DIPARTIMENTO DI MATEMATICA
PURA ED APPLICATA "G. VITALI"

On the Working Set Selection in Gradient Projection-based Decomposition Techniques for Support Vector Machines

T. Serafini, L. Zanni

Revised on July 2004

Preprint nr. 57 (January 2004)

Università degli Studi di Modena e Reggio Emilia

ON THE WORKING SET SELECTION IN GRADIENT PROJECTION-BASED DECOMPOSITION TECHNIQUES FOR SUPPORT VECTOR MACHINES*

THOMAS SERAFINI† and LUCA ZANNI‡

January 2004. Revised July 2004.

Dipartimento di Matematica, Università di Modena e Reggio Emilia, via Campi 213/b, 41100 Modena, Italy

This work deals with special decomposition techniques for the large quadratic program arising in training Support Vector Machines. These approaches split the problem into a sequence of quadratic programming subproblems which can be solved by efficient gradient projection methods recently proposed. By decomposing into much larger subproblems than standard decomposition packages, these techniques show promising performance and are well suited for parallelization. Here, we discuss a crucial aspect for their effectiveness: the selection of the working set, that is the index set of the variables to be optimized at each step through the quadratic programming subproblem. We analyse the most popular working set selections and develop a new selection strategy that improves the convergence rate of the decomposition schemes based on large sized working sets. The effectiveness of the proposed strategy within the gradient projection-based decomposition techniques is shown by numerical experiments on large benchmark problems, both in serial and parallel environments.

Keywords: support vector machines, quadratic programs, decomposition techniques, gradient projection methods, large-scale problems.

AMS Classification: 65K05, 90C20, 68T05

1 Introduction

We consider the numerical solution through decomposition techniques of the convex quadratic programming (QP) problem arising in training the learning methodology named *Support Vector Machines* (SVMs) [4, 25]. The main difficulties in solving this problem are the density and the large size of the matrix of the objective function. In fact, the SVM quadratic program has the form

$$\begin{array}{ll} \min & \mathcal{F}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} - \sum_{i=1}^{n} x_i \\ \text{sub. to} & \sum_{i=1}^{n} y_i x_i = 0, \\ & 0 \le x_i \le C, \qquad i = 1,\dots,n, \end{array} \tag{1}$$

where the size n is the number of labelled examples of the given training set

$$D = \{(\boldsymbol{z}_i, y_i),\ i = 1,\dots,n, \quad \boldsymbol{z}_i \in \mathbb{R}^m,\ y_i \in \{-1, 1\}\},$$

and the entries of G are defined by

$$G_{ij} = y_i y_j K(\boldsymbol{z}_i, \boldsymbol{z}_j), \quad i, j = 1, 2, \dots, n,$$

*This work was supported by the Italian Education, University and Research Ministry (grants FIRB2001/RBAU01JYPN and FIRB2001/RBAU01877P).

†E-mail: serafini.thomas@unimo.it

‡E-mail: zanni.luca@unimo.it

in which $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a special kernel function. Examples of widely-used kernel functions are the linear kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{z}_i^T \boldsymbol{z}_j$), the polynomial kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = (1 + \boldsymbol{z}_i^T \boldsymbol{z}_j)^d,\ d \in \mathbb{N}$) and the Gaussian kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = \exp(-\|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2/(2\sigma^2)),\ \sigma \in \mathbb{R}$). Since these kernel functions make the matrix G dense and in many real life applications the size of the training set is very large ($n \gg 10^4$), problem (1) cannot be generally solved by traditional approaches based on explicit storage of G. The main strategy proposed to overcome this drawback consists in splitting the problem into a sequence of smaller QP subproblems that can be stored in the available memory and efficiently solved. At each step of the iterative procedure, a subset of the variables, sized as the subproblems and identified by a set of indices named *working set*, is optimized through the solution of the subproblem, in order to produce a progress towards the minimum of $\mathcal{F}(\boldsymbol{x})$. Based on this simple idea, many effective decomposition techniques have been developed in the last years. In spite of the same basic idea, these techniques present remarkable differences, the most important of which can be considered the size chosen for the working set, i.e., the size of the subproblems. In fact, the decomposition schemes designed for working sets sized 2 can solve the subproblems analytically [2, 6, 10, 21], while the techniques that use larger working sets need an inner numerical QP solver [3, 8, 9, 18, 27]. Both the frameworks have been explored and have given rise to decomposition packages widely-used within the SVM community. However, from both theoretical and computational viewpoint, many issues about these approaches are currently investigated, for better understanding their behavior and for improving their performance.

In this paper, we present a computational study on the working set selection for special decomposition techniques based on large-sized working sets. We analyse some practical versions of the well known selection strategy proposed by Joachims in [9]. This analysis emphasizes some crucial issues that we have improved in a new working set selection. A wide numerical experimentation shows that the proposed strategy yields a better convergence rate in decomposition schemes based on large working sets. Thus, by using the new selection strategy, we can significantly improve the performance of the decomposition technique introduced in [27], that is appropriately designed for decomposing into large subproblems. The main feature of this technique consists in using a very efficient gradient projection method for solving the inner QP subproblems [24, 26], in such a way that subproblems with size up to $\mathrm{O}(10^3)$ can be managed without excessive overhead. This feature, combined with the new working set selection, allows to fully exploit the benefits, in terms of convergence rate, arising from the possibility to optimize many variables at each iteration. The promising performance of the proposed gradient projection-based decomposition technique is evaluated by a comparison with the Joachims' SVM^{light} software [9] on large-scale benchmark problems. Furthermore, as observed in [27], the possibility to decompose into large subproblems makes the scheme well suited for an easy and effective parallelization; the behavior of a parallel version of the decomposition technique is shown by solving the same benchmark problems on a multiprocessor system.

The paper is organized as follows: section 2 states a decomposition framework including the SVM^{light} and the other schemes discussed in the work, section 3 analyses several existing working set selections and presents the new selection strategy, section 4 shows the behavior of a gradient projection-based decomposition technique that uses the proposed working set selection, both in serial and parallel environments and, finally, section 5 draws some conclusions.

2 The decomposition technique

We present a general iterative decomposition scheme for problem (1) that includes the techniques we are going to discuss about. We start by introducing some essential notations.

At each iteration of the decomposition scheme, the indices of the variables $\boldsymbol{x} = \left[x_1, x_2, \ldots, x_n\right]^T$ are split into the set $\mathcal{B}$ of *basic* variables, referred as the *working set*, and the set $\mathcal{N} = \left\{1, 2, \ldots, n\right\} \setminus \mathcal{B}$ of *nonbasic* variables. Consequently, arrays $\boldsymbol{x}$ and G are arranged with respect to $\mathcal{B}$ and $\mathcal{N}$ as follows:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{bmatrix}, \qquad G = \begin{bmatrix} G_{\mathcal{B}\mathcal{B}} & G_{\mathcal{B}\mathcal{N}} \\ G_{\mathcal{N}\mathcal{B}} & G_{\mathcal{N}\mathcal{N}} \end{bmatrix}.$$

If $\boldsymbol{x}^*$ denotes a solution of (1) and $\boldsymbol{x}^{(k)}$ is the approximation of $\boldsymbol{x}^*$ at the beginning of the k-th iteration, the idea behind the decomposition technique consists in progressing towards the minimum of $\mathcal{F}(\boldsymbol{x})$ by substituting $\boldsymbol{x}_{\mathcal{B}}^{(k)}$ with the vector $\boldsymbol{x}_{\mathcal{B}}^{(k+1)}$ obtained by solving (1) with respect to the variables in the working set only. This leads to a sequence of QP subproblems with size n_{sp} equal to the working set size: $n_{sp} = \#\mathcal{B}$. Of course, in order to achieve a rapid decrease in the objective function, an appropriate choice of the working set is required. Since among existing decomposition approaches [2, 3, 8, 9, 18, 21, 27] the best results are obtained with working set selections similar to the strategy implemented in SVM^{light}, we consider selection rules that follow the Joachims' idea. The main steps of our decomposition scheme are stated in Algorithm DT and discussed in the following of this section.

First of all we examine how to solve the QP subproblem in step 2. In some popular techniques [2, 6, 10, 21], the choice $n_{sp} = 2$ enables an analytical solution of (2) while in the schemes that work with $n_{sp} > 2$ a numerical QP solver is required [8, 9, 18, 27]. The former techniques try to keep the minimal computational cost per iteration while the latter exhibit more expensive iterations, but can achieve a much better convergence rate. For these last approaches, the choice of the inner QP solver is crucial. Recently, very efficient gradient projection methods for problem (2) are developed [5, 24, 26]. They are able to fully exploit the special structure of the constraints, by performing the projection onto the feasible region with nonexpensive algorithms [5, 20], and to exhibit good convergence rate due to appropriate linesearch strategies and/or new selection rules for the steplength parameter. A first decomposition approach based on such kind of QP solvers is proposed in [27]; in this case, the gradient projection method used as inner solver is a special *Variable Projection Method* (VPM) developed in [26] (see also [7, 22, 23]). By using the VPM as inner solver, the decomposition techniques in [27] can split the problem into sufficiently large subproblems ($\mathrm{O}(10^3)$) without making the cost for the inner optimization task excessively dominant. This feature gives rise to new promising decomposition frameworks that are not previously explored. In fact, other popular decomposition techniques based on standard QP packages are generally used with very small subproblems and their behavior for increasing n_{sp} is not deeply investigated, since the overhead due to the inner solver makes them completely not appealing in these situations. In [24] an improved VPM version, named *Generalized VPM* (GVPM), has been compared with the solvers MINOS and pr_LOQO used in the decomposition technique [18] and in the package SVM^{light} respectively. On test problems with size up to 3200, the GVPM has shown much better performance than MINOS and pr_LOQO, and when used within the decomposition technique [27] with sufficiently large n_{sp} has implied significant improvements with respect to the best SVM^{light} results. In the following, the decomposition technique [27] equipped with the GVPM as inner solver will be referred as *Gradient Projection-based Decomposition Technique* (GPDT). During the completion of this work, another gradient projection method, well suited for the subproblems (2), is proposed by Dai and Fletcher in [5]. By exploiting new ideas for the steplength selection and the linesearch strategy, it exhibits better convergence rate than the GVPM. For this reason, it will be interesting to explore the benefits that the new solver can introduce in decomposition techniques like GPDT. We think to deal with this topic in a future work.

ALGORITHM DT (SVM Decomposition Technique)

STEP 1. *Initialization.* Let $\boldsymbol{x}^{(1)}$ be a feasible point for (1), let n_{sp} and n_c be two integers such that $n \geq n_{sp} \geq n_c > 0$, n_c even; set $k \leftarrow 1$. Arbitrarily choose n_{sp} indices for the working set $\mathcal{B}$.

STEP 2. *QP subproblem solution.* Compute the solution $\boldsymbol{x}_{\mathcal{B}}^{(k+1)}$ of

$$\begin{array}{rl} \min & \frac{1}{2}\boldsymbol{x}_{\mathcal{B}}^T G_{\mathcal{B}\mathcal{B}}\boldsymbol{x}_{\mathcal{B}} + \left(G_{\mathcal{B}\mathcal{N}}\boldsymbol{x}_{\mathcal{N}}^{(k)} - (1,1,\dots,1)^T\right)^T \boldsymbol{x}_{\mathcal{B}} \\ \text{sub. to} & \sum_{i\in\mathcal{B}} y_i x_i = -\sum_{i\in\mathcal{N}} y_i x_i^{(k)}, \\ & 0 \leq x_i \leq C \quad \forall i \in \mathcal{B}, \end{array} \tag{2}$$

and set $\boldsymbol{x}^{(k+1)} = \left(\boldsymbol{x}_{\mathcal{B}}^{(k+1)^T}, \ \boldsymbol{x}_{\mathcal{N}}^{(k)^T} \right)^T$.

STEP 3. *Gradient updating.* Update the gradient

$$\nabla\mathcal{F}(\boldsymbol{x}^{(k+1)}) = \nabla\mathcal{F}(\boldsymbol{x}^{(k)}) + \left[\begin{array}{c} G_{\mathcal{B}\mathcal{B}} \\ G_{\mathcal{N}\mathcal{B}} \end{array}\right] \left(\boldsymbol{x}_{\mathcal{B}}^{(k+1)} - \boldsymbol{x}_{\mathcal{B}}^{(k)}\right) \tag{3}$$

and terminate if $\boldsymbol{x}^{(k+1)}$ satisfies the KKT conditions.

STEP 4. *Updating of $\mathcal{B}$.*

STEP 4.1. Find the indices corresponding to the nonzero components of the solution of

$$\begin{array}{rl} \min & V(\boldsymbol{d}) = \nabla\mathcal{F}(\boldsymbol{x}^{(k+1)})^T \boldsymbol{d} \\ \text{sub. to} & \boldsymbol{y}^T \boldsymbol{d} = 0, \\ & d_i \geq 0 \quad \text{for } i \text{ such that } x_i^{(k+1)} = 0, \\ & d_i \leq 0 \quad \text{for } i \text{ such that } x_i^{(k+1)} = C, \\ & -1 \leq d_i \leq 1, \\ & \#\{d_i \mid d_i \neq 0\} \leq n_c. \end{array} \tag{4}$$

STEP 4.2. Update $\mathcal{B}$ by first including the indices found in step 4.1 and then by filling the set up to n_{sp} entries with indices from the previous working set; set $k \leftarrow k+1$ and go to step 2.

The third step of Algorithm DT concerns the computation of $\nabla\mathcal{F}(\boldsymbol{x}^{(k+1)}) = G\boldsymbol{x}^{(k+1)} - \mathbf{1}$. The updating formula (3) gives $\nabla\mathcal{F}(\boldsymbol{x}^{(k+1)})$ by involving only the columns of G corresponding to the indices of $\mathcal{B}$ for which

$(x_i^{(k+1)} - x_i^{(k)}) \neq 0,\ i \in \mathcal{B}$. Since for large problems the matrix G can not fit in the memory, this formula is useful for reducing the computations of G entries, i.e., for avoiding kernel evaluations that in many applications can be very expensive. To this end, another crucial trick consists in using an area of the available memory as a caching area, where some of the most recently used columns of G can be stored in order to avoid their recomputation in the next iterations. Caching strategies are commonly adopted in many decomposition techniques and will be used by SVM^{light} and GPDT in all the subsequent experiments, together sparse representation of the training examples, useful for optimizing kernel evaluations and for reducing memory consumption. When $\nabla\mathcal{F}(\boldsymbol{x}^{(k+1)})$ is obtained, the same stopping criterion proposed in [9, 21] is checked; it is based on the fulfilment of the Karush-Kuhn-Tucker (KKT) conditions within a prefixed tolerance (the equality constraint multiplier is computed as in [9]). Of course other stopping rules could be used but, since we are interested in a comparison with SVM^{light}, we prefer to adopt the stopping criterion of this package.

Finally, step 4 needs to be discussed. The working set selection described in this step is very general and includes as special cases the strategies implemented in the main decomposition schemes [2, 8, 9, 10, 27]. In fact, at most $n_c \leq n_{sp}$ entries of the new working set are selected by following the well known strategy first proposed in [9], that consists in choosing the indices of nonzero components of the solution of (4). Precisely, in [9] the constraint $\#\{d_i \mid d_i \neq 0\} = n_c = n_{sp}$ instead of $\#\{d_i \mid d_i \neq 0\} \leq n_c$ was proposed. We use the inequality constraint since, as pointed out in [1, 13], in theory n_c nonzero elements may not be always available. Furthermore, we allow to select by this strategy at most $n_c \leq n_{sp}$ indices, as actually done by SVM^{light} and as recommended in [8, 27], in order to exploit the benefits arising from including in $\mathcal{B}$ some indices of the previous working set. We solve the linear problem (4) by the nonexpensive algorithm described in [13] and will focus on some ideas for selecting indices from the previous working set in the next section.

Based on special versions of this general working set selection, convergence proves for Algorithm DT can be obtained. The asymptotic convergence of Algorithm DT with $n_c = n_{sp}$ is first proved in [13] under a suitable strict block-wise convexity assumption. If G is assumed positive definite (this is the case of Gaussian kernels and all $\boldsymbol{z}_i \neq \boldsymbol{z}_j$) the linear convergence is ensured in [14]. Convergence results without the above assumptions on G are available in [11, 15] for the case $n_c = n_{sp} = 2$ only. How to extend these results to the general case of $n_{sp} \geq 2$ is a current interesting research issue and only recently some suggestions are appeared. In [19], the asymptotic convergence for the case $n_c = n_{sp} \geq 2$, without assumptions on G, is proved by a proximal point modification of the subproblem (2). However, computational results about this approach are not available and, in particular, the way in which this modification affects in practice the convergence rate is not still explored. We don't deal with this new scheme in the present paper and all the subsequent considerations on working set selections will concern decomposition schemes based on the standard subproblem formulation given in Algorithm DT.

3 On the working set updating

In this section we analyse the working set selection of Algorithm DT, with special attention to the cases where n_{sp} is assumed large. It is well known that a careful choice of $\mathcal{B}$ is crucial for reducing the number of iterations in the above decomposition schemes. Here, starting from an analysis of the classical working set selection used in SVM^{light} and the suggestions in [8, 27], we develop a new improved selection strategy and evaluate its effectiveness on large-scale benchmark problems. All the subsequent computational results are obtained by training Gaussian SVMs on the well known UCI Adult, Web and MNIST data sets [9, 21]; we describe in detail the test problems and the IBM SP4 testing platform in the Appendix.

As already observed, the original selection strategy described in [9] falls within Algorithm DT by assuming $n_c = n_{sp}$. Nevertheless, in the technical documentation of the package, the number n_c of new variables entering the working set in each iteration is recommended to be less than n_{sp} in order to avoid zigzagging. When $n_c < n_{sp}$, the n_c indices entering the working set substitute the oldest elements of $\mathcal{B}$, and in this way a dramatic reduction of the SVM^{light} iterations can be obtained.

We give evidence of this behavior in Table I, where, for a given n_{sp}, we show the number of decomposition iterations (it) and the elapsed time in seconds (time) required by SVM^{light} with $n_c = \frac{n_{sp}}{2}$ and $n_c = n_{sp}$. For each test problem, we denote by $\mathrm{N_{SV}}$ the number of *Support Vectors* (SVs), i.e., the training examples corresponding to the nonzero components of $\boldsymbol{x}^*$ ($x_i^* > 0$), and by $\mathrm{N_{BSV}}$ the number of *Bound Support Vectors* (BSVs), i.e., the training examples corresponding to the components of $\boldsymbol{x}^*$ at the upper bound ($x_i^* = C$). SVM^{light} (version 5.0) is run with pr_LOQO as inner QP solver, a caching area of 512 MB and default setting for other parameters (this means that the tolerance for the stopping rule is 0.001 and the shrinking is enabled). The results in Table I emphasize the importance of using $n_c < n_{sp}$ for reducing the iterations when the number of free variables (unbound support vectors) is much larger than n_{sp}. As observed also in [8], these situations are the most difficult for the decomposition method since the values of many variables need to be decided by processing only a small part of them at each step. On the other hand, when n_{sp} is larger than the number of free variables, a very good convergence of the decomposition scheme is also observed with $n_c = n_{sp}$. However, in practical large-scale applications this last assumption is unrealistic and generally it is advisable to use SVM^{light} with $n_c < n_{sp}$. For what concern the computational time observed in these experiments, we remark that SVM^{light} is not designed to work efficiently with large subproblems and the poor performance observed when n_{sp} increases is not surprising at all. In particular, due to the huge computational time expected, we don't run SVM^{light} on the largest data sets with the largest values of n_{sp}. We evaluate also the behavior of SVM^{light} for many other small values of n_{sp} and n_c but, for each test problem, the values reported in Table I have implied the best computational time.

The use of $n_c < n_{sp}$ and consequently the possibility to keep indices of the previous working set into the new one is also investigated in [8] and [27]. In these papers, similar strategies for choosing indices from the previous working set are presented.

In [8], by studying decomposition methods for the simpler bound-constrained QP problem arising in the nonstandard SVM formulation proposed by [16], a working set selection is suggested in which some indices of free variables in the previous iteration are kept in the new working set, together with the indices coming from a step like step 4.1. The aim of this strategy is to keep the number of free variables as small as possible during the iterative procedure, in order to simplify the decision of their values. In fact, a larger number of free variables causes more difficulty to the decomposition scheme that is not able to consider all these variables together in each iteration. Since, in this way, the convergence rate of the decomposition procedure is markedly improved, the proposed selection strategy is also tested within the decomposition method for the standard SVM formulation (1). Some numerical experiments on small test problems indicate that the selection strategy performs well also in this framework and it is an useful tool for reducing the number of iterations required by SVM^{light} in difficult cases. The experiments in [8] are carried out by running SVM^{light} with $n_{sp} = n_c = 10$ and the proposed working set selection with $n_{sp} = 10$, $n_c = 4$.

TABLE I Number of SVM^{light} iterations for different n_{sp} and n_c.

n_{sp}	n_c	it	time	it	time
UCI Adult data set					
		$n = 32561$		$n = 3185$	
		$N_{SV} = 11698$		$N_{SV} = 1299$	
		$N_{BSV} = 10605$		$N_{BSV} = 1112$	
20	10	4407	278.7	337	3.0
20	20	8546	363.8	418	3.1
600	300	125	505.2	11	22.6
600	600	2707	6315.7	8	19.5
1300	650	48	1703.4	5	127.7
1300	1300	320	7552.0	4	119.1
Web data set					
		$n = 49749$		$n = 9888$	
		$N_{SV} = 3199$		$N_{SV} = 1111$	
		$N_{BSV} = 919$		$N_{BSV} = 179$	
8	4	7235	184.6	2276	11.4
8	8	10174	232.4	2882	12.7
600	300	137	580.2	33	84.4
600	600	4057	11044.8	50	129.5
1500	750	40	1932.4	9	293.8
1500	1500			7	287.0
MNIST data set					
		$n = 60000$			
		$N_{SV} = 3159$			
		$N_{BSV} = 160$			
8	4	9226	901.6		
8	8	10869	1247.8		
200	100	320	1723.1		
200	200	3252	4436.1		

In [27], a decomposition scheme for (1) that uses large values of n_{sp} is developed. The working set selection designed for this case, described in Algorithm WSS1, exploits the above ideas and another trick. In fact, besides the selection of $n_c < n_{sp}$ indices as in SVM^{light} and the choice of indices of previous free variables as in [8], this selection also manages a situation typical when n_{sp} is large: how to fill the current working set when the previous free variables are not enough. For this situation, in [27] it is suggested to keep into the new working set first the indices of null variables and then, if these are not enough, the indices of variables at the upper bound C. Of course, this distinction is effective only for noisy problems, i.e., problems with a sufficiently large number of BSVs. By trying to leave out the variables at the upper bound from the working set, the aim of the proposed criterion is to quickly increase the number of these variables and consequently, since many of them will be often BSVs at the end of the optimization process, to produce a faster approximation of the final set of SVs. On the other hand, when the problems have few BSVs the implementation of this trick doesn't produce significant differences respect to other criteria for selecting null or upper bound variables (after the free variables) in step 4.2 of Algorithm WSS1.

```
ALGORITHM WSS1 (Working set selection used in [27])
STEP 4. Updating of $\mathcal{B}$.
    STEP 4.1. Find the indices corresponding to the nonzero components of the solution of (4).
              Let $\bar{\mathcal{B}}$ be the set of these indices.
    STEP 4.2. Fill $\bar{\mathcal{B}}$ up to $n_{sp}$ entries by adding the indices $j \in \mathcal{B}$ satisfying $0 < x_j^{(k+1)} < C$; if these
              indices are not enough, then add those such that $j \in \mathcal{B}$, $x_j^{(k+1)} = 0$ and, eventually,
              those satisfying $j \in \mathcal{B}$, $x_j^{(k+1)} = C$; set $\mathcal{B} = \bar{\mathcal{B}}$, $k \leftarrow k+1$ and go to step 2.
```

The above considerations are confirmed by the numerical results reported in Table II and Figure 1. In these experiments we test two versions of the GPDT proposed in [27]: the original version with the working set selection described in Algorithm WSS1 and the version that uses a modified working set selection in which, after the indices of free variables, the indices $j \in \mathcal{B}$ such that $x_j^{(k)} = C$ are selected before the indices $j \in \mathcal{B}$ satisfying $x_j^{(k)} = 0$. We denote by GPDT$(C, 0)$ the modified GPDT version. In order to standardize the comparison with SVM^{light}, all the subsequent GPDT codes are executed with the same stopping tolerance and caching area size previously used.

TABLE II Number of iterations for GPDT and GPDT$(C, 0)$

			GPDT		GPDT$(C,0)$	
Data set	n_{sp}	n_c	it	time	it	time
UCI Adult ($n = 32561$)	1300	300	56	235.1	95	265.2
	1300	650	45	285.1	65	297.2
	1300	1000	49	272.7	60	327.4
Web ($n = 49749$)	1500	500	50	257.1	58	322.8
	1500	750	49	313.5	47	300.6
	1500	1000	64	432.6	91	574.8
MNIST ($n = 60000$)	3000	1000	8	681.4	8	739.1
	3000	1500	8	963.8	8	1035.8
	3000	2000	8	1047.4	8	1184.0

In Table II the number of iterations required by the two GPDT versions are compared on the three large-scale test problems already used in the experiments with SVM^{light}. The GPDT exhibits a significant iterations reduction on the UCI Adult test problem, where $\mathrm{N_{BSV}} \approx 0.9\mathrm{N_{SV}}$, while less evident benefits can be observed on the Web data set, where $\mathrm{N_{BSV}} \approx 0.3\mathrm{N_{SV}}$, and the same iterations are required on the MNIST test problem where $\mathrm{N_{BSV}}$

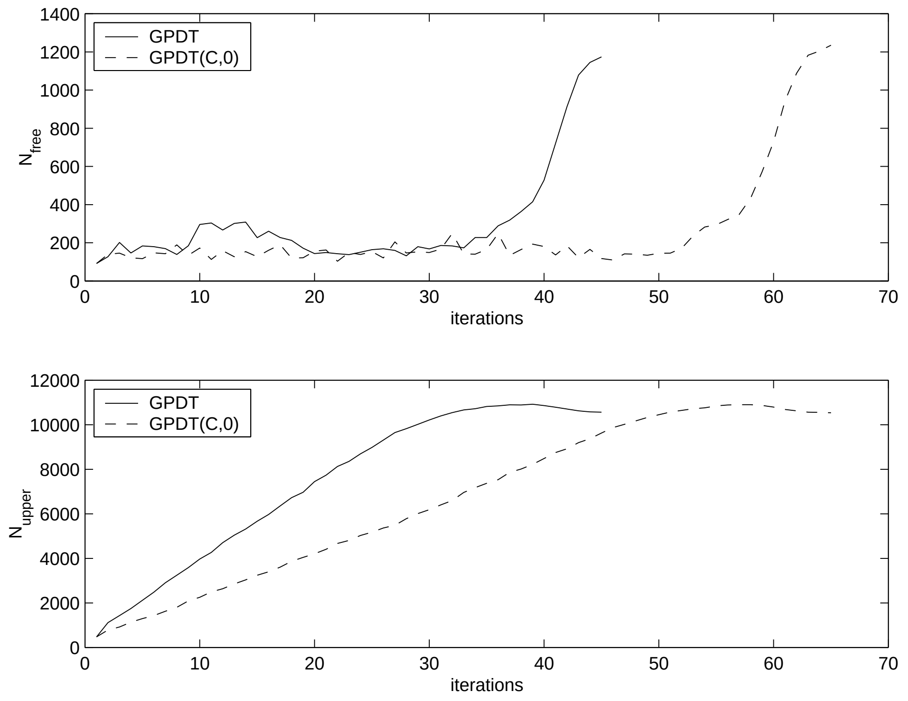

FIGURE 1 Free variables and variables at upper bound for the UCI Adult test problem

$\approx 0.05\mathrm{N}_{\mathrm{SV}}$. For the run with $n_c = 650$ on the UCI Adult test problem, in Figure 1 the number of free variables ($\mathrm{N}_{\mathrm{free}}$) and the number of variables at the upper bound ($\mathrm{N}_{\mathrm{upper}}$) in each iteration are reported. The main difference is that GPDT shows a faster approximation of the variables at the upper bound, emphasizing in this way the effectiveness of its working set selection in case of noisy problems. For what concern the computational time of the two GPDT versions, we remark that different strategies for the working set selection not only affects the number of decomposition iterations but can imply significant differences both in the data of the subproblem (2) and in the sparsity of $\boldsymbol{x}_{\mathcal{B}}^{(k+1)}$. This explains the different cost per iteration sometimes exhibited by GPDT and GPDT(C,0) (see for example the results for the MNIST test problems).

Taking into account all the above considerations, in the following we introduce two improvements to the working set selection WSS1 used by GPDT.

First of all, we improve step 4.2 of Algorithm WSS1 by modifying the criterion for selecting indices from the previous working set $\mathcal{B}$. We require that, in choosing the indices as specified in step 4.2, the most recent indices entering $\mathcal{B}$ have priority over the oldest (see step 4.2 in Algorithm WSS2 for a detailed description). In this way, we give more confidence to the variables that have been optimized for a larger number of iterations and could have already achieved their final optimal values. The effectiveness of this priority criterion, used also in SVM^{light}, is clearly shown by the results in Table III where the same test problems of Table II are solved with a version of GPDT, named GPDT(m.r.), including the new criterion. By comparing the number of iterations required by GPDT and GPDT(m.r.), we observe that the new criterion implies remarkable improvements for the Web data set and similar results on the other two test problems.

The second improvement concerns the introduction of an adaptive reduction of the parameter n_c. We recall

Table III Number of iterations for GPDT(m.r.) and GPDT2

			GPDT(m.r.)		GPDT2	
Data set	n_{sp}	n_c	it	time	it	time
UCI Adult ($n = 32561$)	1300	300	54	232.9	54	237.2
	1300	650	32	217.7	32	215.9
	1300	1000	47	268.5	47	271.5
Web ($n = 49749$)	1500	500	30	191.6	27	187.7
	1500	750	34	258.2	25	168.1
	1500	1000	48	353.1	28	177.3
MNIST ($n = 60000$)	3000	1000	8	685.6	8	680.5
	3000	1500	8	972.4	8	984.7
	3000	2000	8	1035.2	8	1102.3

that, at each iteration, a new working set is built with at most n_c indices selected in step 4.1 and with other indices from the previous working set; this means that the number n_{new} of new indices entering the working set is less than or equal to n_c. The adaptive reduction criterion we propose consists in setting n_c equal to n_{new}, if n_{new} is not smaller than an empirical threshold given by $\max\{10, L\}$, where L is the largest even integer such that $L \leq \frac{n_{sp}}{10}$; otherwise, the threshold value is used as new n_c. The description of this reduction strategy is given in step 4.3 of Algorithm WSS2.

```
ALGORITHM WSS2 (New working set selection)
STEP 4. Updating of $\mathcal{B}$.
    STEP 4.1. Find the indices corresponding to the nonzero components of the solution of (4).
              Let $\bar{\mathcal{B}}$ be the set of these indices.
    STEP 4.2. Fill $\bar{\mathcal{B}}$ up to $n_{sp}$ entries by adding the most recent indices $j \in \mathcal{B}$ † satisfying
              $0 < x_j^{(k+1)} < C$; if these indices are not enough, then add the most recent indices $j \in \mathcal{B}$
              such that $x_j^{(k+1)} = 0$ and, eventually, the most recent indices $j \in \mathcal{B}$ satisfying $x_j^{(k+1)} = C$.
    STEP 4.3. Set $n_c = \min\{n_c, \max\{10, L, n_{new}\}\}$, where $L$ is the largest even number such that
              $L \leq \frac{n_{sp}}{10}$ and $n_{new}$ is the largest even number such that $n_{new} \leq \#\{j,\ j \in \bar{\mathcal{B}},\ j \notin \mathcal{B}\}$;
              set $\mathcal{B} = \bar{\mathcal{B}}$, $k \leftarrow k + 1$ and go to step 2.
```

†We mean the indices that are in the working set $\mathcal{B}$ since the lowest number of consecutive iterations.

The aim of this trick is to reduce the zigzagging due to a large n_c used in too many iterations. As already observed in the discussion on the SVM^{light}, this phenomenon can be reduced by using $n_c < n_{sp}$, but when n_{sp} is large, a too small n_c can vanish the benefits arising from the ability to optimize many new variables in a single decomposition step. The proposed strategy allows to start the decomposition with a sufficiently large n_c and tries to avoid excessive zigzagging by subsequently reducing this large initial value. Finally, when n_c reaches a very small value or becomes too small in comparison with n_{sp}, the reduction is not generally fruitful and an empirical threshold value will be assumed for n_c from now on. Note that the threshold value is also used to check if the initial n_c is so small as to make its reduction unnecessary.

We call GPDT2 the improved GPDT version that uses the working set selection described in Algorithm WSS2 instead of Algorithm WSS1. The advantages due to the introduction of step 4.3 in the working set selection

Table IV Number of GPDT2 iterations for different n_{sp} and n_c.

n_{sp}	n_c	it	time	it	time
UCI Adult data set					
		$n = 32561$		$n = 3185$	
20	10	3689	203.1	258	3.1
600	300	77	198.7	8	2.2
1300	650	32	215.9	5	3.0
Web data set					
		$n = 49749$		$n = 9888$	
8	4	6848	221.3	2255	16.4
600	300	190	207.6	30	15.1
1500	750	25	168.1	8	24.7
MNIST data set					
		$n = 60000$			
8	4	8859	1413.5		
200	100	304	1506.1		
1000	200	55	778.7		
2000	300	22	662.8		
3700	1000	8	704.1		

can be evaluated by comparing the results concerning GPDT(m.r.) and GPDT2 in Table III. The adaptive reduction of n_c allows a significant reduction of the iterations for the Web test problems without decreasing the performance on the other data sets. Furthermore, a comparison between the results obtained with GPDT2 and GPDT highlights the effectiveness of the new working set selection with respect to the selection strategy used in [27].

4 GPDT2 performance

This section presents computational results obtained by the GPDT2 method both on serial and parallel environments. The parallel version of GPDT2 is obtained by performing in parallel the steps 2 and 3 of each decomposition iteration, as explained in [27]: the subproblem (2) is solved by a parallel version of the GVPM and the gradient updating (3) is obtained by distributing the rows of $[G_{\mathcal{BB}}\ G_{\mathcal{BN}}]$ among the available processors.

We evaluate GPDT2 on the most significant tests of Table I; the cases with $n_{sp} = n_c$ are not considered since, as pointed out before, they lead to useless performance. The results are reported in Table IV and show that, even if the optimal timings are achieved for large values of n_{sp}, the cases with smaller n_{sp} are generally executed with comparable times. Significant differences are observed only on the MNIST test problem where, for small n_{sp} values, the too large number of kernel evaluations, very expensive in this application, reduces the performance excessively. We will face possible strategies for overcoming this drawback in a future version of GPDT2. From the results in Table IV we can conclude that GPDT2 is efficient with many different sizes of the working set and its performance is not strictly dependent on a correct guess of n_{sp}. On the other hand, Table I shows that the SVM^{light} effectiveness can markedly decrease if it runs with non optimal values for n_{sp}.

Another important feature of GPDT2 is its good parallel properties. In fact larger decomposition subprob-

Table V Scaling of GPDT2 in a parallel environment

PE	it	time	sp_r	it	time	sp_r
UCI Adult data set, $n = 32561$						
	$n_{sp} = 200,\ n_c = 100$			$n_{sp} = 1300,\ n_c = 650$		
1	420	253.6		32	215.9	
2	471	135.6	1.9	31	116.3	1.9
4	388	90.8	2.8	30	65.8	3.3
8	385	59.7	4.2	31	41.5	5.2
Web data set, $n = 49749$						
	$n_{sp} = 600,\ n_c = 300$			$n_{sp} = 1500,\ n_c = 750$		
1	190	207.6		25	168.1	
2	241	116.8	1.8	23	88.1	1.9
4	193	78.3	2.7	27	64.7	2.6
8	179	60.4	3.4	23	47.0	3.6
MNIST data set, $n = 60000$						
	$n_{sp} = 2000,\ n_c = 300$			$n_{sp} = 3000,\ n_c = 800$		
1	22	662.8		9	670.6	
2	22	375.5	1.8	8	308.3	2.2
4	22	178.4	3.7	8	166.2	4.0
8	23	110.8	6.0	8	108.2	6.2

lems lowers the number of iterations of the decomposition technique, thus reducing the dependencies in the computation and improving the scalability in a parallel environment. Numerical evidence of this fact is shown in Table V, where the performance of the parallel GPDT2 on the different data sets is shown. The number of iterations and the total training time are reported for different numbers of processing elements (PEs) and decomposition subproblem sizes (n_{sp} and n_c). The sp_r columns indicate the relative speedup of the method ($\text{sp}_\text{r}(q) = t_1/t_q$, where t_q is the time needed by the program on q PEs). It can be observed that generally the runs with larger n_{sp} exhibit a better speedup than the same runs with smaller n_{sp}. This behavior is particularly evident on the MNIST data set: even if the serial version has a lower execution time with a smaller n_{sp}, all the parallel runs require less time when larger n_{sp} is used.

Finally, just to emphasize some advantages of the proposed technique over the SVM^{light} software, we can observe that, when GPDT2 decomposes in sufficiently large subproblems, an execution time lower than that given by SVM^{light} is obtained for all the test problems considered in this work. Besides, no parallel version of SVM^{light} is available; this means that great speedups can be obtained if a parallel system is available. For example, on the considered architecture, when 8 processors are used, one can train on the MNIST data set sized $n = 60000$ in 108.2 seconds with the parallel GPDT2 instead of 901.6 seconds needed by the serial SVM^{light} software.

5 Conclusions

This work is about decomposition techniques for the large quadratic program arising in training Support Vector Machines. It is mainly focused on selection rules for updating the working set, which is a crucial step for the effectiveness of these techniques. Some of the most popular working set selections [8, 9, 27] are analyzed and a new selection strategy, able to improve the convergence rate of the decomposition schemes based on large sized working sets, is developed. The improved strategy is derived by analysing two issues that are essential in case of large sized working set: how many new components can enter the working set at each iteration and which components of the previous working set it is convenient to keep. For the former, an adaptive strategy for reducing the number of new variables entering the working set is proposed while, for the latter, a recent criterion is improved by taking into account how long a variable remains into the working set. Numerical evidence of these improvements are given by showing the performance of a decomposition technique based on the new selection strategy and on a gradient projection method for the inner QP subproblems. Both the number of decomposition iterations and the execution time have been improved with respect to other existing techniques on some common large data sets. Finally, the efficient use of large sized working sets has improved the scaling properties of the decomposition method when executed in a parallel environment.

6 Appendix: test problems and test architecture

All the numerical results of this work are obtained by solving the quadratic programs arising in training Gaussian SVMs on the following data sets:

- **UCI Adult data set**

  This data set [17], allows to train an SVM to predict whether a household has an income greater than \$50000. After appropriate discretization [21], the inputs are 123-dimensional binary sparse vectors with sparsity level ≈ 89%. We use the largest version of the data set sized 32561 and a smaller version sized 3185. For both the cases, we train a Gaussian SVM with $C = 1$ and $\sigma = \sqrt{10}$.

- **Web data set**

  The data set [21], (available at <http://www.research.microsoft.com/∼jplatt/smo.html>), concerns a web page classification problem with a binary representation based on 300 keyword features. On average, the sparsity level of the examples is about 96%. We use two versions of the data set: the largest sized 49749 and a smaller with size 9888. The Gaussian SVM parameters are: $C = 5$ and $\sigma = \sqrt{10}$.

- **MNIST data set**

  The MNIST database of handwritten digits [12] contains 784-dimensional nonbinary sparse vectors; the size of the database is 60000 and the sparsity level of the inputs is ≈ 81%. A Gaussian SVM for the class "8" is trained with parameters $C = 10$ and $\sigma = 1800$.

All the experiments are carried out with standard C codes running on the multiprocessor IBM SP4 equipped with 16 nodes of 32 Power4 1.3GHz CPUs each and 64GB virtually shared RAM per node, at CINECA supercomputing Centre, Bologna, Italy (<http://www.cineca.it>). The serial versions of the decomposition techniques run on a single processor of the SP4. The parallel GPDT2 version uses standard MPI communication routines and run on processors within the same node.

References

[1] C.C. Chang, C.W. Hsu, C.J. Lin (2000), The Analysis of Decomposition Methods for Support Vector Machines, *IEEE Transactions on Neural Networks* **11(4)**, 1003-1008.

[2] C.C. Chang, C.J. Lin (2002), LIBSVM: a library for support vector machines, available at <http://www.csie.ntu.edu.tw/∼cjlin/libsvm>.

[3] R. Collobert, S. Benjo (2001), SVMTorch: Support Vector Machines for Large-Scale Regression Problems, *Journal of Machine Learning Research* **1**, 143–160.

[4] N. Cristianini, J. Shawe-Taylor (2000), An Introduction to Support Vector Machines and other Kernel-Based Learning Methods, Cambridge University Press, .

[5] Y.H. Dai, R. Fletcher (2003), New Algorithms for Singly Linearly Constrained Quadratic Programs Subject to Lower and Upper Bounds, Research Report NA/216, Department of Mathematics, University of Dundee.

[6] G.W. Flake, S. Lawrence (2002), Efficient SVM Regression Training with SMO, *Machine Learning* **46**(1), 271-290, available at <http://www.neci.nec.com/homepages/flake/nodelib/html>.

[7] E. Galligani, V. Ruggiero, L. Zanni (2003), Variable Projection Methods for Large-Scale Quadratic Optimization in Data Analysis Applications, *Equilibrium Problems and Variational Models*, P. Daniele et al., eds., Nonconvex Optim. and Its Applic. **68**, Kluwer Academic Publ., Boston, 186–211.

[8] C.W. Hsu, C.J. Lin (2002), A Simple Decomposition Method for Support Vector Machines, *Machine Learning*, **46**, 291–314.

[9] T. Joachims (1998), Making Large-Scale SVM Learning Practical, *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C.J.C. Burges and A. Smola, eds., MIT Press, Cambridge, MA.

[10] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy (2001), Improvements to Platt's SMO Algorithm for SVM Classifier Design, *Neural Computation* **13**, 637–649.

[11] S.S. Keerthi, E.G. Gilbert (2002), Convergence of a Generalized SMO Algorithm for SVM Classifier Design, *Machine Learning* **46**, 351-360.

[12] Y. LeCun, The MNIST Database of Handwritten Digit, available at <http://yann.lecun.com/exdb/mnist>.

[13] C.J. Lin (2001), On the Convergence of the Decomposition Method for Support Vector Machines, *IEEE Transactions on Neural Networks* **12**, 1288-1298.

[14] C.J. Lin (2001), Linear Convergence of a Decomposition Method for Support Vector Machines, Tech. Rep., Department of Computer Science and Information Engineering, National Taiwan University.

[15] C.J. Lin (2002), Asymptotic Convergence of an SMO Algorithm Without any Assumptions, *IEEE Transactions on Neural Networks* **13**, 248-250.

[16] O.L. Mangasarian, D.R. Musicant (1999), Successive Overrelaxation for Support Vector Machines, *IEEE Transactions on Neural Networks* **10**, 1032-1037.

[17] P.M. Murphy, D.W. Aha (1992), UCI Repository of Machine Learning Databases, available at <http://www.ics.uci.edu/∼mlearn/MLRepository.html>.

[18] E. Osuna, R. Freund, F. Girosi (1997), Training Support Vector Machines: an Application to Face Detection, *Proceedings of the CVPR'97*, Puerto Rico, 130–136.

[19] L. Palagi, M. Sciandrone (2002), On the Convergence of a Modified Version of SVM^{light} Algorithm, Technical Report 567, Istituto di Analisi dei Sistemi ed Informatica, Roma.

[20] P.M. Pardalos, N. Kovoor (1990), An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Upper and Lower Bounds, *Math. Programming* **46**, 321–328.

[21] J.C. Platt (1998), Fast Training of Support Vector Machines using Sequential Minimal Optimization, *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges and A. Smola, eds., MIT Press, Cambridge, MA.

[22] V. Ruggiero, L. Zanni (2000), A Modified Projection Algorithm for Large Strictly Convex Quadratic Programs, *J. Optim. Theory Appl.* **104**(2), 281–299.

[23] V. Ruggiero, L. Zanni (2000), Variable Projection Methods for Large Convex Quadratic Programs, *Recent Trends in Numerical Analysis*, D. Trigiante, ed., Advances in the Theory of Computational Mathematics 3, Nova Science Publ., 299–313.

[24] T. Serafini, G. Zanghirati, L. Zanni (2003), Gradient Projection Methods for Quadratic Programs and Applications in Training Support Vector Machines, *Optimization Methods and Software*, (to appear).

[25] V.N. Vapnik (1998), Statistical Learning Theory, John Wiley and sons, New York.

[26] G. Zanghirati, L. Zanni (2003), Variable Projection Method for Large Quadratic Programs in Training Support Vector Machines, Technical Report 339, Department of Mathematics, University of Ferrara.

[27] G. Zanghirati, L. Zanni (2003), A Parallel Solver for Large Quadratic Programs in Training Support Vector Machines, *Parallel Computing* **29**, 535–551.