Math. Program., Ser. A 106, 403–421 (2006)

Digital Object Identifier (DOI) 10.1007/s10107-005-0595-2

Yu-Hong Dai · Roger Fletcher

New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds ⋆

Received: February 2, 2003 / Accepted: March 3, 2005
Published online: October 12, 2005 –

Abstract. There are many applications related to singly linearly constrained quadratic programs subjected to upper and lower bounds. In this paper, a new algorithm based on secant approximation is provided for the case in which the Hessian matrix is diagonal and positive definite. To deal with the general case where the Hessian is not diagonal, a new efficient projected gradient algorithm is proposed. The basic features of the projected gradient algorithm are: 1) a new formula is used for the stepsize; 2) a recently-established adaptive non-monotone line search is incorporated; and 3) the optimal stepsize is determined by quadratic interpolation if the non-monotone line search criterion fails to be satisfied. Numerical experiments on large-scale random test problems and some medium-scale quadratic programs arising in the training of Support Vector Machines demonstrate the usefulness of these algorithms.

1. Introduction

In this paper we consider the following quadratic programming problem

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \\ & \mathbf{a}^T\mathbf{x} = b, \end{aligned} \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric but may be indefinite, $\mathbf{a}$, $\mathbf{c}$, $\mathbf{l}$ and $\mathbf{u}$ (with $\mathbf{l} < \mathbf{u}$) are vectors in $\mathbb{R}^n$, and b is a scalar. Thus there is a single linear equality constraint in the problem, in addition to simple bounds (box constraints) on the variables. We refer to this as the general SLBQP problem.

There are many real-world instances of SLBQP problems. For example, some problems in multicommodity network flow and logistics have the form (1.1) in which the matrix A is diagonal (see Held *et al* [13], Meyer [16], Pardalos and Rosen [19]). The general SLBQP is also solved in training the learning methodology known as Support Vector Machine (SVM) (see Vapnik [25]). This SVM learning methodology has empirically been shown to give good performance on a wide variety of problems such as handwritten

Y.-H. Dai: State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering computing, Academy of Mathematics and System Science, Chinese Academy of Sciences, PO Box 2719, Beijing 100080, PR China. e-mail: dyh@lsec.cc.ac.cn

R. Fletcher: Department of Mathematics, University of Dundee, Dundee DD1 4HN, Scotland, UK. e-mail: fletcher@maths.dundee.ac.uk

⋆ This work was supported by the EPRSC in UK (no. GR/R87208/01) and the Chinese NSF grants (no. 10171104 and 40233029).

character recognition, face detection, pedestrian detection, and text categorization (see Platt [20]).

In this paper, we consider new algorithms for solving SLBQP problems. In the special case that A is diagonal and positive definite, it is possible to have algorithms that only require $O(n)$ operations per iteration. Helgason *et al* [14] propose an $O(n \log n)$ algorithm that is based on appropriate manipulation of the corresponding Kuhn-Tucker conditions. Several linear time algorithms have been proposed by Brucker [4], Calamai and Moré [5], and Pardalos and Kovoor [18] that are based on a similar characterization of the solution. The algorithms in [4] and [5] are based on binary search. For large practical problems, [18] proposes a randomized algorithm that runs in expected linear time and has a very small time constant. A new simple and very efficient algorithm based on secant approximation is proposed in Section 2 of this paper for this special case.

Next we consider SLBQP problems in which A is non-diagonal. One possibility is to solve these using a standard solver for the general QP problem. This approach is adequate for small problems, but may not work well if the dimension n is large. For example, active set methods may require many iterations if the initial active set and the optimal active set are significantly different and only one constraint is dropped or added at each iteration. Also the methods become inefficient if the reduced Hessian matrix becomes large. A successful way of avoiding these difficulties for box constrained QP (BQP) problems has been to use gradient projection methods. This idea dates back a long way, and various references can be found in Dai and Fletcher [7], where we use this idea in conjunction with the Barzilai-Borwein [1] (BB) stepsize formulae. For BQP problems, gradient projection methods take advantage of the fact that projection on to the box is a trivial calculation. However, a line search must be used to ensure global convergence. In fact, gradient projection methods are attractive for any type of optimization problem in which projection on to a (convex) feasible set can be done efficiently. Birgin, Martínez, and Raydan [2] propose a general framework based on gradient projection, the BB formula, and a non-monotonic line search. For the SLBQP problem, projection on to the feasible set of (1.1) can also be done very efficiently using one of the algorithms described in the previous paragraph. Thus in Section 3 of this paper we explore the practical implementation of this type of method. A new choice for the stepsize is proposed. Following our experience in [7], we use a different adaptive non-monotonic line search. In addition, we take the exact one-dimensional minimizer if the initial trial stepsize fails to reach the non-monotone line search criterion.

In Section 4 we test our approach on randomly generated test problems of moderately large dimension, both for positive definite and for indefinite Hessian matrices. We also explore the real-world applications of SVM learning methodology. The problems arising from these applications have dense Hessian matrices and their dimensions can be very large ($n \approx 10^6$, say). In this case, standard QP solvers based on explicit storage of A is impractical due to the difficulty of storing the Hessian. Even in the case that n is not all that large, which we consider here, standard QP solvers can be inefficient. Serafini, Zanghirati and Zanni [23] have implemented two projection gradient algorithms for medium-scale SVM problems and have reported numerical results that are much better than those are obtained with the QP solvers pr_LOQO and MINOS, as suggested in the SVM^{light} package by Joachims [15]. Moreover, in combination with a decomposition technique, it is shown in [23] that projection algorithms are superior to the SVM^{light}

software (version 3.5) equipped with pr_LOQO for SVM problems of up to 60, 000 variables. In Section 4, we will report numerical results of our gradient projection algorithm for some medium-scale SVM problems. Conclusion and discussion are made in the last section.

2. An algorithm for the diagonal and strictly convex case

In this section we develop an algorithm for the special case of (1.1) in which $A = \text{diag}(d_1, d_2, \ldots, d_n)$ is a positive definite diagonal matrix. The positivity of the d_i is important for the development of the algorithm, and we shall show at the end of the section that there are difficulties to be surmounted if the positive definite condition is to be relaxed. We shall refer to the resulting algorithm as *Algorithm 1*. The algorithm has some features in common with previous work, notably that it runs in expected linear time, but also includes some new features. The description of the algorithm aims to provide a more simple exposition of the underlying principles than appears elsewhere.

The algorithm is based on constructing a Lagrangian penalty function

$$\phi(\mathbf{x};\ \lambda) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{c}^T\mathbf{x} - \lambda(\mathbf{a}^T\mathbf{x} - b), \tag{2.1}$$

in which λ is a scalar parameter. Because A is positive definite, no augmentation term is needed. For any fixed value of λ, we may solve the box constrained QP problem

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \phi(\mathbf{x}) \\ \text{subject to} & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \end{array} \tag{2.2}$$

giving a minimizer which we denote by $\mathbf{x}(\lambda)$. Then λ is adjusted in an outer secant-like method to solve the single nonlinear equation

$$r(\lambda) := \mathbf{a}^T\mathbf{x}(\lambda) - b = 0 \tag{2.3}$$

in one variable λ. The minimizer of $\phi(\mathbf{x})$ is readily obtained because (2.2) separates into n problems, each in one variable x_i. The unconstrained minimizer of each problem is the solution of the equation $d_i x_i = c_i + \lambda a_i$, so the solution of (2.2) is given by

$$\mathbf{x}(\lambda) = \text{mid}(\mathbf{l},\ \mathbf{h},\ \mathbf{u}), \tag{2.4}$$

where $\mathbf{h} = \mathbf{h}(\lambda)$ has components $h_i = (c_i + \lambda a_i)/d_i$, and $\text{mid}(\mathbf{l},\ \mathbf{h},\ \mathbf{u})$ is the componentwise operation that supplies the median of its three arguments.

If a value λ^* is located such that $\mathbf{a}^T\mathbf{x}(\lambda^*) = b$, it follows that KT conditions for (1.1) are satisfied. This is because the KT conditions for (2.2) are also necessary for (1.1), and together with the feasibility condition $\mathbf{a}^T\mathbf{x} = b$, they make up the KT conditions for (1.1). It follows when (2.3) is solved that $\mathbf{x}(\lambda^*)$ is a KT point for (1.1) and hence a global minimizer by convexity.

We now consider how the equation $r(\lambda) = 0$ might be solved. It is possible that the constraints of (1.1) are inconsistent, in which case $r(\lambda) = 0$ has no solution, a possibility which we ignore at present and return to towards the end of this section. We note that $a_i x_i$ is a piecewise linear continuous function of λ by virtue of (2.4), which takes either

Bracketing Phase of Algorithm 1

calculate $\mathbf{x}$ by (2.4); $r = \mathbf{a}^T\mathbf{x} - b$;
if $r < 0$
    $\lambda_l = \lambda$; $r_l = r$; $\lambda = \lambda + \Delta\lambda$;
    calculate $\mathbf{x}$ by (2.4); $r = \mathbf{a}^T\mathbf{x} - b$;
    while $r < 0$
        $\lambda_l = \lambda$; $r_l = r$; $s = \max(r_l/r - 1, 0.1)$;
        $\Delta\lambda = \Delta\lambda + \Delta\lambda/s$; $\lambda = \lambda + \Delta\lambda$;
        calculate $\mathbf{x}$ by (2.4); $r = \mathbf{a}^T\mathbf{x} - b$;
    end
    $\lambda_u = \lambda$; $r_u = r$;
else
    $\lambda_u = \lambda$; $r_u = r$; $\lambda = \lambda - \Delta\lambda$;
    calculate $\mathbf{x}$ by (2.4); $r = \mathbf{a}^T\mathbf{x} - b$;
    while $r > 0$
        $\lambda_u = \lambda$; $r_u = r$; $s = \max(r_u/r - 1, 0.1)$;
        $\Delta\lambda = \Delta\lambda + \Delta\lambda/s$; $\lambda = \lambda - \Delta\lambda$;
        calculate $\mathbf{x}$ by (2.4); $r = \mathbf{a}^T\mathbf{x} - b$;
    end
    $\lambda_l = \lambda$; $r_l = r$;
end

the constant values $a_i l_i$ or $a_i u_i$, or the value $a_i(c_i + \lambda a_i)/d_i$, and hence is either constant or a monotonically increasing function of λ. It follows that the same is true of $r(\lambda)$. We note that $r(\lambda)$ might be constant over part of its domain. In fact we may interpret $r(\lambda)$ as the gradient of a dual function of λ after eliminating the primal variables.

Although it would be a simple matter to evaluate the slope of the function $r(\lambda)$ (except at a breakpoint), to be used in an iteration of the Newton-Raphson method, the piecewise linear nature of $r(\lambda)$ makes this unappealing. For example the correction would be infinite on a constant piece of the graph. Yet we would wish to use linear approximation when the linear piece that crosses the λ axis is located, so as to locate λ^* exactly. Thus we have chosen a secant-type approach, with modifications to promote rapid global convergence. Our algorithm divides into two parts, a *Bracketing phase* in which a bracket on a solution is sought, followed by a *Secant phase* in which the bracket is uniformly reduced until the piece which crosses the λ axis is located, and the process terminates.

In the bracketing phase we ask the user to supply an initial value of λ and an initial estimate $\Delta\lambda > 0$ of the likely change to λ. The bracketing phase may be described by the following segment of pseudo-code in which $\mathbf{a}$, b, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{l}$, $\mathbf{u}$ refer to the data that defines (1.1). If a value of $r(\lambda)$ is located which is sufficiently close to zero, then the process terminates. If $r(\lambda) < 0$ then the search for a bracket takes place in the positive λ direction, else if $r(\lambda) > 0$ then in the negative λ direction. If the problem (1.1) is feasible, then the bracketing phase is guaranteed to terminate with a bracket $[\lambda_l,\ \lambda_u]$ which contains a solution of the equation $r(\lambda) = 0$. The calculation of s in the code is such that $\Delta\lambda/s$ is the correction to λ that gives either a secant step based on the two most recent iterates, or a step of 10 times the previous step, whichever is the smaller. In practice, examples were found in which $\Delta\lambda/s$ could be much smaller than $\Delta\lambda$, giving rise to slow convergence. Hence the updated value $\Delta\lambda = \Delta\lambda + \Delta\lambda/s$ has been used, so as to ensure that the new value of $\Delta\lambda$ is greater than the old value. Perhaps a more appealing

Secant phase of Algorithm 1

```
s = 1 − r_l/r_u; Δλ = Δλ/s; λ = λ_u − Δλ;
calculate x by (2.4); r = a^T x − b;
while not converged
    if r > 0
        if s ≤ 2
            λ_u = λ; r_u = r; s = 1 − r_l/r_u;
            Δλ = (λ_u − λ_l)/s; λ = λ_u − Δλ;
        else
            s = max(r_u/r − 1, 0.1); Δλ = (λ_u − λ)/s;
            λ_new = max(λ − Δλ, 0.75 λ_l + 0.25 λ);
            λ_u = λ; r_u = r; λ = λ_new;
            s = (λ_u − λ_l)/(λ_u − λ);
        end
    else
        if s ≥ 2
            λ_l = λ; r_l = r; s = 1 − r_l/r_u;
            Δλ = (λ_u − λ_l)/s; λ = λ_u − Δλ;
        else
            s = max(r_l/r − 1, 0.1); Δλ = (λ − λ_l)/s;
            λ_new = min(λ + Δλ, 0.75 λ_u + 0.25 λ);
            λ_l = λ; r_l = r; λ = λ_new;
            s = (λ_u − λ_l)/(λ_u − λ);
        end
    end
    calculate x by (2.4); r = a^T x − b;
end
```

way of achieving this would have been to define $s = \min(1, \max(r_l/r - 1, 0.1))$ and to use $\Delta\lambda = \Delta\lambda/s$. This would enable the secant step to be accepted in some cases, and hence give termination if the current piece intersects the λ axis. However, in practice the difference between these strategies is likely to be small.

Once a bracket is located, the second stage is to find the solution of the equation $r(\lambda) = 0$ by repeated use of the secant method, with certain modifications designed to speed up convergence remote from the solution. Again the process is defined by some pseudo-code. Initially values of λ_l and λ_u are available with $r(\lambda_l) < 0$ and $r(\lambda_u) > 0$, and a secant step to a new point λ is taken. If $r(\lambda) > 0$ then the iteration proceeds as follows. If λ lies in the left half of the interval $[\lambda_l,\ \lambda_u]$ (that is $s \leq 2$), then a secant step based on λ_l and λ is taken on the next iteration. If λ lies in the right half of the interval, then either a secant step based on λ and λ_u, or a step to the point $\frac{3}{4}\lambda_l + \frac{1}{4}\lambda$ is taken, whichever is the smaller step. This ensures that the interval length is reduced by a factor of $\frac{3}{4}$ or less. In both cases λ_u is replaced by λ to give a new bracket. Similar decisions are taken if $r(\lambda) < 0$ at the start of the iteration. We terminate the secant phase if preset tolerances on either $r(\lambda)$ or $\Delta\lambda$ are met. Of course the parameters in the above pieces of code are somewhat arbitrary, but are of an appropriate size. Although it remains to be examined, we feel that small changes in these values would be unlikely to change the effectiveness of the method to any great extent.

We now return to the issue of how to decide if the constraints of (1.1) are consistent. If a bracket is found during the bracketing phase, then it follows that the constraints are consistent, and no further calculation is required. Alternatively, we may define the vectors

$$\mathbf{a}_+ = \max(\mathbf{a}, \mathbf{0}), \qquad \mathbf{a}_- = \min(\mathbf{a}, \mathbf{0}) \tag{2.5}$$

to be the positive and negative parts of $\mathbf{a}$, respectively. Since the linear function has a maximum value of $\mathbf{a}_+^T\mathbf{u} + \mathbf{a}_-^T\mathbf{l}$ and a minimum value of $\mathbf{a}_+^T\mathbf{l} + \mathbf{a}_-^T\mathbf{u}$ on the set $\{\mathbf{x} \mid \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$, we can conclude inconsistency if either of the conditions

$$\mathbf{a}_+^T\mathbf{u} + \mathbf{a}_-^T\mathbf{l} < b, \qquad \mathbf{a}_+^T\mathbf{l} + \mathbf{a}_-^T\mathbf{u} > b \tag{2.6}$$

hold, and consistency otherwise. However, this requires a certain amount of calculation, which might be avoided if the problem is known to be consistent, or if a bracket is found quickly. In our codes we have resolved this matter in the following way. The user is asked to supply a parameter *ktest*. If a bracket is not found in *ktest* iterations then the calculation in (2.6) is carried out to decide on consistency. If the user knows that the problem is consistent, then *ktest*$= \infty$ can be set. Otherwise we suggest using the value *ktest=4*.

Finally we examine whether it is possible to relax the condition that the A is positive definite. Consider the case that $d_i = 0$ occurs for some value of i. The minimization of $\phi(\mathbf{x}, \lambda)$ still separates into n one-variable problems, and hence is readily solved. In the case of x_i, we simply set

$$x_i = \begin{cases} l_i & \text{if } c_i + \lambda a_i < 0 \\ u_i & \text{if } c_i + \lambda a_i > 0 \end{cases}.$$

The difficulty is that x_i, and hence $\mathbf{x}(\lambda)$, is no longer continuous at a value of λ at which the solution x_i switches from one bound to the opposite bound. The difficulties that arise are illustrated by a simple example. Take $n = 2$, $d_1 = 1$, $d_2 = 0$, $\mathbf{c} = (1, 1)^T$, $\mathbf{a} = (2, 1)^T$, $b = 1$, $\mathbf{l} = \mathbf{0}$ and $\mathbf{u} = (2, 2)^T$. The solution is at $\mathbf{x}^* = (0, 1)^T$ with multiplier $\lambda^* = -1$. The problem is convex and $\mathbf{x}^*$ is the unique global solution, so the problem is in no sense badly behaved. However, consider the graph of the function $r(\lambda)$. For $\lambda < -1$ the minimizer of $\phi(\mathbf{x}, \lambda)$ is $\mathbf{x} = (0, 2)^T$, whereas for $-1 < \lambda \leq \frac{1}{2}$ it is $\mathbf{x} = (0, 0)^T$. Thus $r(\lambda)$ is piecewise constant in the neighbourhood of $\lambda < -1$, and jumps from -1 to $+1$ as λ passes through -1. The solution is determined by choosing $\mathbf{x}$ to solve $\mathbf{a}^T\mathbf{x} = b$ at the point of discontinuity in λ. The discontinuity in $r(\lambda)$ might adversely affect the behaviour of the secant algorithm described above. In particular we cannot expect the algorithm to terminate by locating a linear piece that crosses the λ axis. In the case that A is diagonal and positive semi-definite, one way to circumvent these difficulties is to calculate the values of λ such that $c_i + \lambda a_i = 0$ for $i \in \{i : d_i = 0\}$. The corresponding values of $r(\lambda)$ can then be used to determine an interval that includes the optimal multiplier. If $d_i \neq 0$ for all i but $d_i < 0$ for some values of i, then the problem may have many KT points. In this case, we can still define the one-variable function $r(\lambda)$ as before. However, this function is not in general monotonic, but is still continuous. Methods based on its continuity may be developed to calculate one KT point of the problem.

3. A projected gradient algorithm for the general case

In this section we consider an algorithm for solving (1.1) when A is not a diagonal matrix. The algorithm has the same framework as that of the SPG2 algorithm of Birgin,

Martínez and Raydan [2]. However a new choice for the steplength is used here, and the adaptive non-monotone line search from [7] is incorporated. Our numerical experiments in the next section show that these new techniques are very useful in practice. An overview of the algorithm, referred to as *Algorithm 2*, is

Overview of Algorithm 2
Initialization
for $k = 1, 2, \ldots$ until converged
    Calculate the projection step using Algorithm 1,
    Possibly carry out a line search,
    Calculate a BB-like step length,
    Update the line search control parameters
end

Let us denote the feasible region of (1.1) by Ω. The projection of any vector $\mathbf{z} \in \mathbb{R}^n$ on to Ω is the minimizer of the problem

$$\min\{\tfrac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 : \ \mathbf{x} \in \Omega\}. \tag{3.1}$$

This is a special case of an SLBQP problem in which $A = I$ is the identity matrix, and hence it can be solved efficiently by for example Algorithm 1. On average, we find that only about five iterations are required by Algorithm 1. The first stage in the loop of Algorithm 2 is to take a steepest descent step from a current iterate $\mathbf{x}_k$ with fixed steplength α_k, and then project the resulting point on to Ω. If $\mathbf{g} = A\mathbf{x} - \mathbf{c}$ denotes the gradient vector, we may express this as

$$\mathbf{d}_k = P_\Omega(\mathbf{x}_k - \alpha_k \mathbf{g}_k) - \mathbf{x}_k \tag{3.2}$$

where $P_\Omega(\mathbf{z})$ represents the projection of $\mathbf{z}$ on to Ω. We note that the projection operation has the property that $(\mathbf{x} - P_\Omega(\mathbf{z}))^T(\mathbf{z} - P_\Omega(\mathbf{z})) \leq 0$ for all $\mathbf{x} \in \Omega$, and it readily follows if $\mathbf{d}_k \neq \mathbf{0}$ that $\mathbf{d}_k$ is a feasible descent direction. The solution of (1.1) is characterised by $P_\Omega(\mathbf{x}^* - \mathbf{g}^*) = \mathbf{x}^*$, so the algorithm is deemed to have converged when $\|P_\Omega(\mathbf{x}_k - \mathbf{g}_k) - \mathbf{x}_k\|$ is within a prescribed tolerance. Although this is how convergence was recognised in our codes, it does require an extra projection calculation, and a less expensive alternative in the case that α_k is suitably sized to test $\|P_\Omega(\mathbf{x}_k - \alpha_k \mathbf{g}_k) - \mathbf{x}_k\|/\alpha_k$ (that is $\|\mathbf{d}_k\|/\alpha_k$).

The second stage of Algorithm 2 is to decide if a line search is necessary. A feature of BB type algorithms is that they are inherently non-monotonic, that is to say the objective function value $f(\mathbf{x}_k)$ must be allowed to increase on some iterations in order for the methods to work well (see Fletcher [10]). When minimizing quadratic functions without any constraints, convergence can be established without introducing a line search (see Raydan [21]). However we show in [7] for box constrained QP that it is possible (albeit unlikely) for the algorithm to fail, unless some provision for a line search is made. The same will certainly be true for the SLBQP problem. However it is important that a non-monotonic line search is used, a fact first recognised by Raydan [22] for general unconstrained minimization. In [22] and [2], a non-monotonic line search of the type suggested by Grippo, Lampariello and Lucidi [12] is used. In [7] we show that an alternative adaptive non-monotonic line search is preferable, as it cuts down the number of times

the line search is brought into play, and hence enables the non-monotonic aspects of the BB method to operate more freely. All these non-monotonic algorithms make use of a control parameter $f_{ref} \geq f(\mathbf{x}_k)$, and no line search is carried out if $f(\mathbf{x}_k + \mathbf{d}_k) < f_{ref}$. (In some methods a sufficient reduction on f_{ref} is required.) However, the way in which f_{ref} is defined is different.

For $k \geq 2$ we carry out a line search if $f(\mathbf{x}_k + \mathbf{d}_k) \geq f_{ref}$. We do this simply by a quadratic interpolation along $\mathbf{x}_k + \lambda \mathbf{d}_k$, using the function values $f(\mathbf{x}_k)$ and $f(\mathbf{x}_k + \mathbf{d}_k)$, and the slope $\mathbf{g}_k^T \mathbf{d}_k$. An alternative possibility would be to search along the path obtained by projecting $\mathbf{x}_k - \lambda \mathbf{g}_k$ (as in SPG1 of [2]). However this requires extra projection operations, and in any event the distinction is of little importance because we use very few line searches. Because $f(\mathbf{x})$ is quadratic, $f(\mathbf{x}_k) \leq f_{ref}$ and $f(\mathbf{x}_k + \mathbf{d}_k) \geq f_{ref}$, and $\mathbf{d}_k$ is a descent direction, it follows that we obtain an exact line search along $\mathbf{x}_k + \lambda \mathbf{d}_k$. This is different to methods based on [12] which use an Armijo line search. On iteration $k = 1$, we carry out a line search whenever $f(\mathbf{x}_k + \mathbf{d}_k) \geq f(\mathbf{x}_k)$, because the initial value of α_1 might possibly be unreliable.

The third stage of the algorithm is to compute the stepsize α_{k+1} for the next iteration. Denote the difference vectors $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$. One of the two BB formulae in [1] is defined by

$$\alpha_{k+1} = \mathbf{s}_k^T \mathbf{s}_k / \mathbf{s}_k^T \mathbf{y}_k. \tag{3.3}$$

There are many references to the fact that this choice is far more efficient than the classical steepest descent stepsize, and in Section 4 we give some results for this choice in the context of an SLBQP problem. Recently many other BB-like formulae for the stepsize have been proposed. We have therefore tried alternating the two BB formulae on successive iterations, as recommended in [7], but find this to be ineffective for random indefinite test problems and SVM test problems. Other alternative stepsize choices, such as the AS and CSDS methods described in [6], did not perform well in the box constrained QP context [7], and we have not used them here. Here we also present a new choice for the stepsize. To this end, we recall that the BB formula (3.3) can be obtained by solving the one-dimensional subproblem: minimize $\|\alpha_{k+1}^{-1} \mathbf{s}_k - \mathbf{y}_k\|_2$, in which $\alpha_{k+1}^{-1} I$ can be regarded as some approximation to the Hessian at $\mathbf{x}_{k+1}$. For each integer $i \geq 0$, we can have a similar formula if we replace the pair $(\mathbf{s}_k, \mathbf{y}_k)$ with $(\mathbf{s}_{k-i}, \mathbf{y}_{k-i})$, yielding $\alpha_{k+1} = \mathbf{s}_{k-i}^T \mathbf{s}_{k-i} / \mathbf{s}_{k-i}^T \mathbf{y}_{k-i}$ (this formula can be seen in [11]). It might be interesting to study how to pick the *best* integer i according to some rule. Here we propose a new formula based on the idea of taking an average of the most recent $m \geq 1$ difference pairs, where m is a preset integer. Hence we define the vectors $S^{(k)} = (\mathbf{s}_k^T, \ldots, \mathbf{s}_{k-m+1}^T)^T$ and $Y^{(k)} = (\mathbf{y}_k^T, \ldots, \mathbf{y}_{k-m+1}^T)^T$, and determine α_{k+1} by minimizing $\min \|\alpha_{k+1}^{-1} S^{(k)} - Y^{(k)}\|_2$. Therefore we obtain

$$\alpha_{k+1} = \frac{S^{(k)T} S^{(k)}}{S^{(k)T} Y^{(k)}} = \frac{\sum_{i=0}^{m-1} \mathbf{s}_{k-i}^T \mathbf{s}_{k-i}}{\sum_{i=0}^{m-1} \mathbf{s}_{k-i}^T \mathbf{y}_{k-i}}. \tag{3.4}$$

In fact, we let $\bar{m}$ be the maximal integer such that $\mathbf{s}_{k-i}^T \mathbf{y}_{k-i} > 0$ for all $0 \leq i \leq \bar{m}$, and we assume conventionally that $\mathbf{s}_k^T \mathbf{y}_k \leq 0$ if $k \leq 0$. Our practical algorithm then calculates the stepsize by (3.4) with m replaced by $\min(m, \bar{m})$. We also chop any extreme values

of α_k by truncating them to lie in preassigned interval $[\alpha_{min}, \alpha_{max}]$. If it happens that $\bar{m} = 0$, that is $\mathbf{s}_k^T \mathbf{y}_k \leq 0$, we simply set $\alpha_{k+1} = \alpha_{max}$. It is obvious that the formula (3.4) with $m = 1$ reduces to the BB formula (3.3). However, our numerical experiments suggest that $m = 2$ is a better choice in the BQP or SLBQP context. Results obtained using this choice of stepsize are described in Section 4.

The fourth stage of Algorithm 2 is to update the line search parameters. The line search we use is that described in [7], which is related to methods described by Toint [24] and by Dai and Zhang [8]. The parameters involved are the reference value f_{ref} referred to above, the current best function value f_{best}, a candidate value f_c for possible reduction of f_{ref}, and a preassigned number L denoting the number of iterations with $f(\mathbf{x}_k) \geq f_{best}$ that are allowed before reducing f_{ref} to f_c. The updating process may be described as follows:

if $f_k < f_{best}$,
    $f_{best} = f_k,\ f_c = f_k,\ l = 0,$
else
    $f_c = \max\{f_c, f_k\},\ l = l + 1,$
    if $l = L,\ f_{ref} = f_c,\ f_c = f_k,\ l = 0,$ end
end.

A simple argument in [7] proves global convergence in real arithmetic. An important property of this method is that if a better value of $f(\mathbf{x}_k)$ is always found in at most L iterations, then f_{ref} remains at its initial value of infinity, and no line searches are needed after the first iteration.

Algorithm 2 requires parameter settings for L, α_{min} and α_{max}, and initial values for $\mathbf{x}_1$, $f_{ref} = \infty$, $\mathtt{l} = 0$ and $f_{best} = f_c = f(\mathbf{x}_1)$. The major computational expense is that required to update the gradient vector, which can be done in $O(np)$ operations, where p is the number of non-zero components of $\mathbf{d}_k$. The number of projection calculations can also be significant, since although each iteration of Algorithm 1 only requires $O(n)$ operations, quite a large number of projection calculations may be required.

4. Numerical experiments

We tested Algorithm 1 and Algorithm 2 with Matlab 5.0 on Linux SuSE 6.4 (2 x Intel Pentium III, 750 MHZ), except that the numerical experiments with SVM problems were made with Matlab 7.0 on a Windows XP professional computer (pentium 4 , 2.60 GHZ).

At first, we compare Algorithm 1 and an algorithm based on binary search, referred to as BS algorithm, for random diagonal SLBQP test problems. The BS algorithm works on the multiset $\mathcal{S} = \mathcal{B} \cup \{-\infty,\ +\infty\}$, where $\mathcal{B} = \{(d_i l_i - c_i)/a_i,\ (d_i u_i - c_i)/a_i : i = 1, 2, \ldots, n\}$ are break points of function (2.4). If two breakpoints λ_l and λ_u have been obtained with $r(\lambda_l) < 0$ and $r(\lambda_u) > 0$, the BS algorithm picks up the exact median λ_m of the multiset $\mathcal{S}$ restricted to $[\lambda_l,\ \lambda_u]$ and updates either λ_l or λ_u based on the value $r(\lambda_m)$. See [18] for more details. If $r(\lambda_m) \neq 0$ for all λ_m's, the BS algorithm requires at least $\lfloor \log(2n + 2)/\log(2) \rfloor$ iterations, where $\lfloor v \rfloor$ stands for the largest integer not greater than v. For test problems, we consider the following three cases, where

$a_i \in [-500, 500]$ means that a_i is generated in the interval $[-500, 500]$ at a uniform distribution and $a_i \in \{-1, 1\}$ means that a_i is randomly chosen to be either -1 or 1, *etc.*:

Case I. $a_i \in [-500, 500]$, $l_i \in [-1000, 0]$, $u_i \in [0, 1000]$, $d_i \in [d_1, d_n]$ with $d_1 = 1$ and $d_n = 10^4$, $c_i \in [-1000, 1000]$;

Case II. $a_i \equiv 1$, $l_i \equiv 0$, $u_i \in [0, 1000]$, $d_i \in [d_1, d_n]$ with $d_1 = 1$ and $d_n = 10^4$, $c_i \in [-1000, 1000]$;

Case III. $a_i \in \{1, -1\}$, $l_i \equiv 0$, $u_i \equiv 1000$, $d_i \equiv 1$, $c_i \in [-1000, 1000]$;

Specifically, Case I is the general case, Case II has a similar form to practical problems arising from multicommodity network flow and logistics (for example, see [18]), and Case III has the same form as the projection subproblems from support vector machine problems. Denote by b_{min} and b_{max} the minimal and maximal value of $\mathbf{a}^T\mathbf{x}$ subject to $\mathbf{l} \le \mathbf{x} \le \mathbf{u}$, respectively. The number b in the linear constraint is chosen by the following way:

$$b = b_{min} + \delta_b\,[b_{\max} - b_{\min}], \tag{4.1}$$

where $\delta_b \in [0, 1]$ is a parameter. This way of choosing b ensures that each test problem is feasible. For symmetry, we use three different values of b: 0.1, 0.3, and 0.5. The dimension n is chosen to be 10^4, 10^5 or 10^6.

We have tested Algorithm 1 and the BS algorithm on the above three types of random diagonal SLBQP problems with MATLAB. The initial values for λ and $\Delta\lambda$ in Algorithm 1 are always chosen as 0 and 2, respectively. For each test problem, we terminate the calculation of Algorithm 1 when $|r(\lambda)|$ is smaller than the final value of this quantity as obtained by the BS algorithm. For any fixed n, δ_b and the problem type, 10 random problems were generated. Table 4.1 lists the average number of iteration and CPU time (in second) required by each algorithm.

From Table 4.1, we can see that the CPU times required by both Algorithm 1 and the BS algorithm grow linearly with the dimension n. Further, for all the cases with different n and δ_b, Algorithm 1 is uniformly better than the BS algorithm. The improvement of Algorithm 1 over the BS algorithm is substantial when $\delta_b = 0.5$. One explanation is that in this case the function $r(\lambda)$ is better approximated by a linear function over the range of values λ that are sampled.

We have tested Algorithm 2 on three kinds of test problems. The preset parameters for Algorithm 2 are chosen to be $\alpha_{\min} = 10^{-5}$, $\alpha_{\max} = 10^5$ and $L = 10$. For Algorithm 1, which is used as the projection subroutine of Algorithm 2, the initial values

Table 4.1. Comparing Algorithm 1 and the BS algorithm

n	δ_b	Case I				Case II				Case III			
		BS		Algorithm 1		BS		Algorithm 1		BS		Algorithm 1	
		iter	time	iter	time	iter	time	iter	time	iter	time	iter	time
10^4	0.1	14.2	0.156	14.9	0.141	14.4	0.157	10.8	0.122	14.1	0.142	11.6	0.109
	0.3	14.2	0.159	9.9	0.108	14.2	0.156	12.3	0.138	14.4	0.138	9.4	0.094
	0.5	14.5	0.154	4.7	0.063	14.4	0.164	14.0	0.149	14.3	0.136	4.9	0.056
10^5	0.1	17.7	1.761	14.2	1.373	17.9	1.765	11.4	1.150	17.5	1.489	11.8	0.960
	0.3	17.6	1.691	10.6	0.988	17.6	1.627	12.9	1.196	17.5	1.472	9.9	0.819
	0.5	17.6	1.714	4.1	0.468	17.8	1.701	14.3	1.339	17.7	1.497	4.5	0.446
10^6	0.1	21.0	20.253	14.6	13.679	21.0	21.175	11.5	11.898	21.1	17.523	11.8	9.371
	0.3	21.0	21.480	10.6	10.514	20.9	20.883	12.7	12.595	21.0	17.753	10.1	8.386
	0.5	21.0	19.965	4.2	4.805	21.0	19.702	14.1	12.873	20.9	17.219	4.8	4.590

for λ and $\Delta\lambda$ on the first call are chosen as 0 and 2, respectively. They are then set to λ' and $1 + |\lambda'|$ on the second call and to λ' and $1 + |\lambda' - \lambda''|$ subsequently. Here λ' and λ'' are the values of λ provided by the previous two projections. The accuracy required in the projection algorithm at the k-th iteration is $|r(\lambda)| \leq 10^{-5}$ for $k = 1$ and $|r(\lambda)| \leq 10^{-5} \min\{1, \|P_\Omega(\mathbf{x}_{k-1} - \mathbf{g}_{k-1}) - \mathbf{x}_{k-1})\|_\infty\}$ for $k \geq 2$. We also terminate the projection algorithm if $\Delta\lambda \leq 10^{-10}$. The choice of the initial stepsize α_1 is described below. The parameter m in the formula (3.4) is chosen to be $m = 2$. A comparison of different values of m is provided below for random SPD test problems. See Section 5 for further discussion on the choice of m.

In order to test the efficacy of our adaptive line search procedure, we also tested Algorithm 2 with the non-monotone line search in [12]. This line search determines the reference function value f_{ref} at the k-th iteration by

$$f_{ref} = \max\{f(\mathbf{x}_{k-i}) : i = 0, \ldots, \max\{k-1, M-1\}\}, \tag{4.2}$$

where M is some preset positive integer. Again, we take the exact one-dimensional minimizer if the initial stepsize fails to meet the line search condition. In these tests we have chosen the commonly accepted value of $M = 10$. We denote this algorithm by Algorithm 2*. In addition, if the formula (3.3) is used to calculate the stepsize, we refer to the corresponding algorithms as BB if the adaptive non-monotone line search is applied, and as BB* if the reference function value is determined by (4.2). We compare all the four algorithms: Algorithm 2, Algorithm 2*, BB and BB* for all our test problems.

Three kinds of problems are used in our tests: (1) random symmetric positive definite (SPD) test problems, (2) random indefinite test problems, and (3) medium-scale problems generated by training Gaussian SVMs on two real-world data sets. The descriptions of the test problems and the corresponding numerical results are presented as follows. We let $\mathbf{e}$ denote the column vector of ones, and the vectors $\mathbf{p}_i i = 1, 2, \ldots, 7$ denote random vectors whose elements are sampled from a uniform distribution in $[0, 1]$. Also we denote $v_{i,j}$ to be the j-th component of the vector $\mathbf{v}_i$.

Random SPD test problems. The generation of these problems is based on the generation of random SPD BQP test problems in Moré and Toraldo [17] and Dai and Fletcher [7]. At first, we generate a BQP problem using the five parameters n, $ncond$, $ndeg$, $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$ such that $\bar{\mathbf{x}}$ is the solution of the BQP problem and $\bar{\mathbf{x}}_1$ is the starting point. Specifically, we denote $A = PDP^T$, where

$$P = (I - 2\mathbf{p}_3\mathbf{p}_3^T)(I - 2\mathbf{p}_2\mathbf{p}_2^T)(I - 2\mathbf{p}_1\mathbf{p}_1^T),$$

where D is a diagonal matrix with whose i-th component is defined by

$$\log d_i = \frac{i-1}{n-1} ncond, \quad i = 1, \ldots, n.$$

The parameter $ncond$ in the above relation specifies the condition number of A. We set $\bar{\mathbf{x}} = 2\mathbf{p}_4 - 1$, namely, $\bar{\mathbf{x}}$ is chosen randomly in the interval $[-1, 1]$. The choice of active set $\mathcal{A}(\bar{\mathbf{x}})$ depends on the integer parameter $na(\bar{x})$. Random numbers μ_i in $[0, 1]$ are generated for $i = 1, \ldots, n$ and i is selected for $\mathcal{A}(\bar{x})$ if $\mu_i \leq na(\bar{\mathbf{x}})/n$. This algorithm is also used to select the active constraints $\mathcal{A}(\bar{\mathbf{x}}_1)$ at the starting point of the BQP problem

on the basis of the parameter $na(\bar{\mathbf{x}}_1)$. Components of $\bar{\mathbf{x}}_1$ that are not in $\mathcal{A}(\mathbf{x}_1)$ are set to $(l_i + u_i)/2$.

The values of $\mathbf{l}$, $\mathbf{u}$, and $\mathbf{c}$ can now be determined using the parameter $ndeg$. This parameter specifies the extent to which the resulting problem will be close to being dual degenerate. We determine the value of $\nabla f(\bar{\mathbf{x}}) = \mathbf{r}$ by setting $|r_i| = 10^{-\mu_i ndeg}$ for $i \in \mathcal{A}(\bar{\mathbf{x}})$, where μ_i is randomly generated in [0, 1]. The right hand term $\mathbf{c}$ is set to $\mathbf{c} = A\bar{\mathbf{x}} - \mathbf{r}$ and we define

$$l_i = -1, \quad u_i = +1, \quad r_i = 0,$$

for $i \notin \mathcal{A}(\bar{x})$, and

$$l_i = \bar{x}_i, \quad u_i = +1, \quad r_i > 0,$$

or

$$l_i = -1, \quad u_i = \bar{x}_i, \quad r_i < 0,$$

for $i \in \mathcal{A}(\bar{x})$. Then we have constructed an SPD BQP problem whose solution is $\bar{\mathbf{x}}$. Further, we let $\mathbf{x}_{fs}$ be the vector with components $l_i + p_{5,i}(u_i - l_i)$, $\mathbf{a} = 2\mathbf{p}_6 - \mathbf{e}$ and $b = \mathbf{a}^T\mathbf{x}_{fs}$. The starting point for the SLBQP problem is set to $\mathbf{x}_1 = P_\Omega(\bar{\mathbf{x}}_1)$. Hence we have described how to generate our random SPD SLBQP test problems by five parameters n, $ncond$, $ndeg$, $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$.

In our random SPD tests, we fix the dimension $n = 10000$ and choose the other parameters by permuting all possibilities from the ranges:

$$ncond \in \{4, 5, 6, 7\}, \quad ndeg \in \{1, 3, 5, 7, 9\}, \quad na(\bar{\mathbf{x}}), na(\bar{\mathbf{x}_1}) \in \{1000, 5000, 9000\}.$$

For each case, we generate one problem randomly. Hence the total number of random SPD problems is 180. For these problems, the first stepsize used is $\alpha_1 = \|P_\Omega(\mathbf{x}_1 - \mathbf{g}_1) - \mathbf{x}_1\|_\infty^{-1}$ and the stopping condition is

$$\|P_\Omega(\mathbf{x}_k - \mathbf{g}_k) - \mathbf{x}_k\|_\infty \le 10^{-5},$$

as used in [2].

We have tested Algorithm 2 with four different values of m, namely 1, 2, 3 and 4. If $m = 1$, Algorithm 2 reduces to the BB method. See Figure 4.1 for the performance profiles. These suggested that the preferred value of m is 2 and this has therefore been used in our other experiments with Algorithm 2.

Figure 4.2 shows the performance profiles of Algorithm 2, Algorithm 2*, BB, BB* for the random SPD problems. From Figure 4.2, it is clear that Algorithm 2 has the most wins (has the highest probability of being the optimal solver) and that the probability that Algorithm 2 is the winner on a given problem is about 45%. Moreover, Algorithm 2 solves 98% of the problems within a factor of 2 of the best performance.

Taking Algorithm 2 as an example, the average number of iterations required by Algorithm 1 for each problem lies in the interval [1.72, 7.89]. For all the problems, the average number of projection iterations is 4.19. The maximal number of iterations taken by Algorithm 1 for each problem is between 3 and 13. If the BS algorithm is used for projection, we know that the required number of iterations is at least $\lfloor \log(2n+2)/\log 2 \rfloor =$

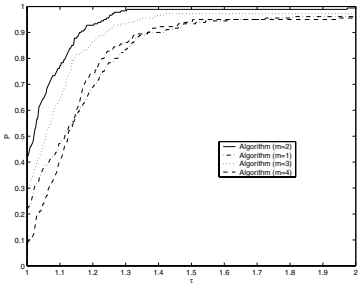

Fig. 4.1. Performance profiles for Algorithm 2 with different m's

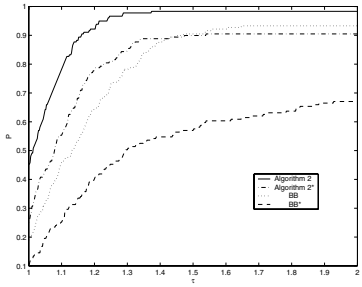

Fig. 4.2. Performance profiles for random SPD test problems

14 when $n = 9000$. These data show that both Algorithm 1 and Algorithm 2 work very well for random SPD test problems.

To analyze the dependence of the results on the parameters $ncond$, $ndeg$, $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$, we fix one of the parameters and vary the remaining three parameters. Taking Algorithm 2 as an example, it is found that the parameter $ncond$ plays an important role in determining performance. As $ncond$ ranges from 4, 5, 6 to 7, the average CPU time (in second) of the corresponding problems ranges from 9.38, 17.00, 30.58 to 56.66. When $ndeg$ takes different values in {1, 3, 5, 7, 9}, the average CPU time of the corresponding problems is about 22 seconds. This reflects the fact that Algorithm 2 is not sensitive to the dual degeneracy. The dependency of the method on the parameters $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$ is not straightforward and there is no clear pattern.

Random indefinite test problems. The generation of these problems is based on four parameters n, $ncond$, $na(\bar{\mathbf{x}}_1)$ and $negeig$. The first three parameters are used to generate the matrix A, the vector $\mathbf{c}$, and the starting point $\bar{\mathbf{x}}_1$ as before. The parameter $negeig$ is used to specify the number of negative eigenvalues of A. Given an integer $negeig \in [1, n]$, we change the sign of the i-th diagonal entry of D if $p_{7,i} \leq negeig/n$. The lower and upper bounds are set to $\mathbf{l} = -\mathbf{e}$ and $\mathbf{u} = \mathbf{e}$. The vector $\mathbf{a}$ and the scalar b are generated as before.

We have generated 48 indefinite SLBQP test problems by fixing $n = 10^4$, and choosing the other parameters by permuting all possibilities from the ranges:

$$ncond \in \{4, 5, 6, 7\}, \quad negeig \in \{1000, 2500, 5000, 9000\},$$
$$na(\bar{\mathbf{x}}) \in \{1000, 5000, 9000\}.$$

The choice of the first stepsize and the stopping condition are the same as before. Figure 4.2 shows the performance profiles of the four algorithms for the random indefinite problems. Figure 4.2 shows that BB has the most wins and Algorithm 2 ranks the second. However, Algorithm 2 can solve most of the problems within a factor 2 of the best solver.

In the indefinite case, each problem has many different local minimizers. To compare the best function values obtained by each algorithm, we introduce the following curve similar to performance profile in [9]. Suppose that $\mathcal{P}$ is the set of test problems and $\mathcal{S}$ is the set of solvers. Suppose that $f_{p,s}$ is the best function value found by solver s for

problem p. We denote by $\underline{f}_p$ and $\bar{f}_p$ the minimal and maximal values of $\{f_{p,s} : s \in \mathcal{S}\}$, respectively. For each $f_{p,s}$, we define the ratio

$$r_{p,s} = \begin{cases} 0, & \text{if } \bar{f}_p - \underline{f}_p \leq \epsilon; \\ (f_{p,s} - \underline{f}_p)/(\bar{f}_p - \underline{f}_p), & \text{otherwise,} \end{cases} \tag{4.3}$$

where $\epsilon > 0$ is some non-negative parameter. By the above definition, we will have $r_{p,s} = 0$ if $f_{p,s} = \underline{f}_p$, and $r_{p,s} = 1$ if $f_{p,s} = \bar{f}_p$ and $\bar{f}_p - \underline{f}_p > \epsilon$. The other value of $f_{p,s}$ is corresponding to some value of $r_{p,s}$ in $[0,\ 1]$. Given a factor $\tau \in [0,\ 1]$, for each solver s we define

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\,\{p \in \mathcal{P} :\ r_{p,s} \leq \tau\}, \tag{4.4}$$

where n_p is the total number of test problems. We are specially interested in the curve $\tau \in [0,\ 1] \to \rho_s(\tau)$.

Figure 4.4 plots the above curves for Algorithm 2, Algorithm 2*, BB, BB* on the 48 indefinite test problems with $\epsilon = 0.01$ and $\tau \in [0,\ 0.9]$. From Figure 4.4, we can see that Algorithm 2 is most effective in being able to find a solution of good quality. Therefore we still recommend Algorithm 2 for the indefinite SLBQP test problems.

The average performance of Algorithm 1 is still excellent. Taking Algorithm 2 as an example, the average number of iterations required by Algorithm 1 for each problem lies in the interval $[2.62,\ 8.44]$. For all the problems, the average number of projection iterations is 4.94. Therefore for indefinite test problems, Algorithm 1 again has the better average performance than the BS algorithm. Nevertheless, in 3 cases, Algorithm 1 takes more than 35 iterations. We have observed the worst case to occur when the linear piece crossing the λ axis is very short, and so is difficult to determine quickly. It might be interesting to design a hybrid projection algorithm of Algorithm 1 and the BS algorithm to avoid the worse case of Algorithm 1 and improve the average performance of the BS algorithm.

An analysis is also made for the dependence of the results on the parameters $ncond$, $neig$ and $na(\bar{\mathbf{x}}_1)$ by fixing one of the parameters and varing the remaining three parameters. The parameter $ncond$ still is essential to the performance of the Algorithm 2. As $ncond$ ranges from 4, 5, 6 to 7, the average CPU time (in seconds) of the corresponding problems ranges from 19.37, 29.01, 50.05 to 73.40. When $neig$ takes the values 1000, 2500, 5000, 9000, the average CPU times are 35.6825, 36.32, 40.76 and 56.34. This suggests that for Algorithm 2, the problem becomes more difficult as the number of negative eigenvalues increases. The analysis with $na(\bar{\mathbf{x}}_1)$ shows that it is better to start Algorithm 2 at a point with relatively many active constraints. The average CPU time for $na(\bar{\mathbf{x}}_1) = 1000$ is 52.85, where the CPU times for the other two cases are only 37.69 and 36.09.

SVM test problems [23]. Given a training set of labelled examples

$$D = \{(\mathbf{z}_i, w_i), i = 1, \ldots, n, \quad \mathbf{z}_i \in \mathbb{R}^m, w_i \in \{-1, 1\}\},$$

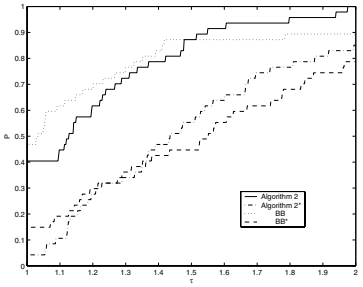

Fig. 4.3. Performance profiles for indefinite test problems

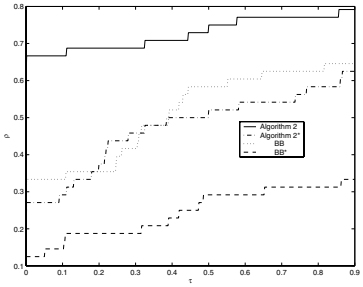

Fig. 4.4. Performance profiles based on the final function values

the SVM algorithm classifies new examples $\mathbf{z} \in \mathbb{R}^m$ by a decision function $F : \mathbb{R}^m \to \{-1, 1\}$ of the form

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

where $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is some kernel function and $\mathbf{x}^* = (x_i^*)$ solves

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\mathbf{x}^T G\mathbf{x} - \mathbf{e}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{0} \le \mathbf{x} \le C\mathbf{e} \\ & \mathbf{w}^T\mathbf{x} = 0, \end{aligned} \tag{4.5}$$

where G has entries $G_{ij} = w_i w_j K(\mathbf{z}_i, \mathbf{z}_j)$, $i, j = 1, 2, \ldots, n$, and C is a parameter of the SVM algorithm. The quantity $b^* \in \mathbb{R}$ is easily derived after $\mathbf{x}^*$ is computed. Our problems are generated by training SVMs on two real-world data sets: the MNIST database of handwritten digits and the UCI Adult data set. Test problems of size $n = 800$, 1600 and 3200 are constructed from MNIST by considering the first $n/2$ inputs of digit "8"and the first $n/2$ inputs of the other digits. From the UCI Adult, the versions with $n = 1605$, 2265 and 3185 are considered. The Gaussian kernel

$$K(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\|\mathbf{z}_i - \mathbf{z}_j\|_2^2/(2\sigma^2)\right) \tag{4.6}$$

is used in our tests, which ensures the semi-positive definiteness of the matrix G. The parameters C in (4.6) and σ in (4.6) are set to $(C, \sigma) = (10, 2000)$ for the MNIST database and $(1, \sqrt{10})$ for the UCI Adult data sets.

For these SVM test problems, a different stopping condition is used, which is based on the fulfilment of the KKT conditions within 10^{-3} (see [15]). The initial point is $\mathbf{x}_1 = \mathbf{0}$. The first stepsize is simply set to $\alpha_1 = \|\mathbf{g}_1\|_\infty$ since the projection $P_\Omega(\mathbf{x}_k - \mathbf{g}_k)$ is not used by the stopping condition. In addition, we terminate the projection algorithm if either $|r(\lambda)| \le 10^{-8}$ or $\Delta\lambda \le 10^{-8}$. The numerical results are shown in Table 4.2, where $\#iter$ the number of required iterations and $\#ls$ is the number of line search. k_{aver} and $k_{\max}$ denote the average number of iterations and the maximum number of iterations,

Table 4.2. Numerical results for SVM test problems

Problem	n	#iter	#ls	SV	BSV	k_{aver}	k_{max}	time	BB	
	800	128	8	281	1	4.18	12	0.5469	152	0.6563
MNIST	1600	198	20	457	9	4.87	13	2.7031	273	3.7656
	3200	508	82	807	25	5.28	16	27.8281	558	30.8906
	1605	106	4	691	584	4.49	7	1.9531	108	1.9844
UCI Adult	2265	141	9	1011	847	4.22	11	4.3594	130	4.0469
	3185	186	17	1303	1108	4.43	11	10.4531	167	9.4063

respectively, taken by Algorithm 1. SV and BSV stand for the number of support vectors and bound support vectors, respectively (a training example $\mathbf{z}_i$ is called a support vector if the corresponding x_i^* is nonzero and a bound support vector if $x_i^* = C$). The required CPU time (in seconds) is listed under the column "time". For BB, we list the iteration numbers and CPU times.

Table 4.3 also lists the numbers of iterations required by Algorithm 2, Algorithm 2*, BB, BB*, and the SPGM and GVPM algorithms (here we note that SPGM is first proposed in [2] and GVPM is proposed in [23]; the iteration numbers for these two algorithms are copied from [23]). For Algorithm 2* and BB*, we also show the required CPU time for each problem. From Figure 4.5, we see that both Algorithm 2 and Algorithm 2*

Table 4.3. Numerical comparisons for SVM test problems

Problem	n	SPGM	GVPM	Algorithm 2	Algorithm 2*		BB	BB*	
	800	163	161	128	141	0.6406	152	161	0.7969
MNIST	1600	303	277	198	218	3.2188	273	287	4.1250
	3200	691	513	508	428	24.7969	558	558	31.2969
	1605	90	153	106	96	1.8750	108	90	1.9063
UCI Adult	2265	135	196	141	142	4.5625	130	150	4.8438
	3185	175	282	186	155	9.1719	167	143	8.2813

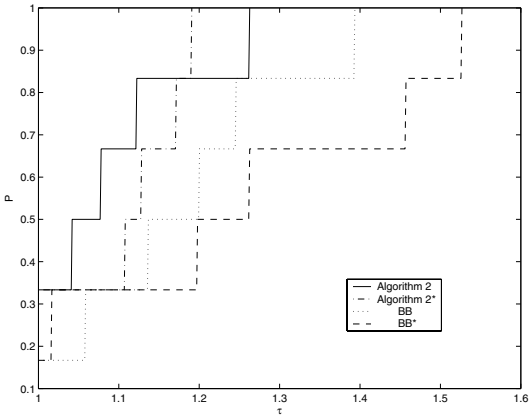

Fig. 4.5. Performance profiles for support vector machine problems

perform well on the SVM test problems. Again, from Table 1, we see that the average performance of the projection algorithm, Algorithm 1, is excellent. To achieve a very accurate solution, Algorithm 1 only requires around 4 or 5 iterations on average and in the worst case 16 iterations, although often a lot less.

To sum up, our numerical experiments in this section demonstrate the usefulness of (3.4) and suggest that Algorithm 2 is a strong contender for solving SLBQP problems. The average performance of Algorithm 1 is excellent and in most cases the heuristics for globalizing the secant iteration works well. In addition, to avoid the worse case of Algorithm 1 and improve the average performance of the BS algorithm, it might be worthwhile to consider a hybrid projection algorithm based on the two algorithms.

5. Conclusion and discussion

In this paper, we have proposed a new algorithm based on secant approximation, namely, Algorithm 1, for singly linearly constrained quadratic programs subject to lower and upper bounds (SLBQP) where the Hessian matrix is diagonal and positive definite. For general SLBQP problems, we have designed a gradient projection algorithm, namely, Algorithm 2, which uses Algorithm 1 to do projections, calculates the stepsize by a new formula (3.4) and incorporates a recently-established adaptive non-monotone line search. Our numerical experiments on large-scale random test problems and some medium-scale quadratic programs arising in training support vector machines demonstrate the usefulness of these algorithms.

Algorithm 1 allows the projections of the projected-gradient-like algorithm computed approximately. Nevertheless, the theory introduced in Birgin *et al* [3] might enable us to establish the global convergence of Algorithm 2 under suitable assumptions. The efficacy of the new stepsize (3.4) is still to be examined by further numerical experiments. Recently, Zanni [26] uses Algorithm 2 with $m = 2$ and $L = 2$ as the inner solver of the GPDT algorithm and obtains much better numerical results than the original GPDT algorithm using GVPM. In the unconstrained quadratic case, we have made some numerical experiments for (3.4) with different values of m, and the suggested value of m is between 3 and 5. We shall report these results elsewhere.

Another feature of Algorithm 2 is that, if the initial stepsize fails to meet the line search conditions, Algorithm 2 takes the exact one-dimensional minimizer instead of reducing the stepsize by some Armijo rule. This strategy ensures that at most 2 stepsizes are calculated at each iteration. To see whether this strategy is useful, we have compared algorithms SPGM and BB*. Note that both the algorithms define the reference function values by (4.2) with $M = 10$. The difference is in that, if the initial stepsize fails to meet the line search conditions, the former decides the second initial stepsize by some Armijo rule (see [2] for details), but the latter takes the exact one-dimensional minimizer. The numerical results in Table 4.3, suggest that an exact line search performs better in the context of QP problems.

In our work on the box constrained QP problem [7], we found that the best approach was to alternate the use of the two BB stepsize formulae given in [1]. We tried a similar approach for the SLBQP problems used in this paper, but found that the results were noticeably worse for the SVM problems. For these problems, the alternation method

failed to provide a solution in a reasonable number of iterations. At present we have no explanation for the discrepancy in these results. However, we note that [23] reports considerable success in using the two stepsize formulae in a more sophisticated way in the solution of SVM problems (see Table 4.3 for the numerical results of the method, GVPM). It is worth mentioning that GVPM is a monotone algorithm. We do not yet know whether there exists some efficient way to combine the alternation technique of the stepsize and the non-monotone line search.

Acknowledgements. The authors would very much like to thank Professor Gaetano Zanghirati at Università di Ferrara, and his colleagues, for the invaluable help that they provided during our numerical tests on the SVM problems. Many thanks are also due to the two anonymous referees, whose suggestions and comments improve the quality of this paper greatly.

References

1. Barzilai, J., Borwein, J.M.: Two-point step size gradient methods. IMA J. Numer. Anal. **8**, 141–148 (1988)
2. Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM J. Optim. **10**, 1196–1211 (2000)
3. Birgin, E.G., Martínez, J.M., Raydan, M.: Inexact spectral projected gradient methods on convex sets. IMA J. Numer. Anal. **23**, 539–559 (2003)
4. Brucker, P.: An O(n) algorithm for quadratic knapsack problems. Oper. Res. Lett. **3**, 163–166 (1984)
5. Calamai, P.H., Moré, J.J.: Quasi-Newton updates with bounds. SIAM J. Numer. Anal. **24**, 1434–1441 (1987)
6. Dai, Y.H., Fletcher, R.: On the asymptotic behaviour of some new gradient methods, Research report NA/212, Department of Mathematics, University of Dundee, 2003, and in Mathematical Programming, Series A, Vol. 103, No. 3, 541–560 (2005)
7. Dai, Y.H., Fletcher, R.: Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming, Research report NA/215, Department of Mathematics, University of Dundee, 2003, and in Numerische Mathematik A, Vol. 100, No. 1, 21–47 (2005)
8. Dai, Y.H., Zhang, H.C.: An adaptive two-point stepsize gradient algorithm. Numer. Algorithms **27**, 377–385 (2001)
9. Dolan, E.D., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. Ser. A **91**, 201–203 (2002)
10. Fletcher, R.: On the Barzilai-Borwein method, Research report, Department of Mathematics, University of Dundee, 2001. and in Optimization and Control with Applications, Eds. L. Qi, K. Teo and X. Yang, Kluwer Academic Publishers, Series in Applied Optimization, (to appear).
11. Friedlander, A., Martínez, J.M., Molina, B., Raydan, M.: Gradient method with retards and generalizations. SIAM J. Numer. Anal. **36**, 275–289 (1999)
12. Grippo, L., Lampariello, F., Lucidi, S.: A nonmonotone line search technique for Newton's method. SIAM J. Numer. Anal. **23**, 707–716 (1986)
13. Held, M., Wolfe, P., Crowder, H.: Validation of subgradient algorithms. Math. Prog. **6**, 62–88 (1974)
14. Helgason, R., Kennington, J., Lall, H.: A polynomially bound algorithms for a singly constrained quadratic program. Math. Prog. **18**, 338–343 (1980)
15. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C.J.C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning , MIT Press, Cambridge, Massachussets, 1998
16. Meyer, R.R.: Multipoint methods for separable nonlinear networks. Math. Programming Study **22**, 185–205 (1984)
17. Moré, J., Toraldo, G.: Algorithms for bound constrained quadratic programming problems. Numer. Math. **55**, 377–400 (1989)
18. Pardalos, P.M., Kovoor, N.: An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. Math. Prog. **46**, 321–328 (1990)
19. Pardalos, P.M., Rosen, J.B.: Constrained global optimization: Algorithms and applications. In: Lecture Notes in Computer Science, Vol. 268 (Springer, Berlin, 1987)
20. Platt, J.C.: Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel Methods - Support Vector Learning, Schölkopf, B., Burges, C., Smola, A. (eds.) MIT Press, Cambridge, Massachusets, 1998

21. Raydan, M.: On the Barzilai and Borwein choice of steplength for the gradient method. IMA J. Numer. Anal. **13**, 321–326 (1993)
22. Raydan, M.: The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. SIAM J. Optim. **7**, 26–33 (1997)
23. Serafini, T., Zanghirati, G., Zanni, L.: Gradient projection methods for quadratic programs and applications in training support vector machines. Research report, 2003 Optim. Meth. Software, Vol. 20, 347–372 (2005)
24. Toint, Ph.L.: A non-monotone trust region algorithm for nonlinear optimization subject to convex constraints. Math. Prog. **77**, 69–94 (1997)
25. Vapnik, V.: Estimation of dependences based on empirical data. Springer-Verlag, 1982
26. Zanni, L.: An improved gradient projection-based decomposition technique for support vector machines. Research report, Dipartimento di Mathematica, Università di Modena e Reggio Emilia, Italy