A Newton's method for the continuous quadratic knapsack problem *

Roberto Cominetti† Walter F. Mascarenhas‡

Paulo J. S. Silva§

October 13th, 2013

Abstract

We introduce a new efficient method to solve the continuous quadratic knapsack problem. This is a highly structured quadratic program that appears in different contexts. The method converges after $O(n)$ iterations with overall arithmetic complexity $O(n^2)$. Numerical experiments show that in practice the method converges in a small number of iterations with overall linear complexity, and is faster than the state-of-the-art algorithms based on median finding, variable fixing, and secant techniques.

Keywords: continuous quadratic knapsack, simplex projections, semismooth Newton, duality

Mathematics Subject Classification MSC2010: 65K05, 90C20, 90C56

1 Introduction

In this paper we develop an efficient method to solve the following *continuous quadratic knapsack problem*

$$\begin{aligned} \min_x \ & \tfrac{1}{2}\, x'Dx - a'x \\ \text{s.t. } & b'x = r \\ & l \le x \le u \end{aligned} \tag{1}$$

*This work was supported by Fondecyt 1100046 and Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F, PRONEX-Optimization (PRONEX-CNPq/FAPERJ E-26/171.510/2006-APQ1), CNPq (Grant 305740/2010-5), and FAPESP (Grant 2012/20339-0).

†Departamento de Ingeniería Industrial, Universidad de Chile. e-mail: rccc@dii.uchile.cl.

‡Computer Science Dept., University of São Paulo, Brazil. e-mail: walter.mascarenhas@gmail.com.

§Applied Mathematics Dept., University of Campinas, Brazil. e-mail: pjssilva@ime.unicamp.br.

where $x \in \mathbb{R}^n$ is the decision variable; $D = \mathrm{diag}(d)$ is a diagonal matrix with positive entries $d_i > 0$; a, b are vectors in $\mathbb{R}^n$; r is a scalar; and $l < u$ are lower and upper bounds, some of which may be infinite. We assume that all the b_i's are nonzero since for a null coefficient one may use separability to fix the corresponding variable x_i at its optimal value and remove it from the problem.

As D is strictly positive it induces the norm $\|x\|_D = \sqrt{x'Dx}$, so that (1) amounts to projecting the vector $D^{-1}a$ onto the intersection of the hyperplane $b'x = r$ and the box defined by the lower and upper bounds. Thus there is a unique optimal solution as long as the feasible set is nonempty. This simple problem appears in many settings including resource allocation problems [3, 4, 16], multicommodity network flows [15, 24, 28], Lagrangian relaxation using subgradients methods [14], and quasi-Newton updates with bounds [8]. A particularly relevant application that requires the repeated solution of high dimensional problems of type (1) is the training of *support vector machines* using the spectral projected gradient method [10].

The continuous quadratic knapsack is a well studied problem. Using simple duality arguments it can be restated as the solution of a one-dimensional piecewise linear equation that is amenable to different solutions strategies. From the complexity viewpoint the most efficient methods are based on median search algorithms which attain linear time complexity [6, 8, 9, 16, 19, 21, 25]. Another class of algorithms is based on variable fixing techniques [3, 23, 26, 29]. These methods have quadratic complexity though in practice they turn out to be faster than median search methods. Recently, Dai and Fletcher [10] proposed a specially tailored secant method to solve the equation reformulation of the problem, also with good practical performance.

In this paper we propose a semismooth Newton method to solve (1). The approach is similar to Dai and Fletcher's where we replace the secant step by a Newton's step, avoiding the initial bracketing phase required by the secant method. The method also exploits variable fixing ideas and builds upon the work of Bitran and Hax [3] and Kiwiel [20] among others. Actually, we show that the variable fixing methods are closely related to the Newton's iteration and even mimic it in some special cases. Our numerical experiments suggest that the semismooth Newton method is faster than all previous methods, specially when using hot-starts to solve a sequence of similar problems.

The paper is organized as follows. In Section 2 we present the dual problem and its reformulation as a piecewise linear equation in a single variable. In Section 3 we introduce the semismooth Newton method for the case with single bounds and prove its finite convergence without any globalization strategy. In Section 4 we extend the method to the case of lower and upper bounds. Section 5 relates the proposed algorithm to previous methods in the literature, unveiling some interesting connections. Finally, Section 6 reports numerical experiments comparing the Newton's method with the fastest methods currently available.

2 A dual reformulation

As any convex quadratic problem, the continuous quadratic knapsack problem conforms to strong duality. Hence it is natural to consider the solution of its dual as a mean to recover the primal solutions.

Taking into account the separability of the objective function and of the box constraints, we can see that the complicating constraint is the linear one. Hence it is natural to dualize only this constraint by considering the single variable dual problem

$$\max_{\lambda} \inf_{l \le x \le u} \left\{ \tfrac{1}{2}\, x'Dx - a'x + \lambda(r - b'x) \right\}. \tag{2}$$

The inner infimum that defines the dual objective is easily solved with optimal solution

$$x(\lambda) = \operatorname{mid}(l, D^{-1}(b\lambda + a), u) \tag{3}$$

where $\operatorname{mid}(l, x, u)$ stands for the component-wise median of the three vectors. Moreover, the KKT conditions for (1) can be written as

$$\varphi(\lambda) := b'x(\lambda) = r \tag{4}$$

which is equivalent to primal-dual optimality, so that problem (1) reduces to a one dimensional equation with

$$\varphi(\lambda) = \sum_{i=1}^{n} b_i \operatorname{mid}(l_i, (b_i\lambda + a_i)/d_i, u_i). \tag{5}$$

An example of the function $\varphi(\lambda)$ is shown in Figure 1. Since each term of the sum is piecewise linear and non-decreasing, the same properties hold for φ. The points where it changes derivative, $(d_i l_i - a_i)/b_i$ and $(d_i u_i - a_i)/b_i$, are called *breakpoints*.

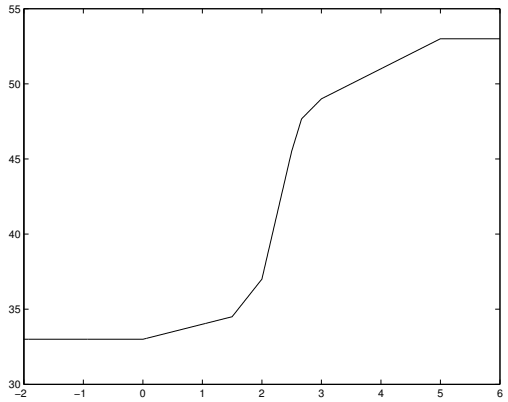

Figure 1: A function φ with 6 breakpoints

Note that problem (1) is infeasible if and only if equation (4) does not have a solution. This can only happen if $\varphi(\cdot)$ is constant up to the first breakpoint and starts already above r or if it is constant after the last breakpoint but did not attain the value r.

3 Newton's method with single bounds

Let us consider first the special case of problem (1) in which each variable has either an upper or a lower bound but not both. By a simple change of variables we reduce to the case with lower bounds only (*i.e.* $u_i = \infty$ for all i). In this case the minimizer $x(\lambda)$ in the dual objective is just

$$x(\lambda) = \max(l, D^{-1}(b\lambda + a)) \tag{6}$$

and the components of the summation that define φ are

$$\begin{cases} \max(b_i l_i, (b_i^2\lambda + b_i a_i)/d_i) & \text{if } b_i > 0 \\ \min(b_i l_i, (b_i^2\lambda + b_i a_i)/d_i) & \text{if } b_i < 0 \end{cases} \tag{7}$$

which consist of two basic "shapes". The first starts as a constant function and then after the breakpoint it turns into a linear function with positive slope. The other starts as an increasing linear functions and flats out after the breakpoint. We call the first type of break points *positive*, because they increase the derivative of φ, and the second type will be called *negative*. We denote these breakpoints as $z_i = (d_i l_i - a_i)/b_i,\ i = 1, \ldots, n$.

The derivative $\varphi'(\lambda)$ is simple to calculate. All one needs is to sum up the slopes b_i^2/d_i corresponding to positive breakpoints to the left of λ plus the slopes of negative breakpoints to the right. In the case that λ is itself a breakpoint, there is change of slope but the right and left derivatives are still well defined and can be computed by the formulas

$$\varphi'_+(\lambda) = \sum_{\substack{b_i>0 \\ z_i\le\lambda}} b_i^2/d_i + \sum_{\substack{b_i<0 \\ z_i>\lambda}} b_i^2/d_i, \tag{8}$$

$$\varphi'_-(\lambda) = \sum_{\substack{b_i>0 \\ z_i<\lambda}} b_i^2/d_i + \sum_{\substack{b_i<0 \\ z_i\ge\lambda}} b_i^2/d_i. \tag{9}$$

With these notations, the proposed Newton's method is given by Algorithm 1. We claim that the iteration converges globally without any kind of globalization strategy. To this end we note that since there are at most n breakpoints the function $\varphi(\cdot)$ has at most at most $n+1$ linear pieces and therefore the Newton's iterates may generate at most $n+1$ distinct points (when the slope is zero we consider the closest breakpoint as the Newton's iterate). Thus, convergence can only fail if the method generates a cycle that repeats indefinitely. In order to show that these cycles can not occur we use the following simple fact.

Lemma 3.1 *Let $\alpha < \beta$ be two scalars. If $\gamma \in (\alpha, \beta)$ is a point where φ is differentiable then*

$$\varphi'(\gamma) \le \varphi'_+(\alpha) + \varphi'_-(\beta).$$

Algorithm 1 – Newton's method for single bounds

1. Given λ_k, compute $x(\lambda_k)$, $\varphi(\lambda_k)$, $\varphi'_-(\lambda^k)$, $\varphi'_+(\lambda^k)$.
2. If $\varphi(\lambda_k) = r$: report $x(\lambda_k)$ as the optimal solution and stop.
3. If $\varphi(\lambda_k) < r$:
   - If $\varphi'_+(\lambda_k) > 0$, set $\lambda_{k+1} := \lambda_k - \frac{\varphi(\lambda_k)-r}{\varphi'_+(\lambda_k)}$
   - If $\varphi'_+(\lambda_k) = 0$, set λ_{k+1} as the closest breakpoint to the right of λ_k. If no such breakpoint exists declare the problem infeasible and stop.
4. If $\varphi(\lambda_k) > r$:
   - If $\varphi'_-(\lambda_k) > 0$, set $\lambda_{k+1} := \lambda_k - \frac{\varphi(\lambda_k)-r}{\varphi'_-(\lambda_k)}$
   - If $\varphi'_-(\lambda_k) = 0$, set λ_{k+1} as the closest breakpoint to the left of λ_k. If no such breakpoint exists declare the problem infeasible and stop.
5. Set $k := k + 1$ and go to 1.

Proof. Let p and n denote respectively the sums of the slopes b_i^2/d_i for the positive and negative breakpoints in (α, β). Using (8) and (9) it follows that $\varphi'_-(\beta) = \varphi'_+(\alpha) + p - n$. Moreover, using (8) we see that $\varphi'_+(\alpha) \geq n$. Hence, for any $\gamma \in (\alpha, \beta)$ we have

$$\begin{aligned}\varphi'(\gamma) &\leq \varphi'_+(\alpha) + p \\ &\leq \varphi'_+(\alpha) + p + \varphi'_+(\alpha) - n \\ &= \varphi'_+(\alpha) + \varphi'_-(\beta).\end{aligned}$$

Theorem 3.2 *Algorithm 1 stops in at most $n + 1$ iterations with an overall complexity of $O(n^2)$ arithmetic operations.* □

Proof. We already observed that there are at most $n + 1$ Newton images so that all we need to show is that the method cannot cycle. For the sake of a contradiction, suppose that a cycle is reached. Since the solution set is a closed interval and Newton's method is monotone on each side of it, a cycle must have points both to the left and to the right of the solution set.

Let α be the largest point in the cycle to the left of the solution set and β the smallest one to the right. Let γ be a point of maximal derivative for φ in the open interval (α, β). Note that the derivative of φ can not be constant in this interval, otherwise the method would yield a solution in one step from either α or β and there would be no cycle. Moreover, both $\varphi'_+(\alpha)$ and $\varphi'_-(\beta)$ must be non-zero since otherwise the method would move to a breakpoint closer to the solution set but on the same side, contradicting the definition of α and β.

Now consider the Newton images from α and β, that belong to the cycle.

From the definition of α and β we get

$$\alpha - \frac{\varphi(\alpha) - r}{\varphi'_+(\alpha)} \geq \beta \iff \varphi'_+(\alpha)(\beta - \alpha) \leq -\varphi(\alpha) + r,$$
$$\beta - \frac{\varphi(\beta) - r}{\varphi'_-(\beta)} \leq \alpha \iff \varphi'_-(\beta)(\beta - \alpha) \leq \varphi(\beta) - r.$$

Adding these inequalities we get

$$\begin{aligned}(\varphi'_+(\alpha) + \varphi'_-(\beta))(\beta - \alpha) &\leq \varphi(\beta) - \varphi(\alpha) \\ &< \varphi'(\gamma)(\beta - \alpha) \\ &\leq (\varphi'_+(\alpha) + \varphi'_-(\beta))(\beta - \alpha)\end{aligned}$$

where the strict inequality in the middle follows from the fact that φ' is not constant over (α, β), while the last bound comes from Lemma 3.1.

This contradiction proves that no cycle can occur and the method terminates in no more than $n+1$ iterations. Since the cost of an iteration is dominated by the evaluation the function and the one-sided derivative all of which takes $O(n)$ arithmetic operations, the overall complexity is $O(n^2)$. □

The worst case bound in the previous result is actually sharp and can be attained in some specific instances. To this end we note that every piecewise linear increasing convex function can be written in the form (5) with $u_i = \infty$. In particular, if we consider the equation $\varphi(\lambda) = 0$ for the function

$$\varphi(\lambda) := \max_{i=1,\ldots,n} \tfrac{1}{3^{n-i}}(\lambda - i + 1)$$

which has positive breakpoints at $z_i = i + 1/2$ for $i = 1, \ldots, n-1$, the Newton's method started from any point $\lambda_0 > n - 1/2$ will go successively through the iterates $\lambda_1 = n - 1, \lambda_2 = n - 2, \ldots, \lambda_i = n - i$, attaining the solution $\lambda_n = 0$ after exactly n steps. Computing the function values at those points yields an overall complexity of $\Omega(n^2)$ arithmetic operations.

4 Newton's method with lower and upper bounds

Let us consider now the original problem with both lower and upper bounds. In order to simplify the analysis we assume that the vector b is strictly positive. Clearly, if $b_i < 0$ for some i, a simple change of variables can revert this sign. These changes can be done either explicitly in the model data or can be dealt implicitly in the algorithm implementation.

In this case, each term in the sum that defines $\varphi(\lambda)$ is constant up to its first breakpoint $z_i^- = (d_i l_i - a_i)/b_i$, where it turns into an increasing linear function with slope b_i^2/d_i up to the next breakpoint $z_i^+ = (d_i u_i - a_i)/b_i$ where it becomes constant again (see Figure 2). Naturally, if some bounds are infinite the corresponding breakpoints are not present.

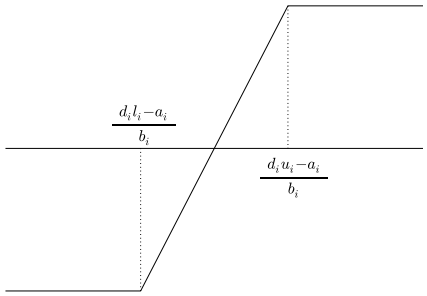

Figure 2: The components of φ

In any case φ is still a nondecreasing piecewise linear function, though the key Lemma 3.1 does not hold anymore and Newton's method may cycle as illustrated by the following simple example that cycles between -1.0 and 1.0 (see Figure 3)

$$\begin{aligned} &\min \tfrac{1}{2}\, x'x \\ &\text{s.t. } \sqrt{2}x_1 + x_2 + x_3 = 0 \\ &\quad -\tfrac{1}{\sqrt{2}} \le x_1 \le \tfrac{1}{\sqrt{2}}\ ; \quad 0 \le x_2\ ; \quad x_3 \le 0. \end{aligned}$$

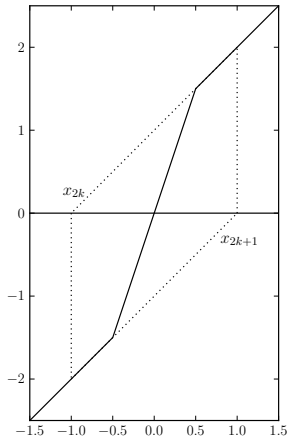

Figure 3: An example where Newton's method cycles.

As a matter of fact, the components in Figure 2 can be superposed to represent any nondecreasing piecewise linear function so that one can build examples where Newton's method has cycles of any prescribed length. Thus, we require

a globalization strategy to ensure convergence. Inspired by the proof of Theorem 3.2 we let α_k be the largest iterate with $\varphi(\alpha_k) < r$ computed by the algorithm up to iteration k, and β_k the smallest iterate with $\varphi(\beta_k) > r$. A cycle can only occur if the next Newton's iterate falls outside the current interval (α_k, β_k). If this happens we proceed by taking a secant step which leads to a variation of the original Newton's method in Algorithm 2 below. This variation revisits the secant method of Dai and Fletcher [10], replacing the initial bracketing phase by Newton's iterations and using Newton whenever possible.

The secant globalization is complemented with the following variable fixing technique that helps to reduce the problem size as the method progresses. Namely, consider a given λ and suppose that $\varphi(\lambda) < r$. If $x_i(\lambda) = u_i$ for some coordinate i then this coordinate is also u_i at the optimal solution. The reason for this is that $\varphi(\cdot)$ is nondecreasing so that the solution of the equation must be larger than λ, while (6) implies that if $x_i(\lambda)$ is already u_i it will remain constant for larger values of the multiplier. Hence, we can fix the value of the i-th coordinate, update the right hand side of the constraint, and solve a smaller problem. The same reasoning applies when $\varphi(\lambda) > r$ in which case the variables that are mapped to the lower bound are the ones that can be fixed. This *variable fixing* strategy is actually very effective and decreases the problem size rapidly. This technique is the basis of some of the most efficient algorithms (*cf.* [20]) based on the pioneering work of Bitran and Hax [3]. We incorporate variable fixing in the final version of the method as described in Step 6 of Algorithm 2. The convergence of the method is established in the following theorem.

Theorem 4.1 *Algorithm 2 stops in at most* $4n + 1$ *iterations with an overall complexity of* $O(n^2)$ *arithmetic operations.*

Proof. We note that in each iteration the new multiplier λ_{k+1} is different from al the previous ones, which is ensured by the fact that $\lambda_{k+1} \in (\alpha_k, \beta_k)$.

Since there are at most $2n$ breakpoints, the function $\varphi(\lambda)$ has has at most $2n + 1$ linear pieces which bounds the number of Newton's iterates. We also count as Newton's steps the closest breakpoints obtained in the case of zero slopes. On the other hand, each modified secant iteration removes at least one breakpoint from the open bracket interval (α_k, β_k) so that there can be at most $2n$ of such steps. Altogether we have at most $4n+1$ iterations, each one involving $O(n)$ arithmetic operations which yields the conclusion. □

Considering the example given in the end of Section 3, we see that the upper complexity bound is again tight. However, the numerical experiments presented in Section 6 show a much better practical performance of $O(n)$ arithmetic operations.

Remark. While the proof of Theorem 4.1 shows that there can be at most $2n$ secant steps, one can also argue heuristically that such steps should be rare. Namely, among the lower and upper bounds of each variable only one of them can be active at the optimal solution, so that eventually the method will behave as in the case of single bounds where no globalization strategy is needed.

Algorithm 2 – Newton-secant method for continuous quadratic knapsack

1. Let $k = 0, \lambda_0 \in \mathbb{R}$, $\alpha_{-1} = -\infty$, $\beta_{-1} = +\infty$.
2. If $\varphi(\lambda_k) = r$: report $x(\lambda_k)$ as the optimal solution and stop.
3. If $\varphi(\lambda_k) < r$:
   - Define $\alpha_k := \lambda_k, \beta_k := \beta_{k-1}$,
   - If $\varphi'_+(\lambda_k) > 0$, set $\lambda_N := \lambda_k - \frac{\varphi(\lambda_k)-r}{\varphi'_+(\lambda_k)}$. If $\lambda_N < \beta_k$, define $\lambda_{k+1} := \lambda_N$ and go to Step 6. Otherwise go to Step 5.
   - If $\varphi'_+(\lambda_k) = 0$, set λ_{k+1} as the closest breakpoint to the right of α_k and go to Step 6. If no such breakpoint exists declare the problem infeasible and stop.
4. If $\varphi(\lambda_k) > r$:
   - Define $\alpha_k := \alpha_{k-1}, \beta_k := \lambda_k$,
   - If $\varphi'_-(\lambda_k) > 0$, set $\lambda_N := \lambda_k - \frac{\varphi(\lambda_k)-r}{\varphi'_-(\lambda_k)}$. If $\lambda_N > \alpha_k$, define $\lambda_{k+1} := \lambda_N$ and go to Step 6. Otherwise go to Step 5.
   - If $\varphi'_-(\lambda_k) = 0$, set λ_{k+1} as the closest breakpoint to the left of β_k and go to Step 6. If no such breakpoint exists declare the problem infeasible and stop.
5. Let λ_S be the secant step in $[\alpha_k, \beta_k]$, and α, β the smallest and largest breakpoints in (α_k, β_k). If $\varphi(\lambda_S) < r$ set $\lambda_{k+1} := \max(\lambda_S, \alpha)$ otherwise set $\lambda_{k+1} := \min(\lambda_S, \beta)$.
6. Use variable fixing to fix as many variables as possible.
7. Set $k := k + 1$ and go to Step 2.

5 Related work

The semismooth Newton method described above has several points in common with previous algorithms. In this section we discuss some of these connections, while in Section 6 we present some numerical experiments comparing their practical performance.

A first comparison can be made with the semismooth Newton method for support vector machines developed in [12, Ferris and Munson]. In that paper the Fisher-Burmeister function is used to reformulate the complementarity conditions for the KKT system as a system of $n+1$ semismooth nonlinear equations. The paper solves this system by a semismooth Newton method that requires in each iteration the choice of a generalized Jacobian, though the authors do not specify a particular one. In this framework, our Newton's method can be

interpreted as the choice of a particular generalized Jacobian which arises naturally from the underlying one-dimensional equation. This specific choice allows to establish the finite convergence of the method, improving the results in [12] that prove only convergence of accumulation points with Q-quadratic convergence under appropriate conditions. Note however that [12] is designed to solve the more general non-diagonal quadratic programs that arise in support vector machines, so that the comparison is not completely fair.

From the complexity point of view our $O(n^2)$ estimate is not competitive with the linear complexity $O(n)$ attained by median search algorithms. These methods use a bracketing interval $[\alpha_k, \beta_k]$ that contains the solutions of the equation, and perform a binary search by choosing the median of all the breakpoints in this interval as a trial point λ to determine the next interval $[\alpha_{k+1}, \beta_{k+1}]$. The number of breakpoints in the new interval is cut by half, and the procedure continues until only 2 breakpoints remain which identifies a linear piece of $\varphi(\lambda)$ where the solution can be computed by solving a single linear equation. While each iteration cuts the work by half, it keeps half of the breakpoints so that the upper bound complexity coincides with the lower bound. In our Newton's method the worst case requires $O(n)$ iterations but in practice we frequently observe a fixed number of iterations (typically around 5) so that the practical performance grows linearly with the dimension.

The semismooth Newton method can also be compared to the secant method of Dai and Fletcher [10]. This method requires an initial phase to find a bracketing interval $[\alpha_0, \beta_0]$ and then proceeds with secant steps to narrow down the interval until it reaches a linear piece of $\varphi(\lambda)$ where the next secant step finds an exact solution. The Newton's iteration avoids the initial bracketing phase and progresses towards the solution from the first iteration. It only uses a secant step as a globalization strategy when Newton fails to get closer to the solution, in which case a bracketing interval is already available.

Finally, Newton's method has an interesting connection with the variable fixing method of Kiwiel [20]. This method proceeds by momentarily ignoring the lower and upper bounds and finds the optimal dual solution λ after which it computes $\varphi(\lambda)$ and proceeds to fix as many variables as possible based on the technique described in the previous section. The fixed variables are removed and the procedure is repeated for the lower dimensional problem. Since each step fixes at least one variable (usually many more), the method is guaranteed to converge in at most n iterations. Although the underlying philosophy is very different from the Newton's iteration, it turns out that when $b > 0$ and only lower bounds or upper bounds are present, both methods coincide. More precisely

Proposition 5.1 *Suppose $b > 0$ and either $u_i = \infty$ for all i or $l_i = -\infty$ for all i. Let $x_i = (b_i\lambda_0 + a_i)/d_i$ be the initial point in the variable fixing method where the multiplier λ_0 is given by $\lambda_0 = (r - s_0)/q_0$ with*

$$s_0 = \textstyle\sum_{i=1}^n \frac{a_i b_i}{d_i} \quad ; \quad q_0 = \sum_{i=1}^n \frac{b_i^2}{d_i}.$$

Then the Newton's iteration started from λ_0 generates exactly the same iterates

as the variable fixing method.

Proof. We consider only the case of lower bounds, since the other case is similar. In this case the function $\varphi(\cdot)$ is convex so that the Newton's iterates will approach the solution set monotonically with $\varphi(\lambda_k) \geq r$ for all $k \geq 1$.

Let x^k, λ_k be the sequences generated by the variable fixing method (notice the change of sign with respect to [20] which uses the variable $t_k = -\lambda_k$ instead). Let I_k denote the indices of the variables that have not yet been fixed up to the k-th iteration, so that $x_i^k = l_i$ for $i \notin I_k$ and $x_i^k = (b_i\lambda_k + a_i)/d_i$ for $i \in I_k$. The method proceeds by fixing all the variables $i \in I_k$ such that $x_i^k \leq l_i$ and updates $I_{k+1} = \{i \in I_k : x_i^k > l_i\}$.

According to [20, Lemma 4.1(c)], we have

$$\varphi(\lambda_k) - r = \nabla_k - \Delta_k$$

with $\nabla_k = \sum_{i \in I_k} b_i(l_i - x_i^k)_+$ and $\Delta_k = \sum_{i \in I_k} b_i(x_i^k - u_i)_+$. Since $u_i = \infty$ we have in fact $\Delta_k = 0$ and therefore $\varphi(\lambda_k) - r = \nabla_k \geq 0$. Furthermore, using formula (25) in [20] it follows that the multipliers λ_k satisfy the recursion

$$\begin{aligned} \lambda_{k+1} &= \lambda_k - \tfrac{1}{q_{k+1}}\nabla_k \\ q_{k+1} &= q_k - \textstyle\sum_{i \in I_k \setminus I_{k+1}} \frac{b_i^2}{d_i}. \end{aligned}$$

The update for q_k iteratively removes from the original sum q_0 all the slopes $\frac{b_i^2}{d_i}$ of the variables fixed along the iterations. It is easy to see that this gives precisely $q_{k+1} = \varphi'_-(\lambda_k)$ so that the update of λ_k coincides with Newton's iteration

$$\lambda_{k+1} = \lambda_k - \tfrac{\varphi(\lambda_k) - r}{\varphi'_+(\lambda_k)}.$$

□

REMARK. Proposition 5.1 does not hold when both upper and lower bounds are present, or when the coordinates of b have different signs. In such cases Newton's method and the variable fixing method generate different iterates.

REMARK. The variable fixing method always starts from the specific point λ_0, and coincides with Newton only if we choose the same starting point. In this sense Newton's method is more flexible as it can be started from any initial guess. This can be useful if we have a good solution estimate from a similar problem solved previously.

6 Numerical Experiments

In this section we present numerical experiments comparing the proposed Newton's method with our implementations of state-of-the-art solvers based on the secant algorithm [10], median search [19], and variable fixing [20]. All the code

used in these experiments can be downloaded at `http://www.ime.unicamp.br/~pjssilva/research/quadratic_knapsack_source.zip`.

The programs were written in C (ANSI C99) and run on a Desktop with an Intel Core i7 3930K CPU (3.20GHz). To ensure that the processor will always work on its rated clock speed, we turned force the Linux system to use the performance CPU governor. The computer has 64Gb of memory and runs Ubuntu 13.04 64bit. The compiler used was gcc 4.7.3 with optimization flags `- march=native -O3 -fast-math`. The implementation of the algorithm from [19] depends heavily on the median finding routine. We used Floyd and Rivest's SELECT algorithm as implemented by Kiwiel [17]. This code is in Fortran and was compiled using gfortran version 4.7.3 with the same flags.

A simple vector with the indexes of the free variables and indirect indexing was used to implement the variable fixing step in the variable fixing and Newton's methods. This simple strategy ensures that iterations of these methods can still be performed in linear time. In principle, the secant algorithm could benefit from variable fixing as well, but after preliminary experiments we observed that this was not beneficial. Variable fixing increases the work per iteration in the secant method since the function φ needs to be recomputed at the end point of the bracketing interval that was computed in previous iterations. Therefore, we report the results for the secant method in its pure form described by Dai and Fletcher in [10].

The main stopping criterion used for the Newton's and the secant method in our implementation has the form

$$|\varphi(\lambda^k) - r| \leq \epsilon \left(\sum_{i=1}^{n} |b_i x(\lambda^k)_i| + |r| \right),$$

for small $\epsilon > 0$. Since the median search algorithm solves the continuous quadratic knapsack problem basically exactly, we used a stringent value for ϵ, defining it as 10^{-12}. We also adopted a similarly high precision in the stopping criterion suggested for the variable fixing algorithm in [20]. In practice we observed that all methods solved the problems up to machine precision. Finally, both the secant and Newton's methods are initialized with the first multiplier estimate computed by the variable fixing method. This is the multiplier associated to the linear constraint in (1) when it is solved ignoring the bounds.

We performed two types of experiments. The first group were randomly generated instances based on a standard test set for continuous quadratic knapsack described in [5, 18, 19, 20], with the addition of a problem from [10]. The second is the training of support vector machines, also described in [10].

6.1 Randomly generated tests

As in [5, 18, 19, 20], we generated three groups of problems with dimensions ranging from $n = 50000$ up to $n = 2000000$, with data entries chosen by independent draws from uniform distributions $U[a, b]$. The classes are

1. Uncorrelated: $a_i, b_i, c_i \sim U[10, 25]$;

2. Weakly correlated: $b_i \sim U[10, 25]$ and $a_i, d_i \sim U[b_i - 5, b_i + 5]$;

3. Correlated: $b_i \sim U[10, 25]$ and $a_i = d_i = b_i + 5$.

In all cases l_i and u_i were chosen uniformly in $[1, 15]$, respecting the order, while r was chosen uniformly in $[b^t l, b^t u]$. The random numbers were computed using a fast variation of the Mersenne Twister algorithm [22, 27].

The results are presented in Tables 1, 2, and 3. Each line reports the average, maximum, and minimum number of iterations and running times over 100 randomly generated tests for each dimension. In order to obtain a trustable estimate for the running time, each random test was repeated $10^7/n$ times in a loop, totalizing roughly 1 second for the slowest method (median) and 0.5 second for the fastest method (Newton). We report the mean time for each random test. For the Newton's, Secant, and Variable fixing methods we count an iteration whenever the function φ is evaluated (directly or indirectly). In particular, the bracketing phase of the secant method counts as at least two iterations, unless the initial multiplier is the correct solution. The initialization step of the Newton's and the variable fixing methods, that involves computing φ at the starting multiplier, is counted as one iteration. For the median search method one iteration is counted whenever a median search is needed.

The results indicate that the Newton's method is very effective. It requires fewer iterations and, what is more important, its running time is the lowest one. In the largest tests, Newton's method was about 30% faster than the others. The median search method is the slowest, although it has the lowest theoretical complexity. This happens because the other methods do not achieve their worst case performance. The number of iterations in all these methods is virtually constant with n, as seen from the average number iterations and their small deviations. Note, however, that Newton's method apparent advantage due to its smaller number of iterations is not fully reflected in its running time because it needs to calculate both φ and its derivative. Moreover the number of iterations in the median search algorithm is misleading as the final iterations handle a very small number of breakpoints.

We also performed random experiments that are similar to problems arising in multicommodity network flows and logistics (see [10], and [25]). In this case the instances were generated with $d_1 = 1$, $d_n = 10^4$, $d_i \sim U[d_1, d_n]$ for $i = 2, \ldots, n-1$, and $a_i \sim U[-1000, 1000]$, $b_i = 1$, $l_i = 0$, $u_i \sim U[0, 1000]$ all for $i = 1, \ldots, n$, while r was selected uniformly in $[b^t l, b^t u]$. The results for these tests are presented in Table 4. These results may seem surprising, with the variable fixing method performing almost identically as Newton. This happened because in most instances the upper bounds were not binding at the solution and both Newton's and the variable fixing algorithms were generating the same sequence of iterates as suggested by Proposition 5.1 (see also the remark after Theorem 4.1). The slightly smaller running time for the variable fixing method is due to its simpler update of the directional derivatives (see the proof of Proposition 5.1). On the other hand, it is interesting to see that Newton has a much better performance than the secant method in this case.

Dimension	Iterations			Time (msec)			Iterations			Time (msec)		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
	Newton						**Secant**					
50000	4.7	8	3	2.2	2.7	1.8	8.2	12	6	3.3	4.5	2.5
100000	5.2	9	3	4.6	5.4	3.8	8.7	14	6	6.7	9.3	4.8
500000	5.3	9	4	26.0	31.0	21.0	8.6	14	6	34.6	52.0	25.5
1000000	5.2	10	3	51.7	64.0	42.0	8.4	13	6	67.6	97.0	50.0
1500000	5.1	9	3	76.4	93.3	56.7	8.4	14	6	103.4	155.0	76.7
2000000	5.2	11	4	102.2	126.0	84.0	8.5	15	7	139.7	194.0	116.0
	Variable fixing						**Median search**					
50000	7.8	11	6	2.6	3.0	2.2	16.7	17	16	4.2	4.5	3.6
100000	8.2	11	7	5.2	5.8	4.4	17.7	18	17	8.1	8.6	7.3
500000	8.7	12	7	32.5	36.0	28.0	19.9	20	19	46.4	49.0	41.0
1000000	8.8	12	7	66.6	75.0	56.0	20.9	21	20	94.3	99.0	83.0
1500000	8.9	12	7	100.1	113.3	85.0	21.6	22	21	157.1	165.0	141.7
2000000	8.9	12	7	132.8	152.0	112.0	22.0	22	21	209.0	220.0	186.0

Table 1: Uncorrelated test

Dimension	Iterations			Time (msec)			Iterations			Time (msec)		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
	Newton						**Secant**					
50000	4.7	8	3	2.2	2.7	1.8	8.2	12	6	3.3	4.5	2.5
100000	5.2	9	3	4.6	5.4	3.8	8.7	14	6	6.7	9.3	4.8
500000	5.3	9	4	26.0	31.0	21.0	8.6	14	6	34.6	52.0	25.5
1000000	5.2	10	3	51.7	64.0	42.0	8.4	13	6	67.6	97.0	50.0
1500000	5.1	9	3	76.4	93.3	56.7	8.4	14	6	103.4	155.0	76.7
2000000	5.2	11	4	102.2	126.0	84.0	8.5	15	7	139.7	194.0	116.0
	Variable fixing						**Median search**					
50000	7.8	11	6	2.6	3.0	2.2	16.7	17	16	4.2	4.5	3.6
100000	8.2	11	7	5.2	5.8	4.4	17.7	18	17	8.1	8.6	7.3
500000	8.7	12	7	32.5	36.0	28.0	19.9	20	19	46.4	49.0	41.0
1000000	8.8	12	7	66.6	75.0	56.0	20.9	21	20	94.3	99.0	83.0
1500000	8.9	12	7	100.1	113.3	85.0	21.6	22	21	157.1	165.0	141.7
2000000	8.9	12	7	132.8	152.0	112.0	22.0	22	21	209.0	220.0	186.0

Table 2: Weakly correlated test

6.2 Support Vector Machines

A natural application of the continuous quadratic knapsack problem is to perform the projections in a gradient projection method for minimizing a general function subject to quadratic knapsack constraints. Such problems arise naturally, for example, in the training of Support Vector Machines (SVM) where a general strictly convex quadratic must be minimized [7, 10]. More precisely, given a labeled training sample of n points

$$D = \{(z_i, w_i) \mid z_i \in \mathbb{R}^m, w_i \in \{-1, 1\}\},$$

the objective is to find a classification function $F : \mathbb{R}^m \mapsto \{-1, 1\}$ of the form

$$F(z) = \text{sign}\left(\sum_{i=1}^{n} x_i^* w_i K(z, z_i) + b^*\right)$$

which is used to classify new examples z. The map $K : \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$ is known as the kernel function. A commonly used kernel, which we also adopt in our

Dimension	Iterations			Time (msec)			Iterations			Time (msec)		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
	Newton						**Secant**					
50000	4.8	7	3	2.1	2.3	1.7	8.1	12	6	3.0	4.5	1.9
100000	4.7	9	3	4.3	4.7	3.4	8.0	11	6	6.2	8.9	4.5
500000	5.0	8	3	24.4	30.0	18.5	8.3	12	6	32.4	46.5	23.5
1000000	4.9	8	3	49.6	61.0	37.0	8.2	12	6	64.4	94.0	46.0
1500000	5.0	9	3	73.9	95.0	58.3	8.3	12	6	101.1	138.3	68.3
2000000	4.9	7	3	98.3	114.0	74.0	8.1	11	6	128.1	188.0	92.0
	Variable fixing						**Median search**					
50000	7.6	9	7	2.5	2.9	2.3	16.7	17	16	4.1	4.2	3.7
100000	7.8	10	6	5.2	5.7	4.5	17.7	18	17	8.1	8.4	7.3
500000	8.1	10	7	31.0	34.0	27.5	19.9	20	19	45.8	48.5	41.5
1000000	8.3	10	7	63.9	70.0	58.0	20.9	21	20	92.9	96.0	86.0
1500000	8.4	11	7	95.6	105.0	88.3	21.6	22	21	155.0	161.7	140.0
2000000	8.3	10	7	128.1	140.0	116.0	21.9	22	21	207.4	216.0	194.0

Table 3: Correlated test

Dimension	Iterations			Time (msec)			Iterations			Time (msec)		
n	avg	max	min	avg	max	min	avg	max	min	avg	max	min
	Newton						**Secant**					
50000	5.9	8	4	2.2	2.4	1.6	14.2	18	9	4.1	5.2	2.6
100000	5.8	9	4	4.3	4.9	3.4	14.0	19	9	8.1	11.1	5.7
500000	6.1	9	4	26.0	32.5	19.5	14.2	19	9	43.5	59.5	29.0
1000000	6.2	9	4	53.0	66.0	38.0	14.2	19	9	86.7	119.0	54.0
1500000	6.1	11	4	78.3	100.0	56.7	14.0	21	10	128.5	196.7	91.7
2000000	6.1	8	4	105.3	128.0	76.0	14.2	16	11	173.4	200.0	132.0
	Variable fixing						**Median search**					
50000	5.9	8	4	2.2	2.4	1.6	16.6	17	16	3.4	3.6	3.2
100000	5.8	9	4	4.3	4.7	3.3	17.7	18	17	6.7	7.0	6.4
500000	6.1	9	4	26.0	32.0	19.0	19.9	20	19	38.6	40.5	36.5
1000000	6.2	9	4	52.7	64.0	37.0	20.9	21	20	78.1	81.0	75.0
1500000	6.1	11	4	78.0	96.7	56.7	21.6	22	21	133.0	138.3	126.7
2000000	6.2	8	4	105.8	126.0	74.0	21.9	22	21	177.6	184.0	172.0

Table 4: Multicommodity flow test

experiments, is given by the Gaussian kernel $K(u, v) = \exp(\frac{1}{2\sigma^2}\|u - v\|_2^2)$ where $\sigma > 0$.

In order to find the vector $x^* \in \mathbb{R}^n$ that defines an optimal classifier, the SVM framework requires to solve the problem

$$\begin{array}{ll} \min & \frac{1}{2}x^t Hx - e^t x \\ s.t. & w^t x = 0, \\ & 0 \leq x \leq Ce, \end{array} \tag{10}$$

where the matrix H has entries $H_{ij} = K(z_i, z_j)$, $e \in \mathbb{R}^n$ is a vector of ones, and C is a positive parameter. This matrix is positive definite but not diagonal, so we cannot use directly the methods described previously. An alternative is to use a projected gradient algorithm such as the efficient Spectral Projected Gradient (SPG) method or some of its variants [1, 10]. These methods require projections onto the feasible set at each iteration, which leads precisely to a sequence of continuous quadratic knapsack problems.

We also take this opportunity to compare with a very recent method that

appeared first as the `BLGnapsack` subroutine in the BLG code from Gonzalez-Lima, Hager and Zhang [13]. This method was specially developed to solve the projection problems that arise in projected gradients method like SPG and BLG, using a heap structure to store and efficiently manipulate the breakpoints. See details and extensions in the forthcoming report [11]. `BLGnapsack` exploits the fact that the projection is Euclidean, that is, the matrix D in (1) is the identity, and that the vector b is a vector of ones, after the signs of w are converted to become positive as assumed since Section 3. In order to make the comparison with the other methods fair, they were adapted to exploit these features using a specially tailored code generator.

In our experiments we considered the same classification problems used in [10] to test the secant method. The first problem is based on the MNIST database of handwritten digits, downloadable from `http://yann.lecun.com/exdb/mnist/`. The goal is to design an optimal classifier to decide whether a 20×20 pixel image represents the digit 8 or not. We created five instances of dimensions $n \in \{800, 1600, 3200, 6400, 11702\}$ using as training sample the first $n/2$ occurrences of digits 8 and the first $n/2$ occurrences of other digits. The first three dimensions were already used in [10]. The parameters for the SVM were chosen as $C = 10$ and $\sigma = 5$.

The second problem is based on the UCI Adult database downloadable from `http://archive.ics.uci.edu/ml/datasets/Adult`. The problem is to predict, using only the data from the 1994 US Census, whether an adult earns more than US$ 50,000 per year or not. We extracted instances of dimensions $n \in \{1605, 2265, 3185, 6370, 12740\}$, using $C = 1$ and $\sigma = \sqrt{10}$ as the parameters for the SVM.

Our implementation is based on the SPG code from [2] adapted to solve (10). The SPG method is very sensitive numerically and may generate different iterate sequences when using the alternative algorithms in the projection step. In order to compare the relative efficiency of the projection algorithms on the basis of the *same* sequence of subproblems, we only considered the SPG iterates generated by the median search method. At each one of these iterates, the projection subproblem is solved using each algorithm (Newton, secant, BLGnapsack, variable fixing, median search) with $10^6/n$ repetitions to get a reliable estimate of the running times.

The results, presented in Table 5, are not as clear-cut as in the case of random instances. On the UCI Adult test the variable fixing algorithm outperforms Newton in the smaller instances, but looses ground in the largest ones. In MNIST application it is the secant method that has a very similar performance when compared to Newton, but never really being the fastest.

The previous experiments did not exploit an important feature of the Newton's, secant and `BLGnapsack` methods. Namely, they methods can speed up their performance by using hot starts from a good initial guess of the solution, something which is not natural for the variable fixing and median search methods. This fact was already used by Dai and Fletcher to speed up the secant method in the SVM training and it is considered very important for the good performance of `BLGnapsack` [11]. Dai and Fletcher used the multiplier of the

Set	n	Newton		Secant		BLGnap.	Var. Fix		Med. Search	
		Iter.	Time	Iter.	Time	Time	Iter.	Time	Iter.	Time
UCI	1065	5.01	0.053	8.24	0.052	0.049	7.90	0.034	11.01	0.203
UCI	2265	5.25	0.141	9.02	0.175	0.184	8.53	0.126	12.58	0.448
UCI	3185	5.23	0.227	8.94	0.303	0.324	8.16	0.234	12.59	0.630
UCI	6370	5.49	0.570	9.52	0.851	0.746	8.71	0.627	13.73	1.269
UCI	12740	5.83	1.258	9.99	1.940	1.617	9.41	1.392	14.65	2.519
MNIST	800	3.15	0.027	6.79	0.029	0.027	5.81	0.035	10.66	0.136
MNIST	1600	3.26	0.057	7.01	0.061	0.084	5.95	0.070	11.83	0.285
MNIST	3200	3.49	0.143	7.13	0.136	0.228	6.38	0.182	12.83	0.585
MNIST	6400	3.45	0.316	7.26	0.324	0.441	6.76	0.471	13.68	1.155
MNIST	11702	3.60	0.637	7.30	0.682	0.768	7.38	0.977	14.60	2.091

Table 5: Results for the projection in SVM training in UCI Adult and MNIST train sets. The values reported are the average number of iterations and average computing time in milliseconds among all the projections performed while solving (10) using the SPG code. Once again, to get a better CPU time estimate, each projection is repated in a loop $10^6/n$ times For `BLGnapsack` only the time is reported.

last projection as starting point for the next one [10]. We propose a different initial point, namely, we only use the last multiplier to determine the active face of the bound constraints, and compute the multiplier by projecting the current point onto the hyperplane within this face. This initial multiplier, inspired by the variable fixing algorithm, combines the information of the last projection by means of its active face with the information of the current point to be projected. This multiplier estimate performed better than Dai and Fletcher's suggestion.

Table 6 reports the results with hot starts and includes a variation of the Newton method where variable fixing is turned off, called Newton NF. These results confirm a significant speedup for both Newton variants and the secant method. In these runs Newton outperforms all the other algorithms. The Newton variant without variable fixing is even better. The reason for this is that the number of iterations is so small that it is not worthy to spend time fixing variables, an $O(n)$ operation. It is better to perform one or two extra iterations and finish the computation.

References

[1] E.G. Birgin, J.M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods for convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[2] E.G. Birgin, J.M. Martínez, and M. Raydan. Algorithm 813: SPG—Software for Convex-Constrained Optimization. *ACM Transactions on Mathematical Software*, 27(3):340–349, September 2001.

Training set	n	**Newton**		**Newton NF**		**Secant**		`BLGnap.`
		Iter.	Time	Iter.	Time	Iter.	Time	Time
UCI Adult	1065	2.11	0.026	2.11	0.021	4.70	0.038	0.043
UCI Adult	2265	1.79	0.065	1.79	0.061	4.05	0.110	0.133
UCI Adult	3185	2.49	0.152	2.49	0.145	5.45	0.238	0.249
UCI Adult	6370	2.65	0.411	2.65	0.411	5.75	0.606	0.576
UCI Adult	12740	2.72	0.898	2.72	0.911	5.83	1.292	1.218
MNIST	800	1.95	0.020	1.95	0.015	4.45	0.021	0.019
MNIST	1600	2.13	0.048	2.13	0.035	4.73	0.051	0.053
MNIST	3200	2.36	0.134	2.36	0.110	5.16	0.144	0.189
MNIST	6400	2.19	0.302	2.19	0.264	4.90	0.310	0.371
MNIST	11702	2.64	0.656	2.64	0.587	5.70	0.677	0.824

Table 6: Results for the projections in SVM training using hot starts. The computing time required to estimate the initial active face and the corresponding multiplier is included in the reported measurements.

[3] G.R. Bitran and A.C. Hax. Disaggregation and resource allocation using convex knapsack problems with bounded variables. *Management Science*, 27(4):431–441, 1981.

[4] K. Bretthauer and B. Shetty. Quadratic resource allocation with generalized upper bounds. *Operations Research Letters*, 20(2):51–57, 1997.

[5] K.M. Bretthauer, B. Shetty, and S. Syam. A branch- and bound-algorithm for integer quadratic knapsack problems. *ORSA Journal on Computing*, 7:109–116, 1995.

[6] P. Brucker. An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters*, 3(3):163–166, 1984.

[7] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[8] P.H. Calamai and J.J. Moré. Quasi-Newton updates with bounds. *SIAM Journal on Numerical Analysis*, 24(6):1434–1441, 1987.

[9] S. Cosares and D.S. Hochbaum. Strongly polynomial algorithms for the quadratic transportation problem with a fixed number of sources. *Mathematics of Operations Research*, 19(1):94–111, 1994.

[10] Y.H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Mathematical Programming*, 106(3):403–421, 2006.

[11] Timothy A Davis, William W Hager, and James T Hungerford. The separable convex quadratic knapsack problem. Working paper, 2013.

[12] M.C. Ferris and T.S. Munson. Semismooth support vector machines. *Mathematical Programming*, 101:185–204, 2004.

[13] Maria D. Gonzalez-Lima, William W. Hager, and Hongchao Zhang. An Affine-Scaling Interior-Point Method for Continuous Knapsack Constraints with Application to Support Vector Machines. *SIAM Journal on Optimization*, 21(1):361–390, January 2011.

[14] M. Held, Ph. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[15] R. Helgason, J. Kennington, and H. Lall. A polynomially bounded algorithm for a singly constrained quadratic program. *Mathematical Programming*, 18(1):338–343, 1980.

[16] D.S. Hochbaum and S-P. Hong. About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Mathematical Programming*, 69(1-3):269–309, 1995.

[17] K.C. Kiwiel. On Floyd and Rivest's SELECT algorithm. *Theoretical Computer Science*, 347(1-2):214–238, 2005.

[18] K.C. Kiwiel. On linear-time algorithms for the continuous quadratic knapsack problem. *Journal of Optimization Theory and Applications*, 134(3):549–554, 2007.

[19] K.C. Kiwiel. Breakpoint searching algorithms for the continuous quadratic knapsack problem. *Mathematical Programming*, 112(2):473–491, 2008.

[20] K.C. Kiwiel. Variable fixing algorithms for the continuous quadratic knapsack problem. *Journal of Optimization Theory and Applications*, 136(3):445–458, 2008.

[21] N. Maculan, C.P. Santiago, E.M. Macambira, and MHC Jardim. An $O(n)$ algorithm for projecting a vector on the intersection of a hyperplane and a box in $\mathbb{R}^n$. *Journal of Optimization*, 117(3):553–574, 2003.

[22] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[23] C. Michelot. A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 50(1):195–200, 1986.

[24] S.S. Nielsen and S.A. Zenios. Massively parallel algorithms for singly constrained convex programs. *INFORMS Journal on Computing*, 4(2):166–181, 1992.

[25] P.M. Pardalos and N. Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Mathematical Programming*, 46(1-3):321–328, 1990.

[26] A.G. Robinson, N. Jiang, and C.S. Lerme. On the continuous quadratic knapsack problem. *Mathematical Programming*, 55(1-3):99–108, 1992.

[27] Mutsuo Saito and Makoto Matsumoto. SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In Alexander Keller, Stefan Heinrich, and Harald Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 607–622. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] B. Shetty and R. Muthukrishnan. A parallel projection for the multicommodity network model. *The Journal of the Operational Research Society*, 41(9):837–842, 1990.

[29] J.A. Ventura. Computational development of a lagrangian dual approach for quadratic networks. *Networks*, 21(4):469–485, 1991.