defense Advanced Research Project Agency under contract HR0011-12-C-0011. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

Author's addresses: T. A. Davis, (Current address) Computer Science and Engineering Department, Texas A&M University (davis@tamu.edu); William W. Hager, Mathematics Department, University of Florida (hager@ufl.edu); James T. Hungerford, (Current address) M.A.I.O.R. – Management, Artificial Intelligence, and Operations Research, Lucca, ITALY (jamesthungerford@gmail.com)

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

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

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

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) has applications in quadratic resource allocation [Bitran and Hax 1981; Bretthauer and Shetty 1997; Cosares and Hochbaum 1994; Hochbaum and Hong 1995], quasi-Newton updates with bounds [Calamai and Moré 1987], multicapacity network flow problems [Helgason et al. 1980; Nielsen and Zenios 1992; Shetty and Muthukrishnan 1990], continuous formulations of graph partitioning problems [Hager and Hungerford 2015; Hager and Krylyuk 1999], and support vector machines [Dai and Fletcher 2006].

By introducing an auxiliary variable $b \in \mathbb{R}$, we may reformulate (1) as:

$$\min_{\mathbf{x}\in\mathbb{R}^n,\, b\in\mathbb{R}} \{q(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad r \le b \le s, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\}. \tag{2}$$

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, (2) is transformed into the 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):

$$\min_{\mathbf{x}\in\mathbb{R}^n} \{q(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\}, \tag{3}$$

where $b \in \mathbb{R}$ is given. The constraints of (3) are continuous analogues of the constraints that arise in the discrete knapsack problem (see [Bretthauer et al. 1996]).

In quadratic resource allocation, problem (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 [Calamai and Moré 1987; Dai and Fletcher 2006; Hager and Hungerford 2015; Hager and Krylyuk 1999; Helgason et al. 1980; Nielsen and Zenios 1992; Shetty and Muthukrishnan 1990], an objective function F is minimized over a feasible set described by bound constraints and a single linear equality constraint:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \{F(\mathbf{x}) : \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b\} \tag{4}$$

The gradient projection algorithm as formulated in [Hager and Zhang 2006] starts with an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$ to a solution of (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 $s_k \geq 0$ is the stepsize, $\alpha_k \mathbf{I}$ is an approximation to the inverse Hessian of F, and proj($\mathbf{z}$) is the unique projection, relative to the 2-norm, of $\mathbf{z}$ onto the feasible set of (4). That is, proj($\mathbf{z}$) is the solution of the problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \left\{ \frac{1}{2}\|\mathbf{x}-\mathbf{z}\| : \quad \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \qquad \mathbf{a}^\mathsf{T}\mathbf{x} = b \right\}. \tag{5}$$

This projection problem is a special case of Problem (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 (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 [Helgason et al. 1980] 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 [Brucker 1984], linear time algorithms were proposed based on a median search rather than a sort. Subsequent developments of the median search method include those in [Calamai and Moré 1987; Maculan et al. 2003; Pardalos and Kovoor 1990].

In [Bitran and Hax 1981], Bitran and Hax proposed a method for solving a generalization of (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 (3) can be found in [Bretthauer et al. 1996; Kiwiel 2008; Michelot 1986; Robinson et al. 1992; Shor 1985; Ventura 1991]. An efficient and reliable implementation of the variable fixing method for (3), and a thorough convergence analysis, is given by Kiwiel in [Kiwiel 2008].

In [Dai and Fletcher 2006], 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 [Cominetti et al. 2014], 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 using a standard test set of randomly generated problems showed that the semi-smooth Newton method was faster than the variable fixing method, the secant method, and a median based method. As we explain in Section 3, Newton's method is not very well suited for problems where the elements of $\mathbf{d}$ are small relative to elements of $\mathbf{y}$ since the derivative of the dual function is nearly piecewise constant, and when a Newton iterate lands on a flat segment, it jumps towards $\pm\infty$. As shown in [Cominetti et al. 2014], 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 which is built around a monotone break point search implemented using a heap data structure. Given an interval (α, β) containing an optimal dual multiplier λ^* associated with the knapsack constraint, the break points on (α, β) are arranged in a heap. If there are m break points, then building the heap requires about m comparisons, while updating the heap after removing or adding a break point takes about $\log_2 m$ comparisons. Hence, if ei-

ther α or β is separated from λ^* by l break points, then λ^* can be found using about $m + l \log_2 m$ 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 the nonlinear optimization problem (4). In this situation, we compute the projection $\text{proj}(\mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k))$ to obtain the search direction $\mathbf{p}_k$. If λ_k is the Lagrange multiplier associated with the linear constraint in the projection problem $\text{proj}(\mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k))$, then it is observed in [Fu and Dai 2010] that a good guess for the Lagrange multiplier in the projection problem $\text{proj}(\mathbf{x}_{k+1} - \alpha_{k+1} \nabla F(\mathbf{x}_{k+1}))$ at iteration $k+1$ is $\alpha_{k+1}\lambda_k/\alpha_k$. In particular, they observe that this starting guess could reduce the computing time by 40% when compared to the starting guess λ_k at iteration $k+1$. Currently, algorithms which are able to take advantage of a good starting guess include the secant-based algorithm and the semi-smooth Newton method. In [Cominetti et al. 2014], 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), assuming it exists. When on the order of n components of $\mathbf{x}^*$ satisfy $\ell_i < x_i^* < u_i$, the time for an iteration of a Newton-type method is proportional to n, even if the iterates start very close to λ^*. On the other hand, the time to pass over a break point in a step of NAPHEAP is proportional to $\log_2 m$, where m is the number of break points in an interval bracketing a solution. As we will show in the numerical experiments, a hybrid algorithm which uses a Newton-type method to generate a starting guess followed by heap-based search (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 (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 develops a generalization of the variable fixing algorithm that can be used even when some components of $\mathbf{d}$ vanish. Section 5 presents our heap-based algorithm for solving (3). Finally, Section 6 compares the performance of our hybrid NAPHEAP algorithm to the Newton method from [Cominetti et al. 2014] using randomly generated test problems. The structure of the dual function is investigated in an effort to understand the implications of numerical results based on randomly generated problems.

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 (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

$$L(\lambda) = \min\{\mathcal{L}(\mathbf{x}, \lambda) : \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}\}. \tag{6}$$

The dual problem is

$$\max\{L(\lambda) : \lambda \in \mathbb{R}\}. \tag{7}$$

Since $\mathbf{d} > \mathbf{0}$, it follows that L is differentiable (see [Clarke 1975, Thm. 2.1] or [Danskin 1967]), and

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

where $\mathbf{x}(\lambda)$ is the unique minimizer in (6) given by

$$x_i(\lambda) = \mathbf{mid}(\ell_i, (y_i - \lambda a_i)/d_i, u_i),\ i = 1, 2, \ldots, n, \tag{8}$$

and $\mathrm{mid}(a, b, c)$ denotes the median (or middle) of a, b, and c. The following result is well-known (for example, see [Brucker 1984]):

PROPOSITION 2.1. *Suppose* $\mathbf{d} > \mathbf{0}$ *and* (3) *is feasible.*

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

$$\mathbf{\Lambda} := \bigcup_{1 \le i \le n} \left\{ \frac{y_i - \ell_i d_i}{a_i}, \frac{y_i - u_i d_i}{a_i} \right\}. \tag{9}$$

(*2*) (7) *has a solution* $\lambda^* \in \mathbb{R}$ *and* $L'(\lambda^*) = 0$.
(*3*) *If* $L'(\lambda^*) = 0$, *then* $\mathbf{x}(\lambda^*)$ *defined in* (8) *is the unique solution of* (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 (3) (for example, see [Ventura 1991, Thm. 6]):

PROPOSITION 2.2. *If* $\mathbf{d} > \mathbf{0}$, $\mathbf{a} \ge \mathbf{0}$, *and* $\mathbf{x}^*$ *is a solution of* (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

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

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

ALGORITHM 1: Solve a separable quadratic knapsack problem (3) with $\mathbf{d} > \mathbf{0}$ and $\mathbf{a} > \mathbf{0}$.

Input: Vectors $\mathbf{a}$, $\mathbf{d}$, $\mathbf{y}$, $\boldsymbol{\ell}$, $\mathbf{u}$ and scalar b defining the problem (3); initial guess λ_1
Output: The solution vector $\mathbf{x}$
$k = 1$; $\mathcal{L}_1 = \emptyset$; $\mathcal{U}_1 = \emptyset$;
while *true* **do**
  Step 1: Generate λ_k (use initial guess if $k = 1$)
  Step 2: *Stopping Criterion:*
  if $L'(\lambda_k) = 0$ **then**
    $\mathbf{x}^* = \mathbf{x}(\lambda_k)$
    break
  end
  Step 3: *Variable fixing (optional):*
  Define $\mathcal{F}_k = (\mathcal{U}_k \cup \mathcal{L}_k)^c$.
  if $L'(\lambda_k) > 0$ **then**

$$\mathcal{L}_{k+1} = \mathcal{L}_k \cup \{i \in \mathcal{F}_k : x_i(\lambda_k) = \ell_i\} \text{ and } \mathcal{U}_{k+1} = \mathcal{U}_k. \tag{10}$$

  end
  if $L'(\lambda_k) < 0$ **then**

$$\mathcal{U}_{k+1} = \mathcal{U}_k \cup \{i \in \mathcal{F}_k : x_i(\lambda_k) = u_i\} \text{ and } \mathcal{L}_{k+1} = \mathcal{L}_k. \tag{11}$$

  end
  $k = k + 1$
end

The specialized algorithms that have been developed for (3) have the generic form shown in Algorithm 1. The algorithms start with an initial guess λ_1 for the optimal dual multiplier and update λ_k until $L'(\lambda_k) = 0$. The solution $\mathbf{x}^*$ to (3) is the vector $\mathbf{x}(\lambda_k)$ constructed using (8). In particular,

$$x_i^* = \begin{cases} \ell_i & \text{if } i \in \mathcal{L}_k, \\ u_i & \text{if } i \in \mathcal{U}_k, \\ \text{mid}(\ell_i,\ (y_i - \lambda_k a_i)/d_i,\ u_i) & \text{otherwise.} \end{cases}$$

In Step 3, Proposition 2.2 may be used to "fix" the values of some components of $\mathbf{x}^*$ at each iteration.

Algorithms for solving (3) differ primarily in how they choose λ_k in Step 1. For example, in a break point search [Helgason et al. 1980], 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 [Brucker 1984; Calamai and Moré 1987; Maculan et al. 2003; Pardalos and Kovoor 1990], 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 [Dai and Fletcher 2006], 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 [Bitran and Hax 1981; Bretthauer et al. 1996; Kiwiel 2008; Michelot 1986; Robinson et al. 1992; Shor 1985; Ventura 1991], each iteration $k \geq 2$ solves a subproblem over the remaining unfixed variables:

$$\min \left\{ \sum_{i \in \mathcal{F}_k} \frac{1}{2} d_i x_i^2 - y_i x_i : \sum_{i \in \mathcal{F}_k} a_i x_i = b_k \right\}. \tag{12}$$

Here, $\mathcal{F}_k = \mathcal{B}_k^c$, where $\mathcal{B}_k = \mathcal{L}_k \cup \mathcal{U}_k$, and

$$b_k = b - \sum_{i \in \mathcal{B}_k} a_i x_i^*. \tag{13}$$

Note that the constraint $\ell_i \le x_i \le u_i$ is dropped in (12). The iterate λ_k is the optimal multiplier associated with the linear constraint in (12), and is given by

$$\lambda_k^F = \frac{-b_k + \sum_{i \in \mathcal{F}_k} a_i y_i / d_i}{\sum_{i \in \mathcal{F}_k} a_i^2 / d_i}. \tag{14}$$

While the method may begin with an arbitrary guess $\lambda_1 \in \mathbb{R}$, many implementations compute λ_1 using the formula (14) with $\mathcal{F}_1 = \{1, 2, \ldots, n\}$.

The recent method of Cominetti, Mascarenhas, and Silva [Cominetti et al. 2014] updates the multiplier by taking a semi-smooth Newton step in the direction of λ^*:

$$\lambda_k^N = \lambda_{k-1} - \frac{L'(\lambda_{k-1})}{L''_{\pm}(\lambda_{k-1})}, \quad k \ge 2, \tag{15}$$

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 [Cominetti et al. 2014] and the variable fixing algorithm [Kiwiel 2008] 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

$$\begin{aligned} \alpha_j &= \sup\{\lambda_i : L'(\lambda_i) > 0 \text{ and } i \le j\} \quad \text{and} \\ \beta_j &= \inf\{\lambda_i : L'(\lambda_i) < 0 \text{ and } i \le j\}, \end{aligned} \tag{16}$$

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 (14) – see Lemma 4.1 in [Kiwiel 2008]. For Newton scheme in [Cominetti et al. 2014], 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 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

$$x_i(\lambda_k) = x_i^* \text{ for all } i \in \mathcal{B}_{k+1}. \tag{17}$$

In particular, if $i \in \mathcal{B}_{k+1} \setminus \mathcal{B}_k$, then by (10)–(11), $x_i(\lambda_k) = x_i^*$. If $i \in \mathcal{B}_k$, then it entered $\mathcal{B}_k$ at an earlier iteration from either of the updates (10) or (11). 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 (16), 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 (17). 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

$$L_{\mathcal{B}}(\lambda) = \min\{\mathcal{L}(\mathbf{z}, \lambda) : \mathbf{z} \in \mathbb{R}^n, z_i = x_i(\lambda_{k-1}) \text{ for all } i \in \mathcal{B}\}, \tag{18}$$

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 $k \geq 2$ and $L'(\lambda_{k-1}) \neq 0$. Then the variable fixing iterate λ_k^F in (14) is the unique maximizer of $L_{\mathcal{B}_k}(\lambda)$.*

PROOF. By (17), $x_i(\lambda_{k-1}) = x_i^*$ 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

$$\min\left\{\frac{1}{2}\mathbf{z}^\mathsf{T}\mathbf{D}\mathbf{z} - \mathbf{y}^\mathsf{T}\mathbf{z} : \mathbf{a}^\mathsf{T}\mathbf{z} = b, \quad z_i = x_i^* \text{ for all } i \in \mathcal{B}_k\right\}. \tag{19}$$

Since either $k = 1$ or $L'(\lambda_{k-1}) \neq 0$, 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 (12) is the same as (19), the maximizer of $L_{\mathcal{B}_k}$ is the same as the optimal multiplier associated with the linear constraint of (19), which is the same as the optimal multiplier associated with the linear constraint of (12). □

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 $k \geq 2$ and suppose that $L'(\lambda_{k-1}) \neq 0$. 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 $\mathcal{B} = \mathcal{A}(\mathbf{x}(\lambda_{k-1}))\backslash\mathcal{A}_0$.

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

$$L(\lambda_{k-1}) + L'(\lambda_{k-1})(\lambda - \lambda_{k-1}) + \frac{1}{2}L''_+(\lambda_{k-1})(\lambda - \lambda_{k-1})^2. \tag{20}$$

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 (20). 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 (18). 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 (18). 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 (8) 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 (18), 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 [Clarke 1975, 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,

$$z_i'(\lambda_{k-1}) = x_i'(\lambda_{k-1}^+). \tag{21}$$

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 (21). 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_k^F and λ_k^N 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 $\lambda_k^N = \lambda_k^F$.*

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 2.6. Theorem 2.5 generalizes Proposition 5.1 in [Cominetti et al. 2014] in which the semi-smooth Newton iterates are shown to be equivalent to the variable fixing iterates in the case where λ_1 satisfies (14), $\mathbf{a} > \mathbf{0}$, 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 I. Statistics for the variable fixing algorithm applied to an example from Problem Set 1

k	CPU (s)	λ_k	$\lvert\mathcal{F}_k\rvert$	$\lvert\mathcal{B}_{k+1}\backslash\mathcal{B}_k\rvert$
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 II. Statistics for the variable fixing algorithm applied to the same example shown in Table I but with a very good starting guess

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 I shows the performance of the variable fixing algorithm for a randomly selected example from Problem Set 1 of Section 6 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 II we solve the same problem associated with Table I, but with the good 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.

Although Newton's method, starting from the good guess, would converge in 1 iteration on the problem of Table I, 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:

$$d_i = \delta, \quad a_i = 1, \quad y_i = \text{rand}\,[-10, 10], \quad \ell_i = 0, \quad u_i = 1. \tag{22}$$

Here rand $[-10, 10]$ denotes a random number between -10 and 10 with a uniform distribution. The series of problems depends on the parameter δ. In Figure 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

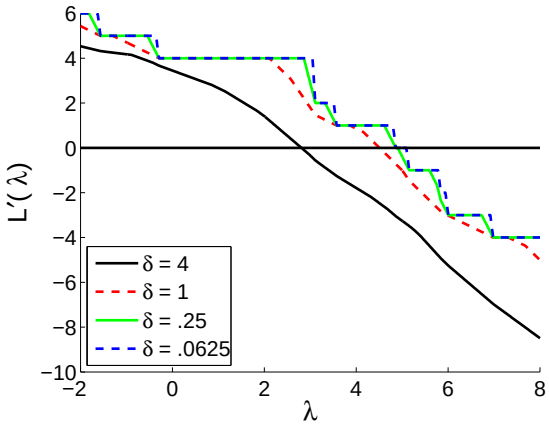

Fig. 1. A plot of L' for the problems described in (22) with four different values for δ.

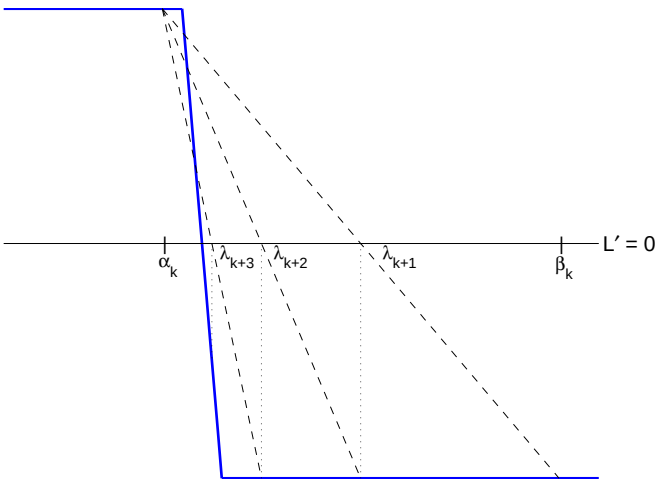

Fig. 2. Potential secant iterates when L' is essentially piecewise constant

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 [Cominetti et al. 2014] 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, Figure 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 [Dai and Fletcher 2006]. 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 I 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 [Cominetti et al. 2014], 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 (3):

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{x} - \mathbf{1}^\mathsf{T}\mathbf{x} \quad \text{subject to} \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{1} \text{ and } \mathbf{1}^\mathsf{T}\mathbf{x} = 0. \tag{23}$$

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

$$\ell_1 = \epsilon_1, \;\; \ell_i = \epsilon_1 + \textstyle\sum_{j=2}^{i} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)} \;\; \text{for all } i \ge 2, \tag{24}$$

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* (24) *satisfy the following:*

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

$$\ell_i = \frac{-\sum_{m=1}^{i-1} \ell_m}{n-i+1} + \frac{\epsilon_i}{\prod_{m=1}^{i-1}(n-m)}. \tag{25}$$

PROOF.

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

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

Hence, the terms in the sum in (24) 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 (24), we obtain

$$\sum_{m=1}^{i-1} \ell_m = (i-1)\epsilon_1 + \sum_{m=1}^{i-1}\sum_{j=2}^{m} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-1}(n-k)}. \tag{26}$$

Notice that for each $2 \le j \le n$, 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 (26). Hence, equation (26) 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 (25), the substitution (24) 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

$$n\epsilon_1 + \sum_{j=2}^{i} \frac{\epsilon_j - (n-j+2)\epsilon_{j-1}}{\prod_{k=1}^{j-2}(n-k)} = \frac{\epsilon_i}{\prod_{k=1}^{i-2}(n-k)}. \tag{27}$$

Hence, (25) is equivalent to (27). We prove (27) by induction.

If $i = 2$, then the left hand side of (27) is $n\epsilon_1 + \epsilon_2 - n\epsilon_1 = \epsilon_2$, which equals the right hand side. Now suppose that (27) holds for some $i \geq 2$. By the induction hypothesis, we have

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

But this is exactly the assertion made by (27) for $i+1$. Hence, the induction step has been established. □

Inserting $i = n$ in (25), 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 (23). We now show that if the variable fixing algorithm is applied to the problem (23) with $\boldsymbol{\ell}$ chosen according to (24), 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* (23) *with $\boldsymbol{\ell}$ chosen according to* (24)*, 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 (12) 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

$$\min \left\{ \sum_{i=k}^{n} \frac{1}{2} x_i^2 - x_i : \sum_{i=k}^{n} x_i = -\sum_{i=1}^{k-1} \ell_i \right\} \tag{28}$$

The solution is

$$x_{ki} = \frac{-\sum_{m=1}^{k-1} \ell_m}{n-k+1}, \quad i \geq k. \tag{29}$$

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

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

By (25),

$$x_{ki} = \ell_k - \frac{\epsilon_k}{\prod_{m=1}^{k-1}(n-m)}, \quad i \geq k. \tag{30}$$

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

By the definition (24) 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 (30) 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 3.3. 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 III.

Remark 3.4. If Newton's method is applied to (23) 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 (28) 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 (28), 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 (23).

4. VARIABLE FIXING WHEN SOME DIAGONAL ELEMENTS VANISH

When some components of $\mathbf{d}$ vanish, it may not be possible to apply Newton's method to the equation $L'(\lambda) = 0$ since $L''(\lambda)$ could vanish. In this section we develop a version of the variable fixing algorithm that can used even when one or more components of $\mathbf{d}$ vanish. Assuming $L(\lambda) > -\infty$ (that is, λ lies in the domain of L), the set of minimizers $\mathbf{X}(\lambda)$ for the dual function (6) is

$$\mathbf{X}(\lambda) = \arg\min\{\mathcal{L}(\mathbf{x}, \lambda) : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}. \tag{31}$$

If $d_i > 0$, then the i-th component of $\mathbf{X}(\lambda)$ is unique and is given by (8). The associated break points (9) are $(y_i - \ell_i d_i)/a_i$ and $(y_i - u_i d_i)/a_i$. Between the break points, $X_i(\lambda) = (y_i - \lambda a_i)/d_i$, while outside the break point interval, $X_i(\lambda) = \ell_i$ or u_i. If $d_i = 0$ and λ lies in the interior of the domain of the dual function, then

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

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 $\mathbf{d} \geq \mathbf{0}$, $\mathbf{X}(\lambda)$ is a linear function of λ on the interior of any interval located between break points.

By [Clarke 1975, Thm. 2.1] or [Danskin 1967], the subdifferential of L can be expressed

$$\partial L(\lambda) = [L'(\lambda^+), L'(\lambda^-)] \tag{32}$$

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, its 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 (3), then there exists a maximizer λ^* of the dual function, $0 \in \partial L(\lambda^*)$, and $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$. Moreover, any $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$ with $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$ is optimal in (3).*

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 concave optimization problem (see [Luenberger and Ye 2008; Rockafellar 1970]). □

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 (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 $\lambda \leq \lambda^*$.

ALGORITHM 2: Solve a separable quadratic knapsack problem (3) with $\mathbf{d} \geq \mathbf{0}$ and $\mathbf{a} > \mathbf{0}$.

Input: Vectors $\mathbf{a}$, $\mathbf{d}$, $\mathbf{y}$, $\boldsymbol{\ell}$, $\mathbf{u}$ and scalar b defining the problem (3); initial guess λ_1
Output: The solution vector $\mathbf{x}$
$k = 1$; $\mathcal{L}_1 = \emptyset$; $\mathcal{U}_1 = \emptyset$;
while *true* **do**
  Step 1: Generate λ_k (use initial guess if $k = 1$)
  Step 2: *Stopping Criterion:*
  if $0 \in \partial L_k(\lambda_k)$ **then**
    find $\mathbf{x}^* \in \mathbf{X}(\lambda_k)$ such that $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$, where

$$L_k(\lambda) := \min\{\mathcal{L}(\mathbf{x}, \lambda) : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \quad x_i = \ell_i \; \forall \; i \in \mathcal{L}_k, \quad x_i = u_i \; \forall \; i \in \mathcal{U}_k\}$$

    break
  end
  Step 3: *Variable fixing:*
  Define $\mathcal{F}_k = (\mathcal{U}_k \cup \mathcal{L}_k)^c$.
  if $L_k'(\lambda_k^+) > 0$ **then**
    $\mathcal{L}_{k+1} = \mathcal{L}_k \cup \{i \in \mathcal{F}_k : X_i(\lambda_k^+) = \ell_i\}$ and $\mathcal{U}_{k+1} = \mathcal{U}_k$.
  end
  if $L_k'(\lambda_k^-) < 0$ **then**
    $\mathcal{U}_{k+1} = \mathcal{U}_k \cup \{i \in \mathcal{F}_k : X_i(\lambda_k^-) = u_i\}$ and $\mathcal{L}_{k+1} = \mathcal{L}_k$.
  end
  $k = k + 1$
end

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 (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 (32) 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 (8), $\frac{y_i - \lambda a_i}{d_i} \leq \ell_i$. 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 (8), $x_i^* = \ell_i$. 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 $X_i(\lambda^*) = \ell_i$. □

Algorithm 2 is a generic approach for solving (3) in the case where $\mathbf{d} \geq \mathbf{0}$.

In Algorithm 2, the sets $\mathcal{L}_k$ and $\mathcal{U}_k$ store components whose optimal values have been determined to lie at a lower or upper bound, respectively. The minimizing argument of $L_k(\lambda)$ in Step 2 is given by

$$X_i^k(\lambda) = \begin{cases} X_i(\lambda) & \text{if } i \in \mathcal{F}_k, \\ \ell_i & \text{if } i \in \mathcal{L}_k, \\ u_i & \text{if } i \in \mathcal{U}_k. \end{cases}$$

As we will see in Lemma 4.4, L_k in Algorithm 2 may be replaced by L without affecting the algorithm. Hence, in the case $\mathbf{d} > \mathbf{0}$, Algorithm 2 is equivalent to Algorithm 1 since then $\partial L(\lambda) = L'(\lambda)$ and $\mathbf{X}(\lambda^{\pm}) = \mathbf{x}(\lambda)$ for all $\lambda \in \mathbb{R}$.

As in Algorithm 1, the convergence properties of Algorithm 2 depend on how λ_k is chosen in Step 1. Let us define the set

$$\mathcal{Z} = \{j : d_j = 0\}.$$

When $\mathcal{Z} \neq \emptyset$, the variable fixing iterates λ_k^F defined in (14) and the Newton iterates (15) may be invalid, due to division by zero. However, the variable fixing algorithm has a natural extension to the case $\mathcal{Z} \neq \emptyset$, as we will now show.

Given an instance of (3) with $\mathbf{d} \geq \mathbf{0}$, let $\epsilon > 0$ and define $\mathbf{d}^\epsilon$ by

$$d_i^\epsilon = \begin{cases} \epsilon & \text{if } i \in \mathcal{Z}, \\ d_i & \text{if } i \notin \mathcal{Z}, \end{cases}$$

$i = 1, 2, \ldots, n$. Consider the perturbed problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \left\{ \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{D}^\epsilon\mathbf{x} - \mathbf{y}^\mathsf{T}\mathbf{x} : \quad \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b \right\}, \tag{33}$$

where $\mathbf{D}^\epsilon$ is the diagonal matrix with $\mathbf{d}^\epsilon$ on the diagonal. Since $d_i^\epsilon > 0$ for every i, the kth variable fixing iterate for (33) is well-defined and is given by

$$\begin{aligned} \lambda_{k,\epsilon}^F &= \frac{-b_k + \sum_{i \in \mathcal{F}_k \setminus \mathcal{Z}} \frac{y_i a_i}{d_i} + \sum_{i \in \mathcal{F}_k \cap \mathcal{Z}} \frac{y_i a_i}{\epsilon}}{\sum_{i \in \mathcal{F}_k \setminus \mathcal{Z}} \frac{a_i^2}{d_i} + \sum_{i \in \mathcal{F}_k \cap \mathcal{Z}} \frac{a_i^2}{\epsilon}} \\ &= \frac{-\epsilon b_k + \epsilon \sum_{i \in \mathcal{F}_k \setminus \mathcal{Z}} \frac{y_i a_i}{d_i} + \sum_{i \in \mathcal{F}_k \cap \mathcal{Z}} y_i a_i}{\epsilon \sum_{i \in \mathcal{F}_k \setminus \mathcal{Z}} \frac{a_i^2}{d_i} + \sum_{i \in \mathcal{F}_k \cap \mathcal{Z}} a_i^2}. \end{aligned}$$

Hence when $\mathcal{Z} \cap \mathcal{F}_k \neq \emptyset$, we have

$$\lambda_{k,0}^F := \lim_{\epsilon \to 0^+} \lambda_{k,\epsilon}^F = \frac{\sum_{i \in \mathcal{Z} \cap \mathcal{F}_k} y_i a_i}{\sum_{i \in \mathcal{Z} \cap \mathcal{F}_k} a_i^2}.$$

This motivates the following theorem.

THEOREM 4.3. *Let $\mathbf{d} \geq \mathbf{0}$, $\mathbf{a} > \mathbf{0}$, and suppose that* (3) *has an optimal solution. For each iteration $k \geq 1$ of Algorithm 2, let λ_k be defined by*

$$\lambda_k = \begin{cases} \lambda_{k,0}^F & \text{if } \mathcal{Z} \cap \mathcal{F}_k \neq \emptyset, \\ \lambda_k^F & \text{if } \mathcal{Z} \cap \mathcal{F}_k = \emptyset. \end{cases} \tag{34}$$

Then Algorithm 2 converges in finitely many iterations to some λ^* such that $0 \in \partial L(\lambda^*)$. Hence, any $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$ such that $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$ is optimal in (3).

The proof of Theorem 4.3 is based on the following lemma.

LEMMA 4.4. *Let $\mathbf{d} \geq \mathbf{0}$, $\mathbf{a} > \mathbf{0}$, and suppose that* (3) *has an optimal solution. For each iteration $k \geq 1$, define*

$$\begin{aligned} \lambda_k^L &:= \sup\{\lambda_i : 1 \leq i \leq k-1,\ L_i'(\lambda_i^+) > 0\}, \\ \lambda_k^R &:= \inf\{\lambda_i : 1 \leq i \leq k-1,\ L_i'(\lambda_i^-) < 0\}, \end{aligned}$$

where $\sup\ \emptyset := -\infty$ *and* $\inf\ \emptyset := \infty$. *Then, for every $k \geq 1$ such that $\mathcal{Z} \cap \mathcal{F}_k \neq \emptyset$, the following hold:*

(1) $\lambda_k \in (\lambda_k^L, \lambda_k^R)$.
(2) If $0 \notin \partial L_k(\lambda_k)$*, then either* $|\mathcal{L}_{k+1}| > |\mathcal{L}_k|$ *or* $|\mathcal{U}_{k+1}| > |\mathcal{U}_k|$.
(3) $L_k(\lambda) = L(\lambda)$ *for every* $\lambda \in (\lambda_k^L, \lambda_k^R)$.
(4) For every $\lambda \in \mathbb{R}$ *with* $0 \in \partial L_k(\lambda)$*, we have* $0 \in \partial L(\lambda)$.

PROOF. **Parts 1 and 2:** Since $\mathcal{Z} \cap \mathcal{F}_k \neq \emptyset$, we have

$$\lambda_k = \frac{\sum_{i\in\mathcal{Z}\cap\mathcal{F}_k} y_i a_i}{\sum_{j\in\mathcal{Z}\cap\mathcal{F}_k} a_j^2} = \sum_{i\in\mathcal{Z}\cap\mathcal{F}_k} \frac{y_i}{a_i}\left(\frac{a_i^2}{\sum_{j\in\mathcal{Z}\cap\mathcal{F}_k} a_j^2}\right) = \sum_{i\in\mathcal{Z}\cap\mathcal{F}_k} \alpha_{i,k}\left(\frac{y_i}{a_i}\right),$$

where $\alpha_{i,k} = \frac{a_i^2}{\sum_{j\in\mathcal{Z}\cap\mathcal{F}_k} a_j^2} > 0$, and $\sum_{i\in\mathcal{Z}\cap\mathcal{F}_k} \alpha_{i,k} = 1$. Hence, λ_k is a convex combination of breakpoints associated with components in $\mathcal{Z} \cap \mathcal{F}_k$. In particular, there exist $i, j \in \mathcal{Z} \cap \mathcal{F}_k$ such that

$$\frac{y_i}{a_i} \leq \lambda_k \leq \frac{y_j}{a_j}. \tag{35}$$

Next, we claim that $\lambda_k^L < \frac{y_i}{a_i}$. Indeed, suppose by way of contradiction that this is not true. Then, by definition of λ_k^L, there is some (smallest) $l \leq k-1$ such that $L_l'(\lambda_l^+) > 0$ and $\lambda_l \geq \frac{y_i}{a_i}$. Since $\lambda_l \geq \frac{y_i}{a_i}$, $X_i(\lambda_l^+) = \ell_i$ by definition of $\mathbf{X}$, and on Step 3 of the lth iteration of Algorithm 2, $i \in \mathcal{L}_{l+1}\backslash\mathcal{L}_l \subseteq \mathcal{L}_k$, contradicting $i \in \mathcal{F}_k$. By a similar argument, $\frac{y_j}{a_j} < \lambda_k^R$. Hence, by (35) $\lambda_k^L < \lambda_k < \lambda_k^R$, which proves Part 1.

Now, suppose that $0 \notin \partial L_k(\lambda_k)$. Then by (32) either $L_k'(\lambda_k^+) > 0$ or $L_k'(\lambda_k^-) < 0$. Notice that (35) implies that $X_i(\lambda_k^+) = \ell_i$ and $X_j(\lambda_k^-) = u_j$, by definition of $\mathbf{X}$. Hence, if $L_k'(\lambda_k^+) > 0$, then by Step 3 of the kth iteration of Algorithm 2 we have $i \in \mathcal{L}_{k+1}\backslash\mathcal{L}_k$; whereas if $L_k'(\lambda_k^-) < 0$, then $j \in \mathcal{U}_{k+1}\backslash\mathcal{U}_k$. This completes the proof of Part 2.

Parts 3 and 4: We prove Parts 3 and 4 by induction on k. The base cases where $k = 1$ are trivial, since $L_1 = L$. So, suppose that for some $s \geq 1$, we have $\mathcal{Z} \cap \mathcal{F}_k \neq \emptyset$ and Parts 3 and 4 hold for every $k \leq s$. Suppose that $\mathcal{Z} \cap \mathcal{F}_{s+1} \neq \emptyset$. We will show that Parts 3 and 4 hold for $k = s + 1$.

Since the algorithm did not stop on iteration s, we must have $0 \notin \partial L_s(\lambda_s)$. We assume without loss of generality that $L_s'(\lambda_s^+) > 0$ (the proof in the case $L_s'(\lambda_s^-) < 0$ is similar). By the induction hypothesis, $L_s(\lambda) = L(\lambda)$ for any $\lambda \in (\lambda_s^L, \lambda_s^R) \supseteq (\lambda_{s+1}^L, \lambda_{s+1}^R)$. Hence, we will be done with Part 3 when we show that $L_{s+1}(\lambda) = L_s(\lambda)$ for every $\lambda \in (\lambda_{s+1}^L, \lambda_{s+1}^R)$.

Let $\lambda \in (\lambda_{s+1}^L, \lambda_{s+1}^R)$. Since $L_s'(\lambda_s^+) > 0$ by assumption, Step 3 of the algorithm gives $\mathcal{U}_{s+1} = \mathcal{U}_s$ and $\mathcal{L}_s \subset \mathcal{L}_{s+1}$. Hence, $X_i^{s+1}(\lambda) = X_i^s(\lambda) = u_i$ for all $i \in \mathcal{U}_{s+1}$ and $X_i^{s+1}(\lambda) = X_i^s(\lambda) = \ell_i$ for all $i \in \mathcal{L}_s$. Since $\mathcal{F}^{s+1} \subset \mathcal{F}^s$, it follows that $X_i^{s+1}(\lambda) = X_i^s(\lambda)$ for all $i \in \mathcal{F}_{s+1}$. Hence, the only indices where $X_i^{s+1}(\lambda)$ $X_i^s(\lambda)$ could differ are those $i \in \mathcal{L}_{s+1}\backslash\mathcal{L}_s$. We will now show that for every $i \in \mathcal{L}_{s+1}\backslash\mathcal{L}_s$, $X_i^s(\lambda) = \ell_i = X_i^{s+1}(\lambda)$. This will imply $\mathbf{X}^{s+1}(\lambda) = \mathbf{X}^s(\lambda)$, and therefore $L_{s+1}(\lambda) = L_s(\lambda)$.

Let $i \in \mathcal{L}_{s+1}\backslash\mathcal{L}_s$. Since $L_s'(\lambda_s^+) > 0$ by assumption, Step 3 of the algorithm implies

$$X_i(\lambda_s^+) = \ell_i. \tag{36}$$

By Part 1, $\lambda_s \in (\lambda_s^L, \lambda_s^R)$; hence by definition of λ_{s+1}^L,

$$\lambda_{s+1}^L = \lambda_s. \tag{37}$$

Since $\lambda > \lambda_{s+1}^L = \lambda_s$ and $X_i(\cdot)$ is nonincreasing and bounded below by ℓ_i, we have $X_i(\lambda) = \ell_i$. Thus, $X_i^s(\lambda) = X_i(\lambda) = \ell_i = X_i^{s+1}(\lambda)$.

Since $i \in \mathcal{L}_{s+1} \backslash \mathcal{L}_s$ was arbitrary, we have shown that $\mathbf{X}^{s+1}(\lambda) = \mathbf{X}^s(\lambda)$; hence, $L_{s+1}(\lambda) = L_s(\lambda)$. Since $\lambda \in (\lambda^L_{s+1}, \lambda^R_{s+1})$ was arbitrary, we have shown that

$$L_{s+1}(\lambda) = L_s(\lambda) = L(\lambda) \quad \forall\, \lambda \in (\lambda^L_{s+1}, \lambda^R_{s+1}), \tag{38}$$

which proves Part 3 for $k = s + 1$.

Now we prove Part 4 holds for $k = s + 1$. Suppose that $0 \in \partial L_{s+1}(\lambda)$ for some $\lambda \in \mathbb{R}$. Then by (32),

$$L'_{s+1}(\lambda^+) \le 0 \le L'_{s+1}(\lambda^-). \tag{39}$$

We claim that $\lambda \in (\lambda^L_{s+1}, \lambda^R_{s+1})$. Once this claim is proved, it will follow from (38) that $\partial L(\lambda) = \partial L_{s+1}(\lambda)$. Since $0 \in \partial L_{s+1}(\lambda)$, we will have $0 \in \partial L(\lambda)$, and the proof will be complete.

First, we show $\lambda^L_{s+1} < \lambda$. If $\lambda^L_{s+1} = -\infty$, then we are done. So, suppose $\lambda^L_{s+1} > -\infty$. Then

$$L'_{s+1}(\lambda^{L+}_{s+1}) = L'_s(\lambda^{L+}_{s+1}) = L'_s(\lambda^+_s) > 0, \tag{40}$$

where the first equality follows from (38), the second follows from (37), and the inequality follows from our assumption. Since $\partial L_{s+1}(\cdot)$ is monotone nonincreasing, (40) and the first half of (39) imply $\lambda^L_{s+1} < \lambda$.

Now, we show $\lambda < \lambda^R_{s+1}$. If $\lambda^R_{s+1} = \infty$, then we are done. So, suppose $\lambda^R_{s+1} < \infty$. Then $\lambda^R_{s+1} = \lambda_j$ for some $j \le s$ such that $L'_j(\lambda^-_j) < 0$. By (38) and the induction assumption for Part 3,

$$L_{s+1}(\lambda) = L(\lambda) = L_j(\lambda) \quad \forall\, \lambda \in (\lambda^L_{s+1}, \lambda^R_{s+1}) \subseteq (\lambda^L_j, \lambda^R_j).$$

Hence,

$$L'_{s+1}(\lambda^{R-}_{s+1}) = L'_j(\lambda^{R-}_{s+1}) = L'_j(\lambda^-_j) < 0. \tag{41}$$

So, since $\partial L(\cdot)$ is monotone nonincreasing, (41) and the second half of (39) imply $\lambda < \lambda^R_{s+1}$. Thus, $\lambda \in (\lambda^L_{s+1}, \lambda^R_{s+1})$, and our claim is proved. This completes the proof of Part 4 for $k = s + 1$.

Therefore, the proof of Parts 3 and 4 is complete by induction. □

Remark 4.5. By Lemma 4.4, we may replace L_k with L in Algorithm 2. Hence, in the case where $\mathcal{Z} = \emptyset$, Algorithm 2 is equivalent to Algorithm 1, since then $\partial L(\lambda) = L'(\lambda)$ and $\mathbf{X}(\lambda^\pm) = \mathbf{x}(\lambda)$ for all $\lambda \in \mathbb{R}$.

Proof of Theorem 4.3. By Part 2 of Lemma 4.4, since $|\mathcal{L}_k|$ and $|\mathcal{U}_k|$ are bounded from above by n, we eventually must reach an iteration $s \ge 1$ such that either

$$[\mathcal{Z} \cap \mathcal{F}_s = \emptyset] \quad \text{or} \quad [\mathcal{Z} \cap \mathcal{F}_s \ne \emptyset \ \text{ and } \ 0 \in \partial L_s(\lambda_s)]. \tag{42}$$

Let s be the first iteration satisfying (42). In the second case of (42), we have $0 \in \partial L(\lambda_s)$ by Part 3 of Lemma 4.4, and we are done. Now suppose instead that the first case holds. Then for $k \ge s$, $\lambda_k = \lambda^F_k$ and Algorithm 2 reduces to the Variable Fixing Algorithm of [Kiwiel 2008] for the problem

$$\begin{aligned} &\min \quad q(\mathbf{x}) \\ &\text{subject to} \quad \boldsymbol{\ell} \le \mathbf{x} \le \mathbf{u}, \quad \mathbf{a}^\mathsf{T}\mathbf{x} = b, \quad x_i = \ell_i \ \forall\, i \in \mathcal{L}_s, \quad \text{and} \quad x_i = u_i \ \forall\, i \in \mathcal{U}_s. \end{aligned}$$

(The formula for λ^F_k (14) comes directly from [Kiwiel 2008, page 448]). Therefore, by [Kiwiel 2008, Theorem 4.1], λ_k converges to some λ^* such that

$$\partial L_s(\lambda^*) = L'_s(\lambda^*) = \mathbf{a}_s^\mathsf{T}\mathbf{x}_s(\lambda^*) - b_s = 0,$$

where $\mathbf{a}_s$ and $\mathbf{x}_s(\lambda^*)$ represent the vectors formed by only including components in $\mathcal{F}_s$, and b_s is defined by

$$b_s = b - \sum_{i \in \mathcal{L}_s} a_i \ell_i - \sum_{i \in \mathcal{U}_s} a_i u_i. \tag{43}$$

Therefore, by Part 4 of Lemma 4.4, $0 \in \partial L(\lambda^*)$. This completes the proof.

5. NAPHEAP

In many problems, the iterates in a variable fixing method or Newton method exhibit a diminishing returns property: most of the progress towards finding the optimal multiplier is made in the first several iterations, while subsequent iterations yield steps which are much smaller. We propose an algorithm called NAPHEAP which exploits this property. The algorithm first generates an interval (α, β) (possibly semi-infinite) that brackets a solution of the dual problem (7) by applying a few iterations of a Newton-type method. Then all the break points of the dual function between α and β are arranged in heaps. NAPHEAP then starts from either α or β and monotonically visits the break points until reaching an optimal dual solution λ^*. Since L' is linear on any interval between break points, a solution of the dual problem can be evaluated by linear interpolation. If there are m break points in (α, β), then the time to update the heap after visiting a break point is proportional to $\log_2 m$. Similarly, the time to build a heap with m elements is roughly the time to perform m comparisons. Hence, when l break points are visited on the path to the solution, the total running time of the heap phase is proportional to $m + l \log_2 m$. For details concerning the construction and updating of heaps, see [Cormen et al. 2009]. The steps of the NAPHEAP code are partitioned into three phases that we now discuss in more detail.

NAPHEAP phase 0. If $a_i = 0$ for some i, then the optimal value for x_i can evaluated, and x_i can be eliminated from the problem. Hence, without loss of generality, we assume that $a_i \neq 0$ for all i. If $d_i = 0$ for some i, then we evaluate the expressions

$$\alpha_0 = \max\left\{ \frac{y_i}{a_i} : d_i = 0 \text{ and } u_i = \infty \right\} \quad \text{and} \quad \beta_0 = \min\left\{ \frac{y_i}{a_i} : d_i = 0 \text{ and } \ell_i = -\infty \right\},$$

where the maximum and the minimum are defined to be ∞ and $-\infty$ respectively when the arguments are empty. L is finite on the interval $[\alpha_0, \beta_0]$, the domain of the dual function, and an optimal dual solution λ^* lies between α_0 and β_0. If a starting guess λ_1 was not provided, then we evaluate one by performing a single iteration of the variable fixing algorithm. From the sign of $L'(\lambda_1)$, and the values of α_0 and β_0, we can further refine the bracketing interval (α, β).

NAPHEAP phase 1. The bracketing interval generated in phase 0 is further refined by performing a few iterations of a Newton-type method. The optimal number of iterations depends on the problem. When $d_i = 0$ for some i, Version 2.1 of NAPHEAP currently performs a user specified K Newton-type iterations before switching to the break point search. However, when $\mathbf{d} > \mathbf{0}$, an adaptive strategy is employed based on the following observation: Newton's method often converges monotonically to the solution of the dual problem. Hence, if we multiply the Newton step by a scaling factor $\xi > 1$ (default 1.1), then the scaled Newton iterate

$$\lambda_{k+1} = \lambda_k - \xi \left(\frac{L'(\lambda_k)}{L''_{\pm}(\lambda_k)} \right)$$

will typically lie on the opposite side of the root from λ_k when λ_k is sufficiently close to a solution. We continue to perform a scaled Newton iteration until we reach the opposite side of the root, and then we terminate phase 1. When the scaled Newton iterate

lands outside the current bracketing interval, then we attempt a secant iteration. If the bracketing interval is semi-infinite and the secant iteration cannot be performed, then we use a variable fixing iteration. We impose an upper bound on the number of Newton-type iterations (default 20).

NAPHEAP phase 2. Phase 1 either generates the solution of the knapsack problem, or it generates an interval $[\alpha, \beta]$ (possibly semi-infinite) that brackets a solution of the dual problem; either α or β was generated by the last iteration of phase 1. All the break points between α and β are evaluated and arranged in a heap. Starting from the last iterate of phase 1, we monotonically cross the break points until reaching a solution of the dual problem. The solution of (3) is any $\mathbf{x}^* \in \mathbf{X}(\lambda^*)$ for which $\mathbf{a}^\mathsf{T}\mathbf{x}^* = b$.

The heaps that we employ are binary heaps which are trees where each node in the tree has two children. The nodes correspond to break points that lie between α and β. If we start from α and move to the right on the λ axis, then we employ a min-heap; that is, a heap for which the root is the smallest break point and the break points associated with the children of any node are greater than or equal to the break point of the parent. As we move through the break points from left to right, the current root of the tree always contains the next break point. In a similar manner, if we start from β and move left, then the relevant break points are arranged in a max-heap since we need to know the largest of the remaining break points.

In more detail, two separate heaps are maintained. The free heap at the current iterate λ_k consists of break points for those indices i satisfying $\ell_i < x_i(\lambda_k) < u_i$. These break points are values of λ in the direction of λ^* where $x_i(\lambda)$ reaches either ℓ_i or u_i. The bound heap corresponds to those i for which $x_i(\lambda_k) = \ell_i$ or $x_i(\lambda_k) = u_i$ at the current iterate λ_k. The break points in the bound heap are values of λ in the direction of λ^* where $x_i(\lambda)$ leaves a bound and becomes free. When we cross over a break point for an index i in the free heap, $x_i(\lambda)$ reaches a bound, and it cannot leave this bound since there are no more break points in the direction of λ^*. Hence, this break point can be deleted from the free heap and the bound value for x_i^* is known. When we cross over a break point for an index i in the bound heap, $x_i(\lambda)$ becomes free, however, when $d_i > 0$, there is another break point in the direction of λ^* where $x_i(\lambda)$ reaches the opposite bound. This new break point is inserted in the free heap. When $d_i = 0$, there is only one break point y_i/a_i. If this break point lies between λ_k and λ^*, then it belongs in the bound heap. When we reach this break point in the search process, $x_i(\lambda)$ makes a transition from one bound to the opposite bound. In either case, we continue to march across the break points until reaching a point λ^* where $0 \in \partial L(\lambda^*)$.

6. 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 $[A, B]$ with a uniform probability distribution.

(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) $d_i \in (0, 10^{-6}]$, $y_i \in [-25, 25]$, $a_i = 1$, $\ell_i = 0$, $u_i = \infty$.

Problem sets 4, 5, and 6 are related to graph partitioning (see [Hager and Hungerford 2014; Hager and Krylyuk 1999]), multilevel graph partitioning [Hager et al. 2014],

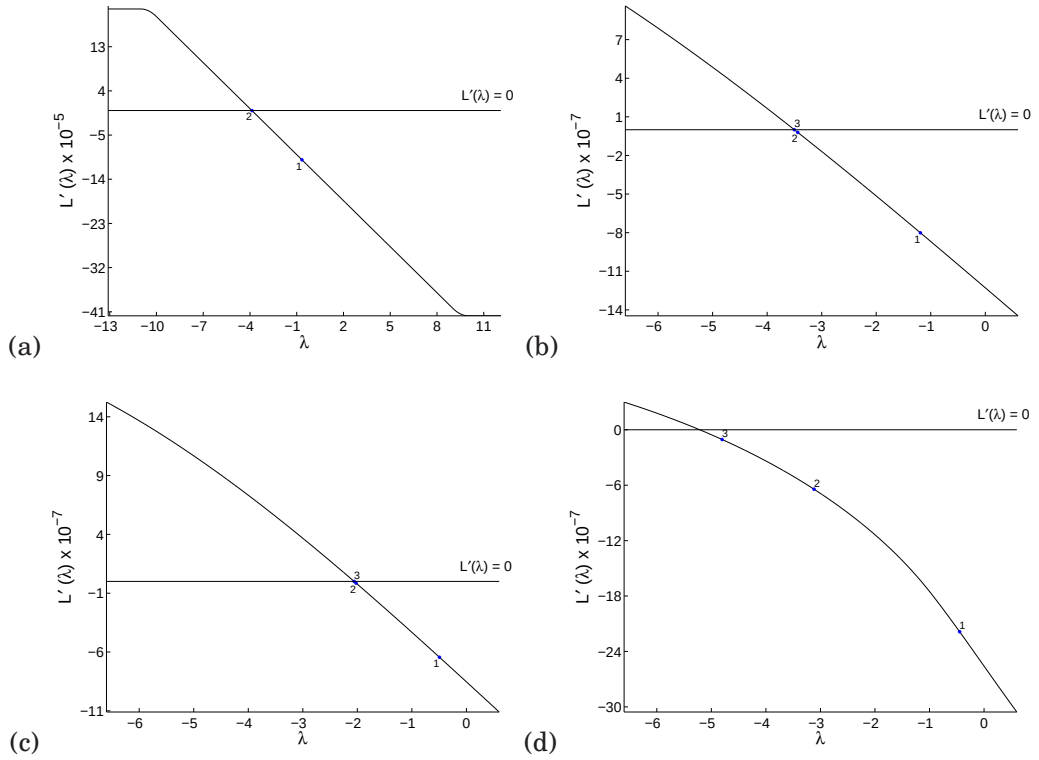

Fig. 3. Plots of L' for test set 4 through test set 1 (in descending order) are shown in panels (a)–(d) respectively. Also shown is the initial variable fixing iterate and 1 or 2 Newton iterates.

and quadratic resource allocation [Bitran and Hax 1981; Bretthauer and Shetty 1997; Cosares and Hochbaum 1994; Hochbaum and Hong 1995], respectively. We consider the case $r = s = b$ in (1). In problem sets 1–5, the scalar b in the constraint $\mathbf{a}^{\mathsf{T}}\mathbf{x} = b$ was chosen randomly in the interval $[A, B]$ where

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

In problem sets 6 and 7, b was chosen randomly in $[1, 100]$.

One deficiency with randomly generated problems is that the resulting dual function has the property that L' is essentially the same for each randomly generated problem instance when n is large. Moreover, for large randomly generated problems, L' tends to be relatively linear. In Figure 3 we plot L' for problems in the test sets 1–4 and $n = 6{,}250{,}000$. One factor related to the degree of nonlinearity for L' is the size and variability of the elements in $\mathbf{d}$. As d_i approaches 0, the two break points (9) associated with x_i coalesce into a single point y_i/a_i, and $x_i(\lambda)$ becomes piecewise constant with values equal to either ℓ_i or u_i. This causes the graph of L' to develop flat regions as seen in Figure 1.

For our test sets, the most linear plot, Figure 3a, corresponds to test set 4 where $d_i = 1$ for each i. For test set 3 (Figure 3b), d_i lies between 1 and 26, and there is a very slight bend in the L' plot. In test set 2 (Figure 3c), d_i lies between 0 and 37.5, while in test set 1 (Figure 3d) d_i is between 0 and 25. When d_i is chosen randomly in the interval $(0, 25]$ or $(0, 37.5]$, the smaller values of d_i can contribute to nonlinearity. Test set 5 is another version of test set 4 with $d_i = 1$ and a nearly linear L'. Test set 6 is similar to test set 1 with respect to the range of d_i, however, $x_i^* = 0$ for most i due to the

Table IV. CPU times (seconds) for NAPHEAP and the Newton method of [Cominetti et al. 2014].

	NAPHEAP			Newton		
Test Set	ave	min	max	ave	min	max
1	0.305	0.272	0.406	0.499	0.453	0.620
2	0.282	0.252	0.313	0.408	0.351	0.480
3	0.276	0.272	0.282	0.386	0.360	0.424
4	0.210	0.183	0.329	0.226	0.200	0.239
5	0.205	0.186	0.243	0.248	0.226	0.358
6	0.231	0.223	0.235	0.318	0.315	0.321
7	0.261	0.215	0.329	4.378	1.757	7.384

Table V. Number of Newton and secant iterations for NAPHEAP and the Newton method of [Cominetti et al. 2014].

	NAPHEAP						Newton					
	Newton Iterations			Secant Iterations			Newton Iterations			Secant Iterations		
Test Set	ave	min	max	ave	min	max	ave	min	max	ave	min	max
1	3.8	3	8	0.0	0	0	6.4	4	11	0.0	0	0
2	3.4	3	5	0.0	0	0	5.3	3	8	0.0	0	0
3	3.0	3	3	0.0	0	0	4.7	4	6	0.0	0	0
4	2.8	1	3	0.1	0	1	4.0	4	4	0.0	0	0
5	3.5	3	8	0.0	0	0	4.7	4	10	0.0	0	0
6	10.6	10	12	0.0	0	0	13.0	12	14	0.0	0	0
7	0.1	0	1	3.2	3	4	92.7	49	132	0.5	0	1

choice of the bound constraints. Since most of the components of the solution become fixed at 0 and drop out of the problem, the problems in test set 6 are easier than those in set 1. Test set 7 is difficult in the sense that the d_i nearly vanish, the break points for x_i nearly coincide, and L' is nearly piecewise constant. Consequently, $L''(\lambda)$ is zero for most choices of λ.

The near linearity of L' in test set 4 can be explained as follows: Assuming λ is not a break point, the second derivative of L is given by

$$L''(\lambda) = -\sum_{i \in \mathcal{F}(\lambda)} a_i^2/d_i, \quad \text{where } \mathcal{F}(\lambda) = \{i : \ell_i < x_i(\lambda) < u_i\}.$$

Since $a_i = d_i = u_i = 1$ and $\ell_i = 0$ in test set 4, $L''(\lambda) = -|\mathcal{F}(\lambda)|$ and

$$\mathcal{F}(\lambda) = \{i : 0 < y_i - \lambda < 1\} = \{i : y_i \in (\lambda, \lambda + 1)\}.$$

Since the y_i are uniformly distributed on $[-10, 10]$, it follows that for $\lambda \in [-10, 9]$ and for large n, it is highly likely that $|\mathcal{F}(\lambda)|$ is near $n/20$, a constant independent of λ. Consequently, L'' is nearly constant and L' is nearly linear on $[-10, 9]$.

In Table IV we compare the running time of NAPHEAP to that of the Newton code developed in [Cominetti et al. 2014]. As pointed out in the introduction, the Newton algorithm of [Cominetti et al. 2014] was faster than either the variable fixing method, the secant method, or a median based method. The experiments were run on a Dell Precision T7610 Workstation with a Dual Intel Xeon Processor E5-2687W v2 (32 cores, 3.4 GHz, 25.6 MB cache, 192 GB memory). Only one core was used in the experiments. There were 10 trials of each problem set, each of size n = 6,250,000. The graph of L' is essentially the same for each trial in a test set and the main difference between the problems in a test set is the vertical placement of the horizontal line $L'(\lambda) = 0$ of Figure 3; the placement of this horizontal line corresponds to the choice of b. We report the average, minimum, and maximum running time for the 10 trials.

As seen in Table IV, NAPHEAP and the Newton code perform nearly the same with respect to CPU time for test set 4, where $d_i = 1$ and L' is nearly linear (Figure 3a). When the nonlinearity increases, as in Figures 3b through 3d, or when the proportion

of free components of $\mathbf{x}^*$ increases, the advantage of NAPHEAP over Newton increases. For test sets 1 and 2, the number of components i such that $\ell_i < x_i^* < u_i$ is on the order of $n/3$ to $n/2$, which implies that the work associated with each Newton iteration near the optimal solution is proportional to n. The difference in the run times for test set 7 is connected with how the codes handle the special case $L''(\lambda_k) = 0$. NAPHEAP uses a secant iteration while the Newton code either moves to a nearby break point where L'' does not vanish or it uses a secant step.

Table V gives the average, maximum, and minimum number of Newton and secant steps taken by NAPHEAP and the Newton method of [Cominetti et al. 2014] for each test set. Neither code performed many secant iterations; in test set 7 where L' is nearly piecewise constant, NAPHEAP performed on average about 3 secant iterations and 1 variable fixing iterating before bracketing the solution and switching to the heap-based break point search. The variable fixing iteration was necessary when the interval bracketing the solution was unbounded. The large number of Newton steps taken by the Newton method in test set 7 is due to many flat regions on the curve L', which are not treated efficiently.

7. CONCLUSIONS

We have presented a new hybrid algorithm NAPHEAP for the continuous quadratic knapsack problem (1) with $\mathbf{d} \geq \mathbf{0}$. The algorithm is based on maximizing the dual function L in (6), and computing the optimal multiplier associated with the linear constraint. A Newton-type method is used to generate an interval that brackets an optimal dual multiplier. After arranging the break points inside the bracketing interval in heaps, one heap for the bound variables and one heap for the free variables at the current iterate, NAPHEAP starts from one side of the bracketing interval and moves monotonically across the break points until reaching an optimal dual multiplier. In order to handle the case where $d_i = 0$ for one or more indices, we developed a new version of the variable fixing algorithm and proved its convergence. For the randomly generated test problems of Section 6, the hybrid algorithm NAPHEAP was faster than pure Newton.

ACKNOWLEDGMENTS

Constructive comments by the reviewers, which significantly improved both the paper and the code, are gratefully acknowledged.

REFERENCES

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

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

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

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

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

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

R. Cominetti, W. F. Mascarenhas, and P. J. S. Silva. 2014. A Newton's method for the continuous quadratic knapsack problem. *Math. Prog. Comp.* 6 (2014), 151–169.

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

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

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

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

Y. S. Fu and Y. H. Dai. 2010. Improved projected gradient algorithms for singly linearly constrained quadrataic programs subject to lower and upper bounds. *Asia-Pac. J. Oper. Res.* 27 (2010), 71–84.

W. W. Hager and J. T. Hungerford. 2014. Optimality Conditions For Maximizing a Function Over a Polyhedron. *Math. Program.* 145 (2014), 179–198.

W. W. Hager and J. T. Hungerford. 2015. Continuous quadratic programming formulations of optimization problems on graphs. *European J. Oper. Res.* 240 (2015), 328–337.

W. W. Hager, J. T. Hungerford, and I. Safro. 2014. A Multilevel Bilinear Programming Algorithm for the Vertex Separator Problem. *SIAM J. Sci. Comput.* submitted, http://clas.ufl.edu/users/hager/papers/GP/ml.pdf (2014).

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

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

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

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

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

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

N. Maculan, C.P. Santiago, E.M. Macambira, and M.H.C. Jardim. 2003. 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), 553–574.

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

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

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

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

R. T. Rockafellar. 1970. *Convex analysis*. Princeton Univ. Press.

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

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

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

Received September 2015; revised Month Year; accepted Month Year