ISSN 2281-4299

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

A Class of Convergent Parallel Algorithms for SVMs Training

Andrea Manno
Laura Palagi
Simone Sagratella

Technical Report n. 17, 2014

A Class of Parallel Decomposition Algorithms for SVMs Training*

Andrea Manno†, Laura Palagi‡and Simone Sagratella§

Updated version of Tech. Report 17/2014 - November 6, 2015

Abstract

The training of Support Vector Machines may be a very difficult task when dealing with very large datasets. The memory requirement and the time consumption of the SVMs algorithms grow rapidly with the increase of the data. To overcome these drawbacks, we propose a parallel decomposition algorithmic scheme for SVMs training for which we prove global convergence under suitable conditions. We outline how these assumptions can be satisfied in practice and we suggest various specific implementations exploiting the adaptable structure of the algorithmic model.

Keywords. Decomposition Algorithm, Big Data, Support Vector Machine, Machine Learning, Parallel Computing

1 Introduction

A Support Vector Machine (SVM) is a well known classification and regression tool that has spread in many scientific fields during the last two decades, see [5]. Given a training set of n input-target pairs

$$D = \{(\mathbf{z}_r, y_r),\ r = 1, \ldots, n,\ \mathbf{z}_r \in \mathbb{R}^m,\ y_r \in \{-1, 1\}\},$$

*The work of Laura Palagi was partially supported by the italian project PLATINO (Grant Agreement n. PON01_01007); the work of Simone Sagratella was partially supported by the grant: Avvio alla Ricerca 397, Sapienza University of Rome

†Dipartimento di Ingegneria dell'Informazione, Università degli Studi Firenze,Via Santa Marta, 3 50139, Florence, Italy. (andrea.manno@unifi.it)

‡Department of Computer, Control and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00185, Rome Italy. (palagi@dis.uniroma1.it)

§Department of Computer, Control and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00185, Rome Italy. (sagratella@dis.uniroma1.it)

an SVM provides a prediction model used to classify new unlabeled samples. The dual formulation of an SVM training problem is

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T Q\mathbf{x} - \mathbf{e}^T\mathbf{x} \\ & \mathbf{y}^T\mathbf{x} = 0 \\ & \mathbf{0} \leq \mathbf{x} \leq C\mathbf{e}, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{e} \in \mathbb{R}^n$ is a vector of all ones, $C > 0$ is a positive constant, $\mathbf{y} \in \{-1, 1\}^n$ and Q is an $n \times n$ symmetric positive semidefinte matrix. Each component of $\mathbf{x}$ is associated with a sample of the training set and $\mathbf{y}$ is the vector of the corresponding labels. Entries of Q are defined by

$$Q_{rq} = y_r y_q K(\mathbf{z}_r, \mathbf{z}_q), \qquad r, q = 1, 2, \ldots, n,$$

where $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a given kernel function [23].
Many real SVM applications are characterized by a large dimensional training set. This implies that hessian matrix Q is so big that it cannot be entirely stored in memory. For this reason classical optimization algorithms that use first and second order information cannot be used to efficiently solve problem (1).
To overcome this difficulty, many decomposition algorithms have been proposed in literature. At each iteration, they split the original problem into a sequence of smaller subproblems where only a subset of variables (working set) are updated. Columns of the hessian submatrix corresponding to each subproblem are, partially or entirely, recomputed at each step. These strategies can be mainly divided into SMO (Sequential Minimal Optimization) and non-SMO methods. SMO algorithms (see e.g. [3, 21]) work with subproblems of dimension two, so that their solutions can be computed analytically; while non-SMO algorithms (see e.g. [10, 13]) need an iterative optimization method to solve each subproblem. From the theoretical point of view, the policy for updating the working set plays a crucial role to prove convergence. In case of SMO methods, a proper selection rule based on the maximal violating principle is sufficient to ensure asymptotic convergence of the decomposition scheme [4, 16]. For larger working sets, convergence proofs are available under further conditions [15, 19, 20].

In recent years SVMs have been applied to huge datasets, mainly related to web-oriented applications. To reduce the big amount of time needed for the training of SVMs on such huge datasets, parallel algorithms have been proposed. Some of these parallel approaches to SVMs consists in distributing the most expensive tasks, such as subproblems solving and gradient updating, among the available processors, see [11, 26, 27]. Another way of fruitfully

exploit parallelism is based on splitting the training data into subsets and distributing them among the processors [2, 28]. Among these parallel techniques, there are also the so called Cascade-SVM (see [12, 25]) that has been introduced to face big dimensional instances. While achieving a good reduction of the training time respect to sequential methods, these methods may lack convergence properties or may require strong assumptions to prove it.

Actually, combining decomposition rules, for the selection of working sets, and parallelism makes the proof of convergence a very difficult task, see [18]. This is mainly due to nonseparability of the feasible set of problem (1).

In this work we propose a class of convergent parallel training algorithms based on the decomposition of problem (1) into a partition of subproblems that can be solved independently by parallel processes. The convergence to a global optimum of problem (1) is proved under realistic assumptions. It partially exploits results introduced in [9, 24]. The algorithmic framework presented may include, as a special case, other convergent theoretical models like [18].

The paper is organized as follows: in Section 2 we introduce some preliminary results; in Section 3 we introduce a general parallel algorithmic scheme. We analyze its convergence properties in Sections 4, 5 and 6; in Section 7 we discuss about some possible practical implementations.

Notation In the following we use this notation. Vectors are boldface. Given a vector $\mathbf{x} \in \mathbb{R}^n$ with components x_r and a subset of indices $P \subseteq \{1, \ldots, n\}$ we denote by $\mathbf{x}_P \in \mathbb{R}^{|P|}$ the subvector made up of components x_r with $r \in P$ and by $\mathbf{x}_{-P} \in \mathbb{R}^{n-|P|}$ the subvector made up of components x_r with $r \notin P$. By $\|\cdot\|$ we indicate the euclidean norm, whereas the zero norm of a vector $\|\mathbf{x}\|_0$ denotes the number of nonzero components of vector $\mathbf{x}$. Further given a square $n \times n$ matrix $\mathbf{Q}$, we denote by $\mathbf{Q}_{*r}$ the $r-$th column of the matrix. Given two subsets of indices $P_r, P_q \subseteq \{1, \ldots, n\}$, we write $Q_{P_r P_q}$ to indicate the $|P_r| \times |P_q|$ submatrix of Q with row indices in block P_r and column indices in block P_q. We denote by $\lambda^Q_{\min}$ and $\lambda^Q_{\max}$ respectively the minimum and maximum eigenvalue of a square matrix Q. For the sake of simplicity we denote the $r-$th component of the gradient as $\nabla f(\mathbf{x})_r = \frac{\partial f(\mathbf{x})}{\partial x_r}$ and as $\nabla_P f(\mathbf{x}) \in \mathbb{R}^{|P|}$ the subvector of the gradient made up of components $\frac{\partial f(\mathbf{x})}{\partial x_r}$ with $r \in P$. We denote by $\mathcal{F}$ the feasible set of problem (1), namely

$$\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{y}^T\mathbf{x} = 0, \quad \mathbf{0} \leq \mathbf{x} \leq C\mathbf{e}\}.$$

Note that all the results that we report in the sequel hold also in the case of feasible set $\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{y}^T\mathbf{x} = b, \quad \mathbf{0} \leq \mathbf{x} \leq C\mathbf{e}\}$, where $\mathbf{y} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, but for sake of simplicity we refer to the case $b = 0$ and $\mathbf{y} \in \{-1, 1\}^n$.

2 Optimality Conditions and Preliminary Results

Let us consider a solution $\mathbf{x}^*$ of problem (1). Since constraints are linear and the objective function is convex, necessary and sufficient conditions for optimality are the Karush-Kuhn-Tucker (KKT) conditions that state that there exists a scalar s such that for all indices $r \in \{1, \ldots, n\}$:

$$\begin{aligned} &\nabla f(\mathbf{x}^*)_r + s y_r \geq 0 \quad \text{if } x_r^* = 0 \\ &\nabla f(\mathbf{x}^*)_r + s y_r \leq 0 \quad \text{if } x_r^* = C \\ &\nabla f(\mathbf{x}^*)_r + s y_r = 0 \quad \text{if } 0 < \ x_r^* < C. \end{aligned} \tag{2}$$

It is well known (see e.g.[13]) that KKT conditions can be written in a more compact form by introducing the following sets

$$I_{up}(\mathbf{x}) := \{r \subseteq \{1, \ldots, n\} : \ x_r < C, \ y_r = 1, \ \text{or } x_r > 0, \ y_r = -1\},$$

$$I_{low}(\mathbf{x}) := \{r \subseteq \{1, \ldots, n\} : \ x_r < C, \ y_r = -1, \ \text{or } x_r > 0, \ y_r = 1\}.$$

Assuming that $I_{up}(\mathbf{x}^*) \neq \emptyset$ and $I_{low}(\mathbf{x}^*) \neq \emptyset$, then we can rewrite (2) as

$$m(\mathbf{x}^*) = \max_{r \in I_{up}(\mathbf{x}^*)} -\frac{\nabla f(\mathbf{x}^*)_r}{y_r} \leq \min_{r \in I_{low}(\mathbf{x}^*)} -\frac{\nabla f(\mathbf{x}^*)_r}{y_r} = M(\mathbf{x}^*). \tag{3}$$

By the convexity of problem (1), we can say that $\mathbf{x}^*$ is optimal if and only if either $I_{up}(\mathbf{x}^*) = \emptyset$ or $I_{low}(\mathbf{x}^*) = \emptyset$ or condition (3) holds.

Such a form of the KKT conditions is the basis of most efficient sequential decomposition algorithms for the solution of problem (1). In decomposition algorithms the sequence $\{\mathbf{x}^k\}$ is obtained by changing at each iteration only a subset of the variables, let's say $\mathbf{x}_{P_i}$ with $P_i \subset \{1, \ldots, n\}$, whilst the other $\mathbf{x}_{-P_i}$ remain unchanged. Thus the sequence takes the form

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k,$$

where $\mathbf{d}^k$ is a sparse feasible descent direction such that $\|\mathbf{d}^k\|_0 = |P_i|$ with $|P_i| << n$ and α^k represents a stepsize along this direction. Whatever the feasible direction $\mathbf{d}^k$ is, since the objective function is quadratic and convex, the choice of the stepsize can be performed by using an exact minimization of the objective function along $\mathbf{d}^k$. Indeed, let $\bar{\beta} > 0$ be the largest feasible step at $\mathbf{x}^k \in \mathcal{F}$ along the descent direction $\mathbf{d}^k$ then

$$\alpha^k := \min \left\{ -\frac{\nabla \mathbf{f}(\mathbf{x}^k)^T \mathbf{d}^k}{\mathbf{d}^{kT} \mathbf{Q} \mathbf{d}^k}, \ \bar{\beta} \right\}. \tag{4}$$

Sequential decomposition methods differ in the choice of the direction $\mathbf{d}^k$, or equivalently in the choice of the so called working set P_i.

Sequential Minimal Optimization (SMO) methods uses feasible descent directions $\mathbf{d}^k$ with $\|\mathbf{d}^k\|_0 = 2$ which is the minimal possible cardinality due to the equality constraint. In a feasible point $\mathbf{x} \in \mathcal{F}$ a feasible direction with two nonzero components $\mathbf{d}^{(ij)}$ is given by

$$d_r^{(ij)} := \begin{cases} \dfrac{1}{y_r} & \text{if } r = i \\ -\dfrac{1}{y_r} & \text{if } r = j \\ 0 & \text{otherwise} \end{cases}, \quad r = 1, \ldots, n. \tag{5}$$

for any pair $(i, j) \in I_{up}(\mathbf{x}) \times I_{low}(\mathbf{x})$. We say that a pair $(i, j) \in I_{up}(\mathbf{x}) \times I_{low}(\mathbf{x})$ is a violating pair at $\mathbf{x}$ if it satisfied also $\nabla f(\mathbf{x})^T \mathbf{d}^{(ij)} < 0$.

The exact optimal stepsize $\alpha \geq 0$ along a direction $\mathbf{d}^{(ij)}$ can be efficiently computed by noting that in (4) we have

$$\bar{\beta} = \min \{\beta_i, \beta_j\}, \tag{6}$$

where

$$\beta_h := \begin{cases} x_h & \text{if } d_h^{(ij)} < 0 \\ C - x_h & \text{if } d_h^{(ij)} > 0. \end{cases} \tag{7}$$

Thus we get the value of the optimal stepsize α along a direction $\mathbf{d}^{(ij)}$ as

$$\alpha := \min \left\{ -\frac{\nabla \mathbf{f}_i y_i - \nabla \mathbf{f}_j y_j}{Q_{ii} + Q_{jj} - 2 y_i y_j Q_{ij}}, \bar{\beta} \right\}. \tag{8}$$

Among such minimal descent directions, i.e. violating pairs, a crucial role is played by the so called *Most Violating Pair* (MVP) direction (see e.g. [13]). To be more specific, given a feasible point $\mathbf{x}$, let us define the sets

$$I_{up}^{MVP}(\mathbf{x}) := \left\{ i \in I_{up}(\mathbf{x}) : \; i \in \arg\max_{h \in I_{up}(\mathbf{x})} -\frac{\nabla f(\mathbf{x})_h}{y_h} \right\},$$

$$I_{low}^{MVP}(\mathbf{x}) := \left\{ j \in I_{low}(\mathbf{x}) : \; j \in \arg\min_{h \in I_{low}(\mathbf{x})} -\frac{\nabla f(\mathbf{x})_h}{y_h} \right\}.$$

If $\mathbf{x}$ is not a solution of problem (1), then $(i_{\texttt{MVP}}, j_{\texttt{MVP}}) \in I_{up}^{MVP}(\mathbf{x}) \times I_{low}^{MVP}(\mathbf{x})$ is a pair, possibly not unique, that violates the KKT conditions at most and it is said *Most Violating Pair* (MVP). In the sequel, for the sake of notational simplicity, we assume that, for every feasible $\mathbf{x}$, the MVP is unique as this

makes no difference in our analysis.
The direction $\mathbf{d}_{\texttt{MVP}} \in \mathbb{R}^n$ corresponding to the pair $(i_{\texttt{MVP}}, j_{\texttt{MVP}}) \in I_{up}^{MVP}(\mathbf{x}) \times I_{low}^{MVP}(\mathbf{x})$ is, among all feasible descent directions with only two nonzero components, the steepest descent one at $\mathbf{x}$.
Now we are ready to introduce the definition of "most violating step". Let $\mathbf{x}_{\texttt{MVP}} = \mathbf{x} + \alpha_{\texttt{MVP}}\mathbf{d}_{\texttt{MVP}}$ with $\alpha_{\texttt{MVP}}$ obtained by (8) with $i = i_{\texttt{MVP}}$, $j = j_{\texttt{MVP}}$.

Definition 2.1 (Most Violating Step) *At a point $\mathbf{x} \in \mathcal{F}$, we define the "Most Violating Step" (MVS) S_{MVP} as:*

$$S_{MVP}(\mathbf{x}) := \|\mathbf{x}_{MVP} - \mathbf{x}\| = |\alpha_{MVP}|\|\mathbf{d}_{MVP}\|. \tag{9}$$

In particular, since $y^i \in \{-1, 1\}$ we have that $S_{\texttt{MVP}}(\mathbf{x}) = |\alpha_{\texttt{MVP}}|\sqrt{2}$.

We can state the optimality condition using the definition of MVS.

Proposition 2.2 *A point $\mathbf{x}^* \in \mathcal{F}$ is optimal for problem (1) if and only if either $I_{up}(\mathbf{x}^*) = \emptyset$ or $I_{low}(\mathbf{x}^*) = \emptyset$ or $S_{MVP}(\mathbf{x}^*) = 0$.*

Proof. As said above $\mathbf{x}^*$ is optimal for problem (1) if and only if either $I_{up}(\mathbf{x}^*) = \emptyset$ or $I_{low}(\mathbf{x}^*) = \emptyset$ or condition (3) holds. Therefore we only have to show that, in the case in which $I_{up}(\mathbf{x}^*) \neq \emptyset$ and $I_{low}(\mathbf{x}^*) \neq \emptyset$, the following holds:

$$S_{\texttt{MVP}}(\mathbf{x}^*) = 0 \quad \Leftrightarrow \quad m(\mathbf{x}^*) \leq M(\mathbf{x}^*).$$

Since $I_{up}(\mathbf{x}^*) \neq \emptyset$ and $I_{low}(\mathbf{x}^*) \neq \emptyset$, we can compute a pair $(i^*_{\texttt{MVP}}, j^*_{\texttt{MVP}}) \in I_{up}^{MVP}(\mathbf{x}^*) \times I_{low}^{MVP}(\mathbf{x}^*)$ and $\mathbf{d}^*_{\texttt{MVP}} = \mathbf{d}^{(i^*_{\texttt{MVP}}, j^*_{\texttt{MVP}})}$ as in (5). $m(\mathbf{x}^*) \leq M(\mathbf{x}^*)$ is equivalent to inequality $\nabla\mathbf{f}(\mathbf{x}^*)^T\mathbf{d}^*_{\texttt{MVP}} \geq 0$. By noting that $\mathbf{d}^*_{\texttt{MVP}}$ is a feasible direction at $\mathbf{x}^*$, then from (6) we have $\bar{\beta} > 0$. Therefore by (8) we can conclude that $\nabla\mathbf{f}(\mathbf{x}^*)^T\mathbf{d}^*_{\texttt{MVP}} \geq 0$ if and only if $\alpha^*_{\texttt{MVP}} = 0$ and, in turn, if and only if $S_{\texttt{MVP}}(\mathbf{x}^*) = 0$, so that the proof is complete. ■ □

3 A Parallel Decomposition Model

In this section we introduce a parallel decomposition scheme for finding a solution of problem (1). The theoretical properties and implementation details are discussed in the next sections. The algorithm fits in a decomposition framework where, as usual, the solution of problem (1) is obtained by a sequence of solution of smaller problems in which only a subset of the variables is changed. To fix notation, let $\mathbf{x}^k \in \mathcal{F}$ and consider a subset $P_i \subset \{1, \ldots, n\}$,

so that $\mathbf{x}^k$ can be partitioned as $\mathbf{x}^k := (\mathbf{x}^k_{P_i}, \mathbf{x}^k_{-P_i})$. The problem of minimizing over $\mathbf{x}_{P_i}$ with $\mathbf{x}_{-P_i}$ fixed to the current value $\mathbf{x}^k_{-P_i}$ is:

$$\min_{\mathbf{x}_{P_i} \in \mathcal{F}^k_{P_i}} f_{P_i}(\mathbf{x}_{P_i}, \mathbf{x}^k_{-P_i}) + \frac{\tau^k_i}{2} \|\mathbf{x}_{P_i} - \mathbf{x}^k_{P_i}\|^2, \tag{10}$$

where a proximal point term with $\tau^k_i \geq 0$ has been added [20], and the feasible set is

$$\mathcal{F}^k_{P_i} := \{\mathbf{x}_{P_i} \in \mathbb{R}^{|P_i|} : \mathbf{y}^T_{P_i} \mathbf{x}_{P_i} = \mathbf{y}^T_{P_i} \mathbf{x}^k_{P_i},\ 0 \leq \mathbf{x}_{P_i} \leq C\mathbf{e}_{P_i}\}.$$

Problem (10) is still quadratic and convex with hessian matrix $\mathbf{Q}_{P_iP_i} + \tau^k_i I_{P_i}$ symmetric and positive semidefinite, and linear term given by $\ell^k_{P_i} + \tau^k_i x^k_{P_i}$, where

$$\ell^k_{P_i} = \sum_{P_j \in \mathcal{P}, j \neq i} \mathbf{Q}_{P_iP_j} \mathbf{x}^k_{P_j} - \mathbf{e}_{P_i}.$$

We denote $\widehat{\mathbf{x}}^k_{P_i}$ as a solution of problem (10), which is unique either if $\tau^k_i > 0$ or if $\mathbf{Q}_{P_iP_i}$ is positive definite.

The parallel scheme that we are going to define is not based on splitting the data set or in parallelizing the linear algebra, but on defining a bunch of subproblems to be solved by means of parallel and independent processes. Unlike sequential decomposition methods, the search direction $\mathbf{d}^k$ is obtained by summing up smaller directions obtained by solving in parallel a bunch of subproblems of type (10).

Let us define a partition $\mathcal{P} = \{P_1, P_2, \ldots\}$ of the set of all indices $\{1, \ldots, n\}$. By definition we have that $P_i \cap P_j = \emptyset$ and $\cup_i P_i = \{1, \ldots, n\}$. The basic idea underlying the definition of the parallel decomposition algorithm is summarized in the scheme below.

Algorithm 3.1 Parallel Decomposition Model

Initialization Choose $\mathbf{x}^0 \in \mathcal{F}$ and set $k = 0$.

Do while (a stopping criterion satisfied)

S.1 (Partition definition)

Set $\mathcal{P}^k = \{P_1, P_2, \ldots, P_{N^k}\}$ and set $\tau_i^k \geq 0$ for all $i = 1, \ldots, N^k$.

S.2 (Blocks selection)

Choose a subset of blocks $\mathcal{J}^k \subseteq \mathcal{P}^k$.

S.3 (Parallel computation)

For all $P_i \in \mathcal{J}^k$ compute in parallel the optimal solution $\widehat{\mathbf{x}}_{P_i}^k$ of problem (10).

S.4 (Direction) Set $\mathbf{d}^k \in \mathbb{R}^n$ block-wise as

$$\mathbf{d}_{P_i}^k = \begin{cases} \widehat{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k & \text{if } P_i \in \mathcal{J}^k, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{11}$$

S.5 (Stepsize) Choose a suitable stepsize $\alpha^k > 0$.

S.6 (Update) Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k$ and $k = k + 1$.

End While

Return $\mathbf{x}^k$.

The scheme above encompasses different possible algorithms depending on the choice of the partition $\mathcal{P}^k$ at **S.1**, the blocks selection $\mathcal{J}^k$ at **S.2** and the stepsize rule at **S.5**.

A widely used standard feasible point is $\mathbf{x}^0 = \mathbf{0}$, but of course different choices are possible if available. The choice of $\mathbf{x}^0 = \mathbf{0}$ presents the advantage that also the gradient is available being $\nabla f(\mathbf{x}^0) = -\mathbf{e}$.

Checking optimality of the current point $\mathbf{x}^k$ may require zero or first order information depending on the stopping criterion adopted. A standard stopping criterion is based on checking condition $m(\mathbf{x}^k) \leq M(\mathbf{x}^k) - \eta$, for a given tolerance $\eta > 0$. In this case the updated gradient $\nabla f(\mathbf{x}^{k+1})$ is needed at each iteration. It is well known that for large scale problem this is a big effort due to expensive kernel evaluations. Indeed we have the following

iterative updating rule

$$\nabla f(\mathbf{x}^{k+1}) = \nabla f(\mathbf{x}^k) + \alpha^k \sum_{P_i \in \mathcal{J}^k} \sum_{h \in P_i} Q_{*h} d_h^k.$$

At **S.1** a partition $\mathcal{P}^k$ of $\{1, \ldots, n\}$ is defined. We point out that both the number N^k of blocks and their composition in $\mathcal{P}^k$ can vary from one iteration to another. For notational simplicity we omit dependency of blocks $P_1, P_2, \ldots, P_{N^k}$ on the iteration k. As usual in decomposition algorithms, a correct choice of the partition is crucial for proving global convergence of the method.

At **S.2** a subset $\mathcal{J}^k$ of blocks in $\mathcal{P}^k$ is selected. These blocks are the only ones used at **S.3** to compute a search direction $\mathbf{d}^k$ according to (11). The selection of blocks makes the algorithmic scheme more flexible since one can set the overall computational burden.

At **S.3** we obtain an optimal solutions $\widehat{\mathbf{x}}_{P_i}^k$ of problem (10) for each $P_i \in \mathcal{J}^k$. Note that $\widehat{\mathbf{x}}_{P_i}^k$ satisfies the optimality condition

$$\left[\nabla_{P_i} f_{P_i}(\widehat{\mathbf{x}}_{P_i}^k, \mathbf{x}_{-P_i}^k) + \tau_i^k (\widehat{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k)\right]^T \mathbf{d}_{P_i} \geq 0 \tag{12}$$

for any feasible direction $\mathbf{d}_{P_i}$ at $\widehat{\mathbf{x}}_{P_i}^k$. The computational burden of this step consists in

- computing vector $\ell_{P_i}^k$ to construct the objective function of (10) for all blocks $P_i \in \mathcal{J}^k$,
- solving the $|\mathcal{J}^k|$ subproblems.

These $|\mathcal{J}^k|$ convex quadratic problems can be distributed to different processes in order to be solved in a parallel fashion.
At **S.4** the algorithm computes search direction $\mathbf{d}^k$.
At **S.5** the algorithm computes stepsize α^k. We show in the next section (Theorems 4.2, 4.3 and 4.4) that, in order to have convergence of the algorithm, α^k can be computed according to a simple diminishing stepsize rule or a linesearch procedure (including the exact minimization rule).

4 Theoretical Analysis

In this section we analyze the theoretical properties of Algorithm 3.1. To this aim, we first introduce the definition of descent block and of descent iteration that are crucial for the following analysis.

Definition 4.1 (Descent block) *Given $\epsilon > 0$. At a feasible point $\mathbf{x}^k$, we say that the block of variables $P_i \subseteq \{1, \ldots, n\}$ is a* descent block *if it satisfies*

$$\|\widehat{\mathbf{x}}_{P_i} - \mathbf{x}^k_{P_i}\| \geq \epsilon S_{\mathit{MVP}}(\mathbf{x}^k), \tag{13}$$

where $\widehat{\mathbf{x}}_{P_i}$ is the optimal solution of the corresponding problem (10) *and $S_{\mathit{MVP}}(\mathbf{x}^k) = \|\mathbf{x}^k_{\mathit{MVP}} - \mathbf{x}^k\|$.*

Whenever at least one descent block P_i is selected in $\mathcal{J}^k$ at **S.2** of Algorithm 3.1, we say that iteration k is a *descent iteration*.

Under the assumption that at least one descent block is selected for optimization at **S.3** of the parallel algorithmic model, we will prove that by using a suitable α^k at **S.5** the sequence $\{\mathbf{x}^k\}$ produced by the algorithm satisfies

$$\lim_{k\to\infty} S_{\mathtt{MVP}}(\mathbf{x}^k) = 0. \tag{14}$$

We prove later on that the assumption is easy to achieve. We first consider the case when the stepsize α^k is determined by a standard Armjio linesearch procedure along the direction $\mathbf{d}^k$.

Theorem 4.2 *Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 3.1 where $\alpha^k \leq 1$ at* **S.5** *satisfies the following Armjio condition*

$$\mathbf{f}(\mathbf{x}^k + \alpha^k \mathbf{d}^k) \leq \mathbf{f}(\mathbf{x}^k) + \theta\alpha^k \nabla\mathbf{f}(\mathbf{x}^k)^T \mathbf{d}^k, \tag{15}$$

with $\theta \in (0, 1)$. Assume that for all iterations k

(i) *a descent iteration $\tilde{k}$ with $k \leq \tilde{k} \leq k + L$ for a finite $L \geq 0$ is generated;*

(ii) *either $\tau_i^k \geq \underline{\tau} > 0$ or $Q_{P_iP_i} \succ 0$, for all $P_i \in \mathcal{J}^k$.*

Then either Algorithm 3.1 terminates in a finite number of iterations to a solution of problem (1) *or $\{\mathbf{x}^k\}$ admits a limit point and it satisfies* (14).

Proof. First of all we note that $\{\mathbf{x}^k\}$ is a feasible sequence, in fact it is sufficient to show that for all k if $\mathbf{x}^k$ is feasible then also $\mathbf{x}^{k+1}$ is feasible. Since for all $P_i \in \mathcal{J}^k$ it holds that $\mathbf{y}_{P_i}{}^T \widehat{\mathbf{x}}_{P_i} = \mathbf{y}_{P_i}{}^T \mathbf{x}^k_{P_i}$, then we can write

$$\begin{aligned}
\mathbf{y}^T\mathbf{x}^{k+1} &= \sum_{P_i\in\mathcal{J}^k} \mathbf{y}_{P_i}{}^T \left(\mathbf{x}^k_{P_i} + \alpha^k(\widehat{\mathbf{x}}_{P_i} - \mathbf{x}^k_{P_i})\right) + \sum_{P_i\notin\mathcal{J}^k} \mathbf{y}_{P_i}{}^T \mathbf{x}^k_{P_i} \\
&= \sum_{P_i\in\mathcal{J}^k} \mathbf{y}_{P_i}{}^T \mathbf{x}^k_{P_i} + \sum_{P_i\notin\mathcal{J}^k} \mathbf{y}_{P_i}{}^T \mathbf{x}^k_{P_i} = \mathbf{y}^T\mathbf{x}^k = 0.
\end{aligned}$$

Finally by noting that for all $P_i \in \mathcal{J}^k$ it holds that $0 \leq \widehat{\mathbf{x}}_{P_i} \leq C$, then by the convexity of the box constraints and since $\alpha^k \leq 1$, we obtain $0 \leq \mathbf{x}^{k+1} \leq C$ and this proves the feasibility of the sequence.

Note that $\forall P_i \in \mathcal{J}^k$ the direction $-\mathbf{d}^k_{P_i}$ is a feasible direction at $\widehat{x}^k_{P_i}$ hence by (12) we can write

$$-\left[\nabla_{P_i} f_{P_i}(\widehat{\mathbf{x}}^k_{P_i}, \mathbf{x}^k_{-P_i}) + \tau^k_i(\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i})\right]^T \mathbf{d}^k_{P_i} \geq 0. \tag{16}$$

Hence it holds that

$$\begin{aligned}
-\nabla_{P_i} f_{P_i}(\mathbf{x}^k_{P_i}, \mathbf{x}^k_{-P_i})^T \mathbf{d}^k_{P_i} = \left(Q_{P_iP_i}\mathbf{x}^k_{P_i} + \ell^k_{P_i}\right)^T (\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) = \\
\left(Q_{P_iP_i}\mathbf{x}^k_{P_i} + \ell^k_{P_i}\right)^T (\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) - (\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i})^T Q_{P_iP_i}(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) + \\
(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i})^T Q_{P_iP_i}(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) = \\
\left(Q_{P_iP_i}\widehat{\mathbf{x}}^k_{P_i} + \ell^k_{P_i}\right)^T (\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) + \\
(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i})^T Q_{P_iP_i}(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) = \\
-\nabla_{P_i} f_{P_i}(\widehat{\mathbf{x}}^k_{P_i}, \mathbf{x}^k_{-P_i})^T \mathbf{d}^k_{P_i} + \\
(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i})^T Q_{P_iP_i}(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) \overset{(16)}{\geq} \\
\tau^k_i \|\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\|^2 + \\
(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i})^T Q_{P_iP_i}(\mathbf{x}^k_{P_i} - \widehat{\mathbf{x}}^k_{P_i}) \geq \\
(\tau^k_i + \lambda^{Q_{P_iP_i}}_{\min})\|\mathbf{d}^k_{P_i}\|^2,
\end{aligned}$$

and then we have

$$\nabla_{P_i} f_{P_i}(\mathbf{x}^k_{P_i}, \mathbf{x}^k_{-P_i})^T \mathbf{d}^k_{P_i} \leq -(\tau^k_i + \lambda^{Q_{P_iP_i}}_{\min})\|\mathbf{d}^k_{P_i}\|^2. \tag{17}$$

We denote by $\rho^k = \min_{P_i \in \mathcal{J}^k}(\tau^k_i + \lambda^{Q_{P_iP_i}}_{\min}) > 0$. By assumption (ii) there exists $\rho > 0$ such that $\rho^k \geq \rho$ for all k.

From conditions (15) and (17) we can write

$$\mathbf{f}(\mathbf{x}^{k+1}) - \mathbf{f}(\mathbf{x}^k) \leq -\alpha^k \theta \rho \|\mathbf{d}^k\|^2, \tag{18}$$

therefore sequence $\{\mathbf{f}(\mathbf{x}^k)\}$ is decreasing. It is also bounded below so that it converges and

$$\lim_{k\to\infty}(\mathbf{f}(\mathbf{x}^{k+1}) - \mathbf{f}(\mathbf{x}^k)) = 0. \tag{19}$$

Let $\bar{\mathbf{x}}$ be a limit point of $\{\mathbf{x}^k\}$, at least one of such points exists being $\mathcal{F}$ compact. Since, by the compactness of $\mathcal{F}$ and by the continuity of f, $-\infty < \mathbf{f}(\bar{\mathbf{x}}) - \mathbf{f}(\mathbf{x}^0)$ for all $\mathbf{x}^0 \in \mathcal{F}$, then, by (18) and (19), we can write

$$\sum_{k=0}^{\infty} \alpha^k \|\mathbf{d}^k\|^2 < +\infty. \tag{20}$$

By condition (i) we can define an infinite subsequence $\{k\}_{\tilde{K}}$ made up of only descent iterations. Then, by (20), it follows that

$$\sum_{k=0,k\in\tilde{K}}^{\infty} \alpha^k \|\mathbf{d}^k\|^2 < +\infty. \tag{21}$$

A standard Armijo linesearch satisfying (15) produces at each iteration an $\alpha^k > 0$ in a finite number of steps (see [1]) and this ensures that

$$\sum_{k=0,k\in\tilde{K}}^{\infty} \alpha^k = +\infty. \tag{22}$$

By (21) and (22), we obtain

$$\liminf_{k\to\infty,k\in\tilde{K}} \|\mathbf{d}^k\| = 0.$$

Now since each $\mathcal{J}^k$ with $k \in \tilde{K}$ contains a descent block at $\mathbf{x}^k$, by (13) we can conclude that

$$\liminf_{k\to\infty,k\in\tilde{K}} S_{\texttt{MVP}}(\mathbf{x}^k) = 0,$$

and then, since $S_{\texttt{MVP}}(\mathbf{x}^k) \geq 0$ for all k, we can write

$$\liminf_{k\to\infty} S_{\texttt{MVP}}(\mathbf{x}^k) = 0. \tag{23}$$

Suppose by contradiction that $\limsup_{k\to\infty} S_{\texttt{MVP}}(\mathbf{x}^k) > 0$, then for any $\gamma > 0$ sufficiently small we would have $S_{\texttt{MVP}}(\mathbf{x}^k) > \gamma$ for infinitely many k and $S_{\texttt{MVP}}(\mathbf{x}^k) < \frac{\gamma}{2}$ for infinitely many k. Therefore, one can always find an infinite set of indices, say $\mathcal{N}$, having the following property: for any $n \in \mathcal{N}$, there exists an integer $i_n > n$ such that $S_{\texttt{MVP}}(\mathbf{x}^n) < \frac{\gamma}{2}$ and $S_{\texttt{MVP}}(\mathbf{x}^{i_n}) > \gamma$. Then it is easy to see that $\mathbf{x}^n \neq \mathbf{x}^{i_n}$ and then $\sum_{k=n}^{i_n-1} \alpha^k \|\mathbf{d}^k\| > 0$ for all $n \in \mathcal{N}$. And then

$$\liminf_{n\in\mathcal{N},n\to\infty} \sum_{k=n}^{i_n-1} \alpha^k \|\mathbf{d}^k\| > 0,$$

which is in contradiction with (20). Then we finally obtain (14). ■ □

Note that since $\mathbf{f}$ is quadratic, condition (15) can be guaranteed by using the exact minimization rule along direction $\mathbf{d}^k$:

$$\alpha^k := \max\left\{\min\left\{-\frac{\nabla\mathbf{f}(\mathbf{x}^k)^T\mathbf{d}^k}{\mathbf{d}^{kT}\mathbf{Q}\mathbf{d}^k},\ \bar{\alpha}^k\right\}, 0\right\}, \tag{24}$$

where

$$\bar{\alpha}^k := \min_{i\in\{1,\dots,n\}:\mathbf{d}_i^k\neq 0} \left\{ \bar{\alpha}_i^k = \left\{ \begin{array}{ll} (\mathbf{x}^k)_i & \text{if } \mathbf{d}_i^k < 0 \\ C - (\mathbf{x}^k)_i & \text{if } \mathbf{d}_i^k > 0 \end{array} \right\} \right.$$

Note that in this case it is not necessary to impose $\alpha^k \leq 1$ since the feasibility is guaranteed by construction.

If n is huge it may be cheaper to compute α^k in an alternative way in order to save costly function evaluations. In particular we propose a diminishing stepsize strategy for which we give two different convergence results based on slightly different hypotheses.

Theorem 4.3 *Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 3.1 where $\alpha^k \in (0,1]$ at* **S.5** *satisfies the following condition*

$$\alpha^k \to 0 \text{ and } \sum_{k=0}^{\infty} \alpha^k = +\infty. \tag{25}$$

Assume that for all iterations k

(i) *k is a descent iteration;*

(ii) *either $\tau_i^k \geq \underline{\tau} > 0$ or $Q_{P_iP_i} \succ 0$, for all $P_i \in \mathcal{J}^k$;*

Then either Algorithm 3.1 converges in a finite number of iterations to a solution of problem (1) *or $\{\mathbf{x}^k\}$ admits a limit point and* (14) *holds.*

Proof. Note that feasibility of the sequence $\{\mathbf{x}^k\}$ and inequality (17) hold, see proof of Theorem 4.2.

For any given $k \geq 0$ we can write (Descent Lemma [1]):

$$\mathbf{f}(\mathbf{x}^{k+1}) - \mathbf{f}(\mathbf{x}^k) \leq \alpha^k \nabla \mathbf{f}(\mathbf{x}^k)^T \mathbf{d}^k + \frac{(\alpha^k)^2 \lambda_{\max}^{\mathbf{Q}}}{2} \|\mathbf{d}^k\|^2. \tag{26}$$

By using (ii) and (17) we can rewrite inequality (26):

$$\mathbf{f}(\mathbf{x}^{k+1}) - \mathbf{f}(\mathbf{x}^k) \leq \alpha^k \left(-\rho + \frac{\alpha^k \lambda_{\max}^{\mathbf{Q}}}{2} \right) \|\mathbf{d}^k\|^2, \tag{27}$$

where $\rho > 0$ is the minimum among $\underline{\tau}$ and all the minimum eigenvalues of all the positive definite principal submatrices of $\mathbf{Q}$. Since, by (25), $\alpha^k \to 0$ it follows that there exist $\bar{\rho} > 0$ and $\bar{k}$ sufficiently large such that for all $k \geq \bar{k}$ inequality (27) implies:

$$\mathbf{f}(\mathbf{x}^{k+1}) - \mathbf{f}(\mathbf{x}^k) \leq -\alpha^k \bar{\rho} \|\mathbf{d}^k\|^2.$$

Since, as said in the proof of Theorem 4.2, $-\infty < \mathbf{f}(\bar{\mathbf{x}}) - \mathbf{f}(\mathbf{x}^{\bar{k}})$ for all $\mathbf{x}^{\bar{k}} \in \mathcal{F}$ and any limit point $\bar{\mathbf{x}}$ of $\{\mathbf{x}^k\}$, in a similar way we can write

$$\sum_{k=\bar{k}}^{\infty} \alpha^k \|\mathbf{d}^k\|^2 < +\infty. \tag{28}$$

By (25), $\sum_{k=\bar{k}}^{\infty} \alpha^k = +\infty$, and then we obtain

$$\liminf_{k\to\infty} \|\mathbf{d}^k\| = 0.$$

Now since each $\mathcal{J}^k$ contains a descent block at $\mathbf{x}^k$, by (13) we can conclude that

$$\liminf_{k\to\infty} S_{\texttt{MVP}}(\mathbf{x}^k) = 0,$$

and the thesis follows under the same reasoning of the proof of Theorem 4.2. ■ □

As stated in Theorem 4.3, a diminishing stepsize rule requires all iterations to be descent. In certain applications (e.g. when variables are randomly partitioned), it could be useful to relax this condition, requiring that only a subsequence of the iterations are descent, as well as for Theorem 4.2. This is formalized in the next theorem where we assume the additional mild hypothesis of monotonicity of sequence $\{\alpha^k\}$.

Theorem 4.4 *Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 3.1 where $\alpha^k \in (0, 1]$ at* **S.5** *satisfies* (25)*. Assume that for all iterations k*

(i) *a descent iteration $\tilde{k}$ with $k \le \tilde{k} \le k + L$ for a finite $L \ge 0$ is generated;*

(ii) *either $\tau_i^k \ge \underline{\tau} > 0$ or $Q_{P_iP_i} \succ 0$, for all $P_i \in \mathcal{J}^k$;*

(iii) *$\alpha^k \ge \alpha^{k+1}$;*

then either Algorithm 3.1 converges in a finite number of iterations to a solution of problem (1) *or $\{\mathbf{x}^k\}$ admits a limit point and* (14) *holds.*

Proof. Following the same reasoning of Theorem 4.3, inequality (28) holds. By condition (i) we can define an infinite subsequence $\{k\}_{\tilde{K}}$ containing only descent iterations. Then by (28) it follows that

$$\sum_{k=\bar{k}, k\in\tilde{K}}^{\infty} \alpha^k \|\mathbf{d}^k\|^2 < +\infty. \tag{29}$$

We can write the following chain of inequalities

$$+\infty = \sum_{k=\bar{k}+1}^{\infty} \sum_{h=0}^{L-1} \alpha^{L\cdot k+h} \leq L \sum_{k=\bar{k}+1}^{\infty} \alpha^{L\cdot k} \leq L \sum_{k=\bar{k},k\in\tilde{K}}^{\infty} \alpha^{k},$$

where the equality is due to (25), the first inequality holds by (iii) and the second inequality holds by (i) and (iii). Then the thesis follows from the same reasoning of the proof of Theorem 4.2. ■ □

5 Construction of the Partitions

To make the results stated in the previous section of practical interest, the major difficulty is to ensure that an iteration is descent in the sense that at least one descent block, according to Definition 4.1, is selected. Next lemma gives a relation between the steplenght produced optimizing over a generic block P_i and the one produced optimizing over any violating pair $(\bar{i}, \bar{j})$ belonging to P_i. This result will be useful in order to practically build a descent block and it is used in Theorem 5.2. In this section we use the simplified assumption that any principal submatrix of Q of order 2 is positive definite.

Lemma 5.1 *Assume that any principal submatrix of Q of order 2 is positive definite. Let $\mathbf{x}^k$ be a feasible point for problem (1) and let $(\bar{i}, \bar{j}) \in I_{up}(\mathbf{x}^k) \times I_{low}(\mathbf{x}^k)$. Suppose that a block $P_i \subseteq \{1, \ldots, n\}$ exists such that $(\bar{i}, \bar{j}) \subseteq P_i$. Let $\overline{\mathbf{x}}^k$ be the unique solution of*

$$\min_{\mathbf{x}_{(\bar{i},\bar{j})} \in \mathcal{F}_{(\bar{i},\bar{j})}} f_{(\bar{i},\bar{j})} \left(\mathbf{x}_{(\bar{i},\bar{j})}, \mathbf{x}^k_{\{1,\ldots,n\}\setminus(\bar{i},\bar{j})} \right). \tag{30}$$

Then there exists a scalar $\bar{\epsilon} > 0$ such that

$$\|\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\| \geq \bar{\epsilon} \|\overline{\mathbf{x}}^k - \mathbf{x}^k\|, \tag{31}$$

where $\widehat{\mathbf{x}}^k_{P_i}$ is a solution of problem (10).

Proof. First we note that

$$\overline{\mathbf{x}}^k = \mathbf{x}^k + \bar{\alpha} \mathbf{d}^{(\bar{i},\bar{j})}$$

with $\mathbf{d}^{(\bar{i},\bar{j})}$ defined in (5) and $\bar{\alpha}$ computed as in (8). For the sake of notation let us set $\overline{\mathbf{d}} = \mathbf{d}^{(\bar{i},\bar{j})}$.

Since $(\bar{i}, \bar{j}) \subseteq P_i$, two cases are possible: (a) $\widehat{\mathbf{x}}^k_{P_i} + \mu \overline{\mathbf{d}}_{P_i} \notin \mathcal{F}_{P_i}$ for all $\mu > 0$, or (b) $\mu > 0$ exists such that $\widehat{\mathbf{x}}^k_{P_i} + \mu \overline{\mathbf{d}}_{P_i} \in \mathcal{F}_{P_i}$, that is $\overline{\mathbf{d}}_{P_i}$ is a feasible direction at $\widehat{\mathbf{x}}^k_{P_i}$.

(a) By construction it holds that $\mathbf{y}_{P_i}^T \overline{\mathbf{d}}_{P_i} = 0$. Then it holds that for all $\mu > 0$:

$$\mathbf{y}_{P_i}^T(\widehat{\mathbf{x}}_{P_i}^k + \mu\overline{\mathbf{d}}_{P_i}) = \mathbf{y}_{P_i}^T\widehat{\mathbf{x}}_{P_i}^k = \mathbf{y}_{P_i}^T\mathbf{x}_{P_i}^k,$$

where last equality holds since $\widehat{\mathbf{x}}_{P_i}^k \in \mathcal{F}_{P_i}$. Therefore we can conclude that for all $\mu > 0$:

$$\widehat{\mathbf{x}}_{P_i}^k + \mu\overline{\mathbf{d}}_{P_i} \notin [0, C]^{|P_i|},$$

and then either $\widehat{x}_{\bar{i}}^k$ or $\widehat{x}_{\bar{j}}^k$ must be on a bound. In particular, supposing w.l.o.g. that is the component $\bar{i}$ the one on the bound, if $\overline{d}_{\bar{i}} > 0$ then $\widehat{x}_{\bar{i}}^k = C$ and then we can write

$$0 < \bar{\alpha}\overline{d}_{\bar{i}} = \overline{x}_{\bar{i}}^k - x_{\bar{i}}^k \leq C - x_{\bar{i}}^k = \widehat{x}_{\bar{i}}^k - x_{\bar{i}}^k;$$

otherwise $\overline{d}_{\bar{i}} < 0$ then $\widehat{x}_{\bar{i}}^k = 0$ and then

$$0 > \bar{\alpha}\overline{d}_{\bar{i}} = \overline{x}_{\bar{i}}^k - x_{\bar{i}}^k \geq 0 - x_{\bar{i}}^k = \widehat{x}_{\bar{i}}^k - x_{\bar{i}}^k.$$

In both cases it holds that

$$|\bar{\alpha}\overline{d}_{\bar{i}}| \leq |\widehat{x}_{\bar{i}}^k - x_{\bar{i}}^k|.$$

Therefore noting that $|\bar{\alpha}\overline{d}_{\bar{i}}| = |\bar{\alpha}|$ and that $|\bar{\alpha}|\sqrt{2} = \|\bar{\mathbf{x}}^k - \mathbf{x}^k\|$ we can conclude that

$$\|\widehat{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k\| \geq |\bar{\alpha}| = \frac{1}{\sqrt{2}}\|\overline{\mathbf{x}}^k - \mathbf{x}^k\|.$$

(b) Since $\overline{\mathbf{d}}_{P_i}$ is a feasible direction at $\widehat{\mathbf{x}}_{P_i}^k$ and since $\widehat{\mathbf{x}}_{P_i}^k$ is an optimal solution of problem (10) at $\mathbf{x}^k$, by (12), we can write

$$\left[\nabla_{P_i} f_{P_i}\left(\widehat{\mathbf{x}}_{P_i}^k, \mathbf{x}_{-P_i}^k\right) + \tau_i^k(\widehat{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k)\right]^T \overline{\mathbf{d}}_{P_i} \geq 0. \tag{32}$$

Since $\bar{\mathbf{x}}^k$ is a solution of (30), and being $-\overline{\mathbf{d}}$ a feasible direction for (30) at $\bar{\mathbf{x}}^k$, then, by the minimum principle and since $(\bar{i}, \bar{j}) \subseteq P_i$, we can write

$$\nabla_{P_i} f_{P_i}\left(\overline{\mathbf{x}}_{P_i}^k, \mathbf{x}_{-P_i}^k\right)^T \overline{\mathbf{d}}_{P_i} \leq 0.$$

And therefore by (32) we can write

$$\begin{aligned}\left[\nabla_{P_i} f_{P_i}\left(\widehat{\mathbf{x}}_{P_i}^k, \mathbf{x}_{-P_i}^k\right) + \tau_i^k(\widehat{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k)\right]^T \overline{\mathbf{d}}_{P_i} \geq \\ \nabla_{P_i} f_{P_i}\left(\overline{\mathbf{x}}_{P_i}^k, \mathbf{x}_{-P_i}^k\right)^T \overline{\mathbf{d}}_{P_i}.\end{aligned} \tag{33}$$

By assumptions, $\sigma > 0$ exists such that

$$\sigma\|\overline{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k\|^2 \leq (\overline{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k)^T Q_{P_iP_i}(\overline{\mathbf{x}}_{P_i}^k - \mathbf{x}_{P_i}^k)$$

$$= \left[\nabla_{P_i} f_{P_i}\left(\overline{\mathbf{x}}^k_{P_i}, \mathbf{x}^k_{-P_i}\right) - \nabla_{P_i} f_{P_i}\left(\mathbf{x}^k_{P_i}, \mathbf{x}^k_{-P_i}\right)\right]^T \bar{\alpha}\overline{\mathbf{d}}_{P_i}. \quad (34)$$

Then combining (33) and (34) we can write

$$\begin{aligned}
&\sigma\|\overline{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\|^2 \leq \tau^k_i(\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i})^T\bar{\alpha}\overline{\mathbf{d}}_{P_i} + \\
&\left[\nabla_{P_i} f_{P_i}\left(\widehat{\mathbf{x}}^k_{P_i}, \mathbf{x}^k_{-P_i}\right) - \nabla_{P_i} f_{P_i}\left(\mathbf{x}^k_{P_i}, \mathbf{x}^k_{-P_i}\right)\right]^T \bar{\alpha}\overline{\mathbf{d}}_{P_i} \leq \\
&(\tau^k_i + \|Q_{P_iP_i}\|)\|\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\|\|\bar{\alpha}\overline{\mathbf{d}}_{P_i}\| = \\
&(\tau^k_i + \lambda^{\mathbf{Q}}_{\max})\|\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\|\|\overline{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\|.
\end{aligned}$$

Therefore we obtain

$$\|\widehat{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\| \geq \frac{\sigma}{\tau^k_i + \lambda^{\mathbf{Q}}_{\max}}\|\overline{\mathbf{x}}^k_{P_i} - \mathbf{x}^k_{P_i}\| = \frac{\sigma}{\tau^k_i + \lambda^{\mathbf{Q}}_{\max}}\|\overline{\mathbf{x}}^k - \mathbf{x}^k\|,$$

and finally we have the proof.

■ □

Theorem 5.2 *Assume that any principal submatrix of Q of order 2 is positive definite. Let $\mathbf{x}^k$ be a feasible point for problem (1) and let $(\bar{i}, \bar{j})$ be a pair of indices such that $\bar{i} \in I_{up}(\mathbf{x}^k)$ and $\bar{j} \in I_{low}(\mathbf{x}^k)$. Suppose that a block $P_i \subseteq \{1, \ldots, n\}$ exists such that $(\bar{i}, \bar{j}) \subseteq P_i$. Let $\bar{\mathbf{x}}^k$ be the unique solution of problem (30) and suppose that $\widetilde{\epsilon} > 0$ exists such that*

$$\|\bar{\mathbf{x}}^k - \mathbf{x}^k\| \geq \widetilde{\epsilon} S_{MVP}(\mathbf{x}^k). \quad (35)$$

Then P_i is a descent block.

Proof. By Lemma 5.1 we know that (31) holds. Therefore by combining (31) and (35) we obtain the proof. ■ □

Theorem 5.2 shows that we can build a descent block at the cost of computing a pair that satisfies (35). Clearly the most violating pair does it, but it is easy to see that any pair that "sufficiently" violates KKT conditions can be used as well.

Now we give a further theoretical result which guarantees that at each iteration of Algorithm 3.1 at least one descent block can be built.

Theorem 5.3 *Let $\mathbf{x}^k$ be a feasible, but not optimal, point for problem (1) then at least one descent block $P_i \subseteq \{1, \ldots, n\}$ exists.*

Proof. By Proposition 2.2, if $\mathbf{x}^k$ is not optimal then $I_{up}(\mathbf{x}^k) \neq \emptyset$, $I_{low}(\mathbf{x}^k) \neq \emptyset$ and $S_{\texttt{MVP}}(\mathbf{x}^k) > 0$. Therefore $P_i = (i_{\texttt{MVP}}, j_{\texttt{MVP}})$ is a descent block. ■ □

6 Global Convergence in a Realistic Setting

So far we have proved that, under some suitable conditions, Algorithm 3.1 either converges in a finite number of iterations to a solution of problem (1) or the produced sequence $\{\mathbf{x}^k\}$ satisfies (14). However the fact that $S_{\texttt{MVP}}(\mathbf{x}^k)$ goes to zero is not enough to guarantee asymptotic convergence of Algorithm 3.1 to a solution of problem (1). Indeed, this is due to the discontinuous nature of indices sets I_{up} and I_{low} that enters the definition of $S_{\texttt{MVP}}(\mathbf{x}^k)$. Actually, this is a well known theoretical issue in decomposition methods for the SVM training problem. However, even in the case when the algorithm were proved to asymptotically converge to an optimal solution, the validity of a stopping criterion based on the KKT conditions must be verified [17]. A possible way to sorting out these theoretical issues is to use some theoretical tricks. For example by properly inserting some standard MVP iterations in the produced sequence $\{x^k\}$ [19] or by dealing with $\epsilon-$solutions [14]. All these theoretical efforts can be encompassed in a realistic numerical setting. Indeed all the papers discussing about decomposition methods rely on the fact that the indices sets I_{up} and I_{low} can be computed in exact arithmetic. In practice what it can actually be computed are the following $\epsilon-$perturbations of sets I_{up} and I_{low}

$$I^{\epsilon}_{up}(\mathbf{x}) := \{r \subseteq \{1,\dots,n\} :\ x_r \leq C-\epsilon,\ y_r = 1,\ \text{or } x_r \geq \epsilon,\ y_r = -1\},$$

$$I^{\epsilon}_{low}(\mathbf{x}) := \{r \subseteq \{1,\dots,n\} :\ x_r \leq C-\epsilon,\ y_r = -1,\ \text{or } x_r \geq \epsilon,\ y_r = 1\},$$

with $\epsilon > 0$. Consequently we can define at a feasible point $\mathbf{x}$ the following quantities

$$m^{\epsilon}(\mathbf{x}) = \max_{r \in I^{\epsilon}_{up}(\mathbf{x})} -\frac{\nabla f(\mathbf{x})_r}{y_r}, \qquad M^{\epsilon}(\mathbf{x}) = \min_{r \in I^{\epsilon}_{low}(\mathbf{x})} -\frac{\nabla f(\mathbf{x})_r}{y_r}.$$

As a matter-of-fact an effective optimality condition which can be used is

$$m^{\epsilon}(\mathbf{x}^k) \leq M^{\epsilon}(\mathbf{x}^k) + \eta, \tag{36}$$

where $\eta > 0$ is a given tolerance. Note that any asymptotically convergent decomposition algorithm can actually converge only to a point satisfying (36), rather than (3).

It is easy to see that $I^{\epsilon}_{up}(\mathbf{x}) \subseteq I_{up}(\mathbf{x})$ and $I^{\epsilon}_{low}(\mathbf{x}) \subseteq I_{low}(\mathbf{x})$ for all $\epsilon > 0$. Furthemore in [22], it has been proved the following result.

Proposition 6.1 *Let $\{x^k\}$ be a sequence of feasible points converging to a point $\bar{x} \in \mathcal{F}$. Then, there exists a scalar $\bar{\epsilon} > 0$ (depending only on $\bar{x}$) such that for every $\epsilon \in (0, \bar{\epsilon}]$ there exists an index $\bar{k} = k_\epsilon$ for which*

$$I^{\epsilon}_{up}(\mathbf{x}^k) := I_{up}(\mathbf{x}^k) \qquad \text{and} \qquad I^{\epsilon}_{low}(\mathbf{x}^k) := I_{low}(\mathbf{x}^k) \qquad \textit{for all } k \geq \bar{k}.$$

This proposition allows to state that for k sufficiently large and ϵ sufficiently small using index sets $I^{\epsilon}_{up}(\mathbf{x})$ and $I^{\epsilon}_{low}(\mathbf{x})$ is equivalent to use the exact ones I_{up} and I_{low} and we have that

$$m^{\epsilon}(\mathbf{x}) = m(\mathbf{x}) \qquad \text{and} \qquad M^{\epsilon}(\mathbf{x}) = M(\mathbf{x}),$$

so that, for any $\epsilon \in (0, \bar{\epsilon}]$ and $k \geq \bar{k}$, (36) reduces to the concept of η-optimal solution introduced in [14]. However this is not true far from a solution and/or for a wrong value of ϵ, being $\bar{\epsilon}$ unknown. Reducing ϵ to the machine precision ϵ_{mach} is the best that we can do in a numerical implementation, so that one can argue that for $\epsilon = \epsilon_{\text{mach}}$ if $I^{\epsilon}_{up}(\mathbf{x}) = \emptyset$ or $I^{\epsilon}_{low}(\mathbf{x}) = \emptyset$, a solution has been reached within the possible tolerance.

Given a point $\mathbf{x}^k \in \mathcal{F}$, we consider the MVP $\epsilon-$step $S^{\epsilon}_{\texttt{MVP}}(\mathbf{x}^k)$ obtained by using $I^{\epsilon}_{up}(\mathbf{x}^k)$ and $I^{\epsilon}_{low}(\mathbf{x}^k)$ instead of $I_{up}(\mathbf{x}^k)$ and $I_{low}(\mathbf{x}^k)$. As a consequence of the definition itself, for any MVP $\epsilon-$direction $\mathbf{d}^k_{\texttt{MVP},\epsilon}$ we get that the feasible $\epsilon-$stepsize $\bar{\beta}^k_{\epsilon}$ defined as in (6) remains bounded from zero by ϵ.

It is easy to see that all results stated so far for Algorithm 3.1 are still valid if we consider the $\epsilon-$definition $S^{\epsilon}_{\texttt{MVP}}(\mathbf{x}^k)$ rather than $S_{\texttt{MVP}}(\mathbf{x}^k)$. Furthemore we have the following result, that fill the gap of convergence.

Theorem 6.2 *Let $\epsilon > 0$ and $\eta > 0$ be given. Let $\{\mathbf{x}^k\}$ be a sequence of feasible points such that $I^{\epsilon}_{up}(\mathbf{x}^k) \neq \emptyset$, $I^{\epsilon}_{low}(\mathbf{x}^k) \neq \emptyset$ and*

$$\lim_{k\to\infty} S^{\epsilon}_{MVP}(\mathbf{x}^k) = 0.$$

Then $\bar{k} > 0$ exists such that, for all $k \geq \bar{k}$, $\mathbf{x}^k$ satisfies (36).

Proof. By definition of $S^{\epsilon}_{\texttt{MVP}}(\mathbf{x}^k)$ we get

$$0 = \lim_{k\to\infty} S^{\epsilon}_{\texttt{MVP}}(\mathbf{x}^k) = \sqrt{2} \lim_{k\to\infty} |\alpha^k_{\texttt{MVP},\epsilon}|. \tag{37}$$

Since by construction $\bar{\beta}^k_{\epsilon} \geq \epsilon$, by (4) we get that (37) implies that $\bar{k} > 0$ exists such that, for all $k \geq \bar{k}$, we have $-\nabla\mathbf{f}(\mathbf{x}^k)^T\mathbf{d}^k_{\texttt{MVP},\epsilon} \leq \eta$, which implies (36). ■□

7 Practical Algorithmic Realizations

Algorithm 3.1 includes a vast amount of specific strategies that may vary according to several implementation choices. Various alternative may be related to the blocks dimension, the blocks composition, the blocks selection, the way to enforce convergence conditions and the methods used to solve the

subproblems. Different algorithms can be designed exploiting these degrees of freedom. In this section we discuss about some possible alternatives and we suggest some practical implementations of Algorithm 3.1.

The dimension of the blocks is a key factor for the training performances. It influences the way in which the subproblems can be solved so it should be carefully determined according to the dataset nature. Mainly we can consider two opposite strategies: blocks of minimal dimension (i.e. SMO-type methods) and higher dimensional blocks. In SMO-type methods we can take advantage of the fact that, for each block, the subproblem can be solved analytically. On the other hand each SMO-block may yield a small decrease of the objective function and slow identification of the support vectors, so that to get fast convergence the simultaneous optimization of a great number of SMO-blocks may be needed when dealing with high dimensional instances. This choice could be well suited for an architecture composed of a great amount of simple processing units, like that of the recent Graphic Processing Unit (GPU). Higher dimensional blocks require the solution of the subproblems by means of some optimization procedure hence the solution of each block may need greater time consumption. On the other hand the decrease of the objective function and the identification of the support vectors may be faster so that less iterations should be needed. Hence this choice may be suitable whenever powerful but, generally, not many processing units are available. Algorithm 3.1 encompasses also the possibility of considering huge blocks assigned at each processor and using a decomposition method to solve the corresponding huge subproblems. This strategy essentially consists in iteratively splitting the original SVM instance into smaller SVMs, distributing them to available parallel processes and then gathering their solutions points in order to properly define the new iterate.

An efficient rule to partition the variables into blocks is crucial for the rate of convergence of the algorithm. A lot of heurisitc methods (see e.g. [13, 15, 29]) have been studied to obtain efficient rules for constructing subproblems that can guarantee a fast decrease of the objective function (e.g. by determining the most violating pair). Such methods usaully make use of first order information, implying that the gradient should be partially or entirely updated at each iteration, and this could be overwhelming in a huge dimensional framework. However practical implementations with a (partially) random composition of the blocks could be considered.

Once we have determined a blocks partition of the whole set of variables, only a subset of the resulting subproblems may, in general, be involved in the optimization process. Indeed, we may further restrict the blocks used to update the current iterate by determining a subset $\mathcal{J}^k$. We need to keep in mind that the main computational burden is due to the gradient update. In-

deed at each iteration, the gradient update must be performed by computing the columns of $\mathbf{Q}$ related only to those variables that are chosen to be in the selected blocks $\mathcal{J}^k$. Hence the choice of $\mathcal{J}^k$ may take into account both the decrease of the objective function and the computational effort for updating the gradient. A minimal threshold on the percentage of objective function decrease associated to each block, with respect to the cumulative decrease of every block, could be a possible discriminant for a blocks selection rule in order to avoid useless computations.

The practical effectiveness of the algorithm is highly related to the way to enforce convergence conditions stated so far. If, for example, we do not take into account the rule of taking at least one descent block every L iterations, we could take at each iteration the same partition of variables. This short-sighted strategy would be totally ineffective, since, in this case, the algorithm would lead only to an equilibrium of the generalized Nash equilibrium problem (see e.g. [6, 7, 8]) in which the players solve the fixed subproblems, but not to a solution of the original SVM, see the following example.

Example 7.1 *Let us consider a dual problem with four variables and in which* $Q = I$, $\mathbf{y} = (1\ 1\ -1\ -1)^T$ *and* $C = 1$. *The unique solution of this problem is* $\mathbf{x}^* = (1\ 1\ 1\ 1)^T$. *Let* $\mathbf{x}^k = (0\ 0\ 0\ 0)^T$ *and suppose to consider a partition of two blocks:* $\mathcal{P} = \{(1,2),(3,4)\}$. *Then the best responses are* $\widehat{\mathbf{x}}^k_{(1,2)} = (0\ 0)^T$ *and* $\widehat{\mathbf{x}}^k_{(3,4)} = (0\ 0)^T$, *but this implies that* $\mathbf{x}^{k+1} = \mathbf{x}^k$. *Therefore if we do not modify* $\mathcal{P}$ *we will never move from the origin, and then will never converge to* $\mathbf{x}^*$. *However, being the origin a fixed point for the best responses of the two processes, it is, by definition, a Nash equilibrium for the game involving these two players.*

Regarding the choice of the steplenght α^k we can use two different strategies: a linesearch or a diminishing stepsize rule. In the first case, as showed before, we can use the exact minimization formula (24). In the second case a simple rule could be $\alpha^k = \frac{1}{k^\xi}$, with $\xi \in (0,1]$, but different choices are also possible. Although preliminary tests showed that the exact minimization is more effective than any other choice, the diminishing stepsize strategy, besides being easy to implement, requires much less computations and this could be of great practical interest for high dimensional instances (training set with many samples and many dense features).

As mentioned above, when the dimension of the blocks is more than two, an optimization algorithm is needed to obtain a solution of the subproblems. Due to its particular structure (convex quadratic objective function over a polyhedron), a lot of exact or approximate methods can be applied for the solution of problem (10). We simply point out that, whenever the dimension of the blocks is so big that each block can be considered a sort of smaller

SVM, a slight modification of any efficient software for the sequential training of SVMs (like LIBSVM) could be used to perform a single optimization step.

As a matter of example, we propose a SMO-type parallel scheme derived from Algorithm 3.1 for which we developed two matlab prototypes. This realization, that we call PARSMO, is based on using a partition $\mathcal{P}$ of minimal dimension blocks thus performing multiple SMO steps simultaneously in order to build the search direction $\mathbf{d}^k$. Each SMO step is assigned to a parallel process that analytically solves a two-dimensional subproblem. Thus the computational effort of each processor is very light and communications must be very fast thus being suitable for a multicore environment.

Algorithm 7.2 PARSMO

Initialization Set $\mathbf{x}^0 = 0$, $\nabla \mathbf{f}^0 = -\mathbf{e}$, $q \geq 1$, $\epsilon > 0$, $\eta > 0$ and $k = 0$.
Select

$$(i_1, j_1) \in I_{up}^{\epsilon,MVP}(\mathbf{x}^k) \times I_{low}^{\epsilon,MVP}(\mathbf{x}^k).$$

Do while $\left(-\nabla \mathbf{f}_{i_1}^k y_{i_1} + \nabla \mathbf{f}_{j_1}^k y_{j_1} \geq \eta\right)$

S.1 (Blocks definition)

Choose $(q-1)$ pairs $\{(i_2, j_2), (i_3, j_3), \ldots, (i_q, i_q)\}$.
Set $\mathcal{J}^k = \{(i_1, j_1), (i_2, j_2), \ldots, (i_q, i_q)\}$.

S.2 (Parallel computation)

For each pair $(i_h, j_h) \in \mathcal{J}^k$ compute in parallel:

1. kernel columns $\mathbf{Q}_{*i_h}$ and $\mathbf{Q}_{*j_h}$ (if not available in the cache),
2. $t_h \mathbf{d}^{(i_h j_h)}$ with $\mathbf{d}^{(i_h j_h)}$ defined as in (5) and t_h as in (8).

S.3 (Direction) $\mathbf{d}^k = \displaystyle\sum_{(i_h, j_h) \in \mathcal{J}^k} t_h \mathbf{d}^{(i_h j_h)}$

S.4 (Stepsize) Compute the steplenght α^k as in (24).

S.5 (Update)

Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k$.
Set $\nabla \mathbf{f}^{k+1} = \nabla \mathbf{f}^k + \alpha^k \displaystyle\sum_{h=1}^{q} t_h \left(d_{i_h}^k \mathbf{Q}_{*i_h} + d_{j_h}^k \mathbf{Q}_{*j_h}\right)$.
Set $k = k + 1$.
Select

$$(i_1, j_1) \in I_{up}^{\epsilon,MVP}(\mathbf{x}^k) \times I_{low}^{\epsilon,MVP}(\mathbf{x}^k).$$

End While

Return $\mathbf{x}^k$.

Starting from the feasible null vector $\mathbf{x}^0 = 0$, which is a well known suitable choice for SVM training algorithms beacuse allows to initialize the gradient $\nabla \mathbf{f}^0$ to $-\mathbf{e}$, the algorithm selects at each iteration the most violating pair $(i_1, j_1) \in I_{up}^{\epsilon,MVP}(\mathbf{x}^k) \times I_{low}^{\epsilon,MVP}(\mathbf{x}^k)$ and further $(q-1)$ pairs that all together make up $\mathcal{J}^k$. Search direction $\mathbf{d}^k$ at **S.3** is obtained by analytically com-

puting stepsise t_h for the q subproblems of type (10) related to the pairs in $\mathcal{J}^k$. Direction $\mathbf{d}^k$ is simply computed by summing up all the SMO steps; it has $\|\mathbf{d}^k\|_0 = 2q$ with $q \geq 1$ which depends on the number of parallel processes that we want to activate. Finally at Step 4 steplenght α^k that exactly minimizes the objective function along $\mathbf{d}^k$ is obtained by (24); note that this step requires no further kernel evaluations. The same holds for the gradient update. Of course as standard in SVM algorithms, a caching strategy can be exploited to limit the computational burden due to the evaluation of kernel columns.

In PARSMO it remains to specify how to select pairs forming blocks $\mathcal{J}^k$. Since the most expensive computational burden is due to the calculation of kernel columns, we want to analyse the impact of a massive use of the cache with respect to a standard one. Indeed we propose two different matlab implementations of PARSMO that use a cache strategy in two different ways. We would compare the performance with a standard sequential MVP implementation in order to analyze possible advantages of the PARSMO scheme.

In the first implementation, the $q-1$ pairs, in addition to a MVP, are selected by choosing those pairs that most violate the first order optimality condition, like in the SVM$^{\text{light}}$ algorithm [13]. Hence we select q pairs $(i_h, j_h) \in I^{\epsilon}_{up}(\mathbf{x}^k) \times I^{\epsilon}_{low}(\mathbf{x}^k)$ sequentially so that

$$-y_{i_1}\nabla f(\mathbf{x}^k)_{i_1} \geq -y_{i_2}\nabla f(\mathbf{x}^k)_{i_2} \geq \cdots \geq -y_{i_q}\nabla f(\mathbf{x}^k)_{i_q},$$

and

$$-y_{j_1}\nabla f(\alpha^k)_{j_1} \leq -y_{j_2}\nabla f(\mathbf{x}^k)_{j_2} \leq \cdots \leq -y_{j_q}\nabla f(\mathbf{x}^k)_{j_q}.$$

In this case, although we can use a standard caching strategy, we cannot control the number of kernel columns evaluations at each iteration that in the worst case can be up to $2q$. The computation of kernel columns $\mathbf{Q}_{*i_h}$ and $\mathbf{Q}_{*j_h}, \forall h \in \{1, \ldots, q\}$, can be performed in parallel by the processors empowered to solve the subproblems. In this case the number of kernel evaluation per iteration would be of course greater than those of a standard MVP, but the overall number of iterations may decrease. Thus we keep the advantages of performing simple analytic optimization, as in SMO methods, whilst moving $2q$ components at the time, as in SVM^{light}. We note that reconstruction of the overall gradient $\nabla \mathbf{f}^{k+1}$ can be parallelized among the q processors and requires a synchronization step to take into account stepsize α^k. Thus the CPU-time needed is essentially equivalent to a gradient update of a single SMO step. In this approach the transmission time among processors may be quite significant and this strictly depends on the parallel architecture. We refer to this implementation as PARSMO-1.

The second implementation selects the $q-1$ pairs, in addition to a MVP $(i_1, j_1) \in I^{\epsilon}_{up}(\mathbf{x}^k) \times I^{\epsilon}_{low}(\mathbf{x}^k)$, exclusively among the indices of the columns currently available in the cache. To be more precise, let $\mathcal{C}$ be the index set of the kernel columns available in the cache. The $q-1$ pairs (i_h, j_h) in $\mathcal{J}^k$ are selected following the same SVM^{light} rule described above for PARSMO-1, but restricted to the index sets $\mathcal{C} \cap I^{\epsilon}_{up}(\mathbf{x}^k) \times \mathcal{C} \cap I^{\epsilon}_{low}(\mathbf{x}^k)$. We refer to it as PARSMO-2. In this case the number of kernel evaluation per iteration is at most two as in a standard MVP implementation. The rationale of this version is to improve the performances of a classical MVP algorithm by using simultaneous multiple SMO optimizations without increasing the amount of kernel evaluations.

In order to have a flavour of the potentiality of these two parallel strategies, we performed some simple matlab experiments for the two versions PARSMO-1 and PARSMO-2. All experiments have been carried out on a 64-bit intel-Core i7 CPU 870 2.93Ghz $\times$ 8. Both PARSMO-1 and PARSMO-2 make use of a standard caching strategy, see [3], with a cache memory of 500 columns. We perform experiments with $q = 1, 2, 4$ and 8 parallel processes. Clearly the case with $q = 1$ corresponds to a classical MVP algorithm with a standard caching strategy. It is worth noting that to preserve the good numerical behavior of PARSMO-1 and PARSMO-2, it is necessary the use of the "gathering" steplenght α^k. In fact, further tests, not reported here, showed that by removing the use of α^k oscillatory and divergence phenomena may occur when using multiple parallel processes. This enforce the practical relevance of our theoretical analysis.

The major aim of the experiments in this preliminary contest is to highlight the benefits of simultaneously moving along multiple SMO directions. We tested both PARSMO-1 and PARSMO-2 on six benchmark problems available at the LIBSVM site `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`, using a standard setting for the parameters ($C = 1$, gaussian parameter $\gamma = 1/\#\text{features}$), see Table 1.

name	#features	#training data	kernel type
a9a	123	32561	gaussian
gisette scale	5000	6000	linear
cod-rna	8	59535	gaussian
real-sim	20958	72309	linear
rcv1	47236	20242	linear
w8a	300	49749	linear

Table 1: Training problems description.

To evaluate the behavior of the algorithms we consider the "relative error"

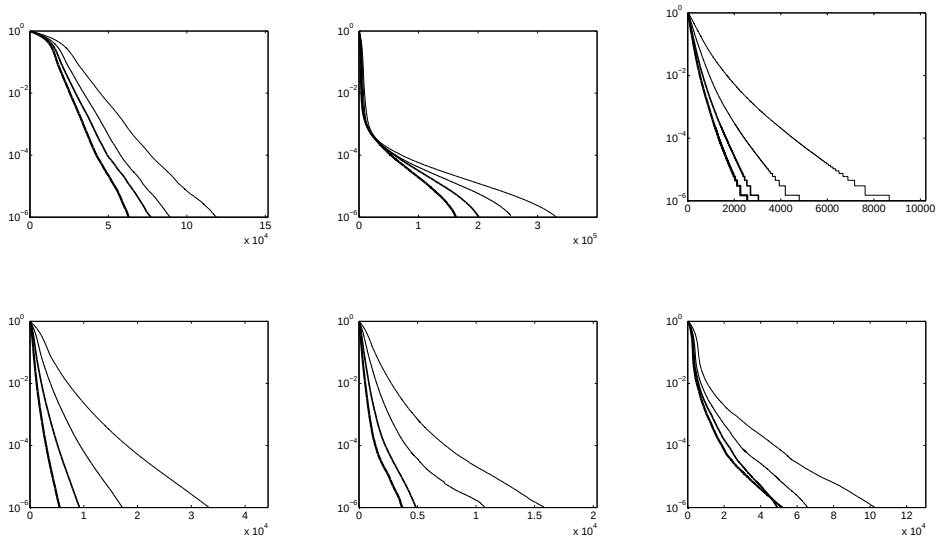
(RE) as

$$RE = \frac{|f^* - f|}{|f^*|},$$

where f^* is the optimal known value of the objective function. As regards PARSMO-1, for each problem we plot the RE versus

i) the number of iterations (see Figure 1);

ii) the number of kernel evaluations per process, which is obtained by dividing the total number of kernel evaluations by the number of parallel processes involved (see Figure 2).



real-sim rcv1 w8a

$q = 1$ $q = 2$ $q = 4$ $q = 8$

Figure 1: PARSMO-1: Relative Error versus iterations.

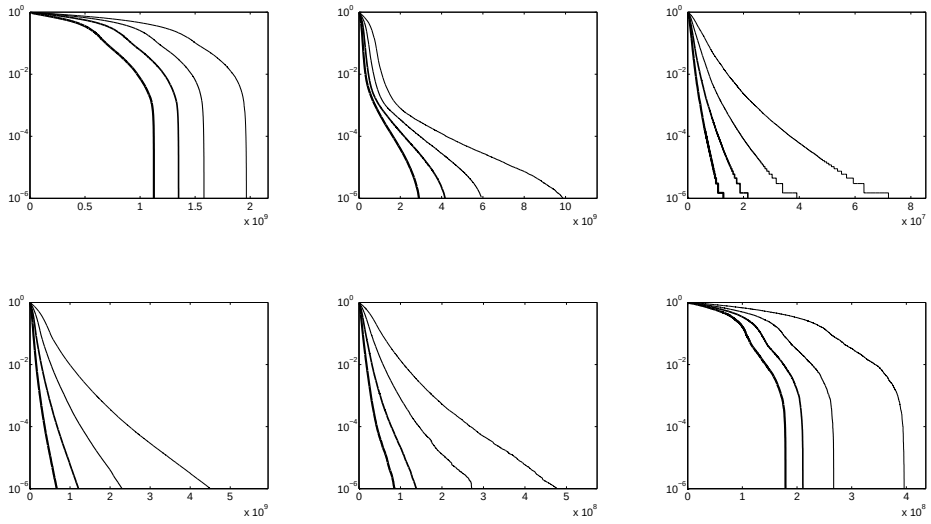


Figure 2: PARSMO-1: Relative Error versus kernel evaluations per process.

Our results show that the larger q is, the steeper the RE decrease is. This emphasizes the positive effect of moving along multiple SMO directions at a time.

As regards PARSMO-2, we note that, except for the MVP pair which can require the computation of the kernel columns $\mathbf{Q}_{*i_1}$ and $\mathbf{Q}_{*j_1}$, each SMO process computes only the analytical solution of the two-dimensional subproblem, since kernel columns are already available in the cache. Thus, PARSMO-2 may produce a cpu time saving even by running the algorithm in a sequential fashion. In order to show the cheapness of its tasks, in Figure 3 we plot RE versus the CPU-time consumed.

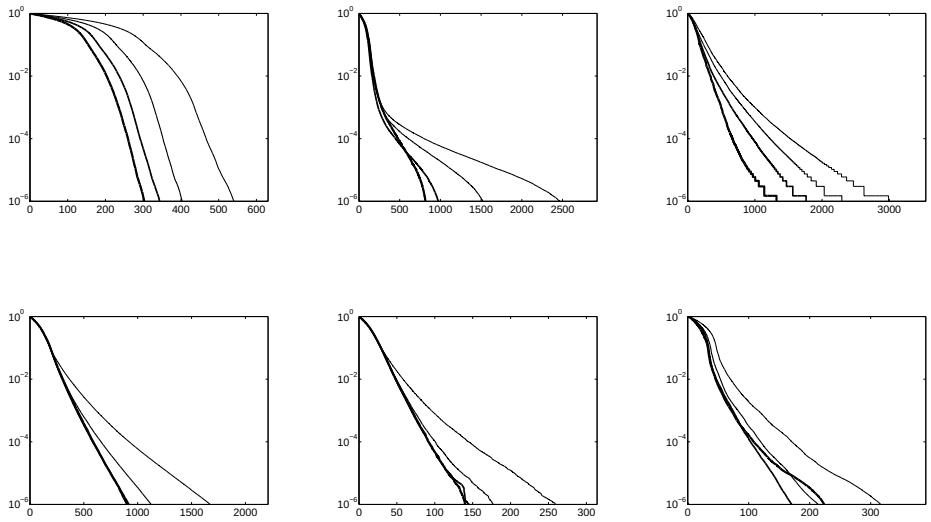


Figure 3: PARSMO-2: Relative Error versus (sequential) CPU-time.

PARSMO-2 with $q > 1$ seems to be faster than a classical MVP algorithm. This is due the use of multiple search directions without suffering from an increase of time consuming kernel evaluations or from the need of iterative solutions of larger quadratic subproblems. It is important to outline that PARSMO-2 achieves its good performances by combining a convergent parallel structure with an efficient sequential implementation, and it seems to be useful also in a single-core environment.

Acknowledgment

The authors thank Prof. Marco Sciandrone (Dipartimento di Ingegneria dell'Informazione, Universit di Firenze) for fruitful discussions and suggestions that improved significantly the paper.

References

[1] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[2] L. J. Cao, S. S. Keerthi, C.-J. Ong, J. Q. Zhang, and H. P. Lee. Parallel sequential minimal optimization for the training of support vector machines. *IEEE Trans. Neural Netw.*, 17(4):1039–1049, July 2006.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Tech.*, 2:27:1–27:27, 2011.

[4] P.-H. Chen, R.-E. Fan, and C.-J. Lin. A study on SMO-type decomposition methods for Support Vector Machines. *IEEE Trans. Neural Netw.*, 17(4):893–908, 2006.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[6] A. Dreves, F. Facchinei, C. Kanzow, and S. Sagratella. On the solution of the KKT conditions of generalized Nash equilibrium problems. *SIAM J. Optim.*, 21(3):1082–1108, 2011.

[7] F. Facchinei and C. Kanzow. Generalized Nash equilibrium problems. *4OR*, 5(3):173–210, 2007.

[8] F. Facchinei and S. Sagratella. On the computation of all solutions of jointly convex generalized Nash equilibrium problems. *Optim. Lett.*, 5(3):531–547, 2011.

[9] F. Facchinei, G. Scutari, and S. Sagratella. Parallel selective algorithms for nonconvex big data optimization. *IEEE Trans. Signal Process.*, 63(7):1874–1889, April 2015.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[11] M. C. Ferris and T. S. Munson. Interior-point methods for massive support vector machines. *SIAM J. Optim.*, 13(3):783–804, 2002.

[12] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *Advances in Neural Information Processing Systems*, pages 521–528. 2004.

[13] T. Joachims. Making large scale SVM learning practical. In C.B.B. Schölkopf, C.J.C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[14] S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Mach. Learn.*, 46(1-3):351–360, 2002.

[15] C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Trans. Neural Netw.*, 12:1288–1298, 2001.

[16] C.-J. Lin. Asymptotic convergence of an SMO algorithm without any assumptions. *IEEE Trans. Neural Netw.*, 13:248–250, 2002.

[17] C.-J. Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Trans. Neural Netw.*, 13(5):1045–1052, 2002.

[18] G. Liuzzi, L. Palagi, and M. Piacentini. On the convergence of a jacobi-type algorithm for singly linearly-constrained problems subject to simple bounds. *Optim. Lett.*, 5(2):347–362, 2011.

[19] S. Lucidi, L. Palagi, A. Risi, and M. Sciandrone. A convergent hybrid decomposition algorithm model for SVM training. *IEEE Trans. Neural Netw.*, 20(6):1055–1060, 2009.

[20] L. Palagi and M. Sciandrone. On the convergence of a modified version of SVM^{light} algorithm. *Optim. Methods Softw.*, 20(2-3):317–334, 2005.

[21] J. C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[22] A. Risi. *Convergent decomposition methods for Support Vector Machines*. PhD thesis, Sapienza University of Rome, 2008.

[23] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[24] P. Tseng and S. Yun. A coordinate gradient descent method for linearly constrained smooth optimization and support vector machines training. *Comput. Optim. Appl.*, 47(2):179–206, 2010.

[25] J. Yang. An improved cascade SVM training algorithm with crossed feedbacks. In *Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums on*, volume 2, pages 735–738, June 2006.

[26] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Comput.*, 29(4):535 – 551, 2003.

[27] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *J. Mach. Learn. Res.*, 7:1467–1492, 2006.

[28] J.-P. Zhang, Z.-W. Li, and J. Yang. A parallel SVM training algorithm on large-scale classification problems. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 3, pages 1637–1641 Vol. 3, Aug 2005.

[29] G. Zoutendijk. *Methods of Feasible Directions: a Study in Linear and Non-Linear Programming*. Elsevier, 1960.