Decomposition Methods for Linear Support Vector Machines

Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, and Chih-Jen Lin
Department of Computer Science and
Information Engineering
National Taiwan University
Taipei 106, Taiwan
cjlin@csie.ntu.edu.tw

Abstract We explain that decomposition methods, in particular, SMO-type algorithms, are not suitable for linear SVMs if there are more data than attributes. To remedy this difficulty, we consider a recent result by Keerthi and Lin (2003) that for an SVM with data not linearly separable, after C is large enough, the dual solutions are at similar faces. Motivated by this property, we show that alpha seeding is extremely useful for solving a sequence of linear SVMs. It largely reduces the number of decomposition iterations to the point that solving many linear SVMs requires less time than the original decomposition method for one single SVM. We also conduct comparisons with other methods which are efficient for linear SVMs, and demonstrate the effectiveness of the proposed approach for helping the model selection.

1 Introduction

Solving linear and non-linear support vector machines (SVM) have been considered two different tasks. For linear SVM without too many attributes in data instances, people have been able to train millions of data (e.g. (Mangasarian and Musicant 2000)); but for other types of problems, in particular, non-linear SVMs, the requirement of huge memory as well as computational time has prohibited us from solving very large problems. Currently, the decomposition method, a specially designed optimization procedure, is one of the main tools for non-linear SVMs. In this paper, we show that existing decomposition methods, in particular SMO-type algorithms, are not suitable for linear SVMs. To remedy this difficulty, motivating from Theorem 3 of (Keerthi and Lin 2003), we develop effective strategies so that decomposition methods become efficient for solving linear SVMs.

First, we briefly describe linear and non-linear SVMs. Given training vectors $x_i \in R^n, i = 1, \ldots, l$, in two classes, and a vector $y \in R^l$ such that $y_i \in \{1, -1\}$, the standard SVM formulation (Cortes and Vapnik 1995) is as follows:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{1.1}$$

If $\phi(x) = x$, usually we say (1.1) is the form of a linear SVM. On the other hand, if ϕ maps x to a higher dimensional space, (1.1) a non-linear SVM.

For a non-linear SVM, the number of variables depends on the size of w and can be very large (even infinite), so people solve the following dual form:

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \tag{1.2}$$

$$\text{subject to} \quad y^T\alpha = 0, \tag{1.3}$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, l,$$

where Q is an $l \times l$ positive semi-definite matrix with $Q_{ij} = y_i y_j \phi(x_i)^T\phi(x_j)$, e is the vector of all ones, and $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel function. (1.2) is solvable because its number of variables is the size of the training set, independent of the dimensionality of $\phi(x)$.

The primal and dual relation shows

$$w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \tag{1.4}$$

so

$$\text{sgn}(w^T\phi(x) + b) = \text{sgn}(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b)$$

is the decision function.

Unfortunately, for large training set, Q becomes such a huge dense matrix that traditional optimization methods cannot be directly applied. Currently, some specially designed approach such as decomposition methods (Osuna, Freund, and

Girosi 1997; Joachims 1998; Platt 1998) and finding the nearest points of two convex hulls (Keerthi, Shevade, Bhattacharyya, and Murthy 2000) are major ways of solving (1.2).

On the other hand, for linear SVMs, if $n \ll l$, w is not a huge vector variable so (1.1) can be solved by many regular optimization methods. As at the optimal solution $\xi_i = \max(0, 1 - y_i(w^T x_i + b))$, in a sense we mainly have to find out w and b. Therefore, if the number of attributes n is small, there are not many main variables w and b in (1.1) no matter how large the training set is. Currently, on a normal computer, people have been able to train a linear SVM with millions of data (e.g. (Mangasarian and Musicant 2000)); but for a non-linear SVM with much fewer data, we already need more computational time as well as computer memory.

Therefore, it is natural to ask whether in an SVM software linear and non-linear SVMs should be treated differently and solved by two methods. It is also interesting to see how capable non-linear SVM methods (e.g. decomposition methods) are for linear SVMs. Here, by linear SVMs we mean those with $n < l$. If $n \geq l$, the dual form (1.2) has fewer variables than w of the primal, a situation similar to nonlinear SVMs. As the rank of Q is less than or (usually) equal to $\min(n, l)$, the linear SVMs we are interested in here are those with low-ranked Q.

Recently, in many situations, linear and non-linear SVMs are considered together. Some approaches (Lee and Mangasarian 2001; Fine and Scheinberg 2001) approximate non-linear SVMs by different problems which are in the form of linear SVMs (Lin and Lin 2003; Lin 2002) with $n \ll l$. In addition, for non-linear SVM model selection with Gaussian kernel, (Keerthi and Lin 2003) proposed an efficient method which has to conduct linear SVMs model selection first (i.e. linear SVMs with different C). Therefore, it is important to discuss optimization methods for linear and non-linear SVMs at the same time.

In this paper, we focus on decomposition methods. In Section 2, we show that existing decomposition methods are inefficient for training linear SVMs. Section 3 presents our new strategy of training linear SVMs via decomposition methods, which is hundred or thousand times faster. The proposed method is compared with existing linear SVM methods in Section 4. We then, in Section 5, apply the

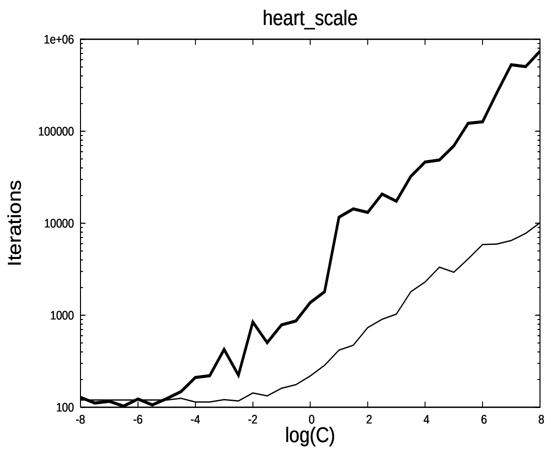

Figure 1: Number of decomposition iterations for solving SVMs with linear (the thick line) and RBF (the thin line) kernel.

new implementation to solve a sequence of linear SVMs required for the model selection method in (Keerthi and Lin 2003). Final discussion and concluding remarks are in Section 7.

2 Existing Decomposition Methods for Linear SVMs with $n \ll l$

The decomposition method is an iterative procedure. In each iteration, the index set of variables is separated to two sets B and N, where B is the working set. Then in that iteration variables corresponding to N are fixed while a sub-problem on variables corresponding to B is minimized. If q is the size of the working set B, in each iteration, only q columns of the Hessian matrix Q are required. They can be calculated and stored in the computer memory when needed. Thus, unlike regular optimization methods which usually require the access of the whole Q, here, the memory problem is solved. Clearly, decomposition methods are specially designed for nonlinear SVMs. In this section, we discuss issues when they are applied to solve linear SVMs.

2.1 Slow Convergence

Unlike popular optimization methods such as Newton or quasi-Newton which enjoy fast convergence, decomposition methods converges slowly as in each iteration only very few variables are updated. We will show that the situation is even worse when solving linear SVMs.

It has been demonstrated (e.g. (Hsu and Lin 2002b)) by experiments that if C is large and the Hessian matrix Q is not well-conditioned, decomposition methods converge very slowly. For linear SVMs, if $n \ll l$, then Q is a low-rank matrix which is not well-conditioned. When C is large, we can see the number of decomposition iterations dramatically increases. The situation is much worse than that for non-linear SVMs. In Figure 2.1, we demonstrate a simple example by using the problem heart from the statlog database (Michie, Spiegelhalter, and Taylor 1994). Each attribute is scaled to $[-1,1]$. We use LIBSVM (Chang and Lin 2001b) to solve linear and nonlinear (RBF kernel, $e^{-\|x_i-x_j\|^2/(2\sigma^2)}$ with $1/(2\sigma^2) = 1/n$) SVMs with $C = 2^{-8}, 2^{-7.5}, \ldots, 2^{8}$ and present the number of iterations. Though two different problems are solved (in particular, their Q_{ij}'s are in different ranges), Figure 2.1 clearly indicates the huge number of iterations for solving the linear SVMs. In the following, we give some theoretical explanation about this difficulty.

For problems which are not linearly separable, (Keerthi and Lin 2003) proved the following result:

Theorem 1 *There exists a finite value C^* and (w^*, b^*) such that $(w, b) = (w^*, b^*)$ solves (1.1) after $C \geq C^*$. In addition, $\sum_{i=1}^{l} \xi_i$ is a constant after $C \geq C^*$.*

Therefore, after $C \geq C^*$, $\alpha^T Q \alpha = w^T w$ becomes a constant. Since $\frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i = e^T \alpha - \frac{1}{2} \alpha^T Q \alpha$, after $C \geq C^*$, $\frac{1}{2} \alpha^T Q \alpha - e^T \alpha$ is a linear decreasing function of C.

It has been shown in (Lin 2001) that, under some conditions, a commonly used decomposition method is linearly convergent. That is, if α^* is a dual optimal solution and α^k is the solution at the kth iteration, then there is a $0 < \lambda < 1$ such that after k is large enough

$$\begin{aligned}
&\frac{1}{2}(\alpha^{k+1})^T Q \alpha^{k+1} - e^T \alpha^{k+1} - (\frac{1}{2}(\alpha^*)^T Q \alpha^* - e^T \alpha^*) \\
\leq\ & \lambda(\frac{1}{2}(\alpha^k)^T Q \alpha^k - e^T \alpha^k - (\frac{1}{2}(\alpha^*)^T Q \alpha^* - e^T \alpha^*)). \qquad (2.1)
\end{aligned}$$

Therefore, using the zero vector as the initial point, the smaller the optimal value is, more decomposition iterations are needed.

When C is large, $\frac{1}{2}\alpha^T Q\alpha - e^T\alpha$ decreases linearly with C. Though linear SVMs with $n < l$ do not satisfy the assumptions in (Lin 2001), if (2.1) still holds, the linear decrement of objective value results in the difficulty that the number of decomposition iterations may increase linearly.

Furthermore, (Lin 2001, Section 4) explains that the linear convergence rate, the reciprocal of λ in (2.1), is relatively smaller than that of non-linear SVMs whose kernel matrices are well-conditioned. This indicates that decomposition methods is inherently not suitable for linear SVMs due to its slow convergence.

2.2 Special Implementation for Linear SVMs

Though we have shown a disadvantage of using decomposition methods for linear SVMs, for practical implementations, we can apply some special properties of linear SVMs to speed up each iteration.

Most decomposition implementations maintain the gradient vector of the dual objective function during iterations. It is used for selecting the working set or checking the stopping condition. We usually calculate the gradient $Q\alpha - e$ by the following way: Suppose α^{k+1} and α^k are solutions of two consecutive decomposition iterations,

$$Q\alpha^{k+1} - e = Q\alpha^k - e + Q(\alpha^{k+1} - \alpha^k).$$

Since from α^k to α^{k+1}, only q components are changed, $Q(\alpha^{k+1} - \alpha^k)$ involves with q columns of the matrix Q. For a non-linear SVM where Q is too large to be stored in the computer memory, the calculation of lq kernel elements becomes the main computational cost in each decomposition iteration. If each kernel evaluation requires $O(n)$ operations, $O(lnq)$ is the complexity of each iteration.

However, for linear SVMs,

$$Q(\alpha^{k+1} - \alpha^k) = X^T(X(\alpha^{k+1} - \alpha^k)),$$

where $X = [y_1x_1, \ldots, y_lx_l]$ is an n by l matrix. Thus, $X(\alpha^{k+1} - \alpha^k)$ involves $O(nq)$ operations and $X^T(X(\alpha^{k+1} - \alpha^k))$ needs $O(ln)$. Therefore, $O(lnq)$ operations are largely reduced to $O(ln)$.

The first decomposition software which implements this for linear SVMs is SVM^{light} (Joachims 1998). Another implementation is BSVM (Hsu and Lin 2002b).

However, for SMO implementations where $q = 2$, the main cost of each iteration is only reduced by half. From this aspect, SMO type implementations are particular not suitable for linear SVMs. In other words, using a larger q, the cost on updating the gradient is the same but the number of iterations is smaller. Thus, we should increase q until the cost of solving the sub-problem (usually $O(q^3)$) becomes a dominant part.

3 Alpha Seeding for Linear SVMs

Results in (Keerthi and Lin 2003) show the following properties of the dual linear SVM:

There is C^* such that for all $C \geq C^*$, there are optimal solutions at the same face. In addition, for all $C \geq C^*$, the primal solution w is the same. The definition that two points are on the same face is as follows:

Let α^1 be a feasible vector of (1.2) for $C = C_1$ and α^2 be a feasible vector of (1.2) for $C = C_2$. We say that α^1 and α^2 are on the same face if the following hold: (i) $\{i \mid 0 < \alpha_i^1 < C_1\} = \{i \mid 0 < \alpha_i^2 < C_2\}$; (ii) $\{i \mid \alpha_i^1 = C_1\} = \{i \mid \alpha_i^2 = C_2\}$; and, (iii) $\{i \mid \alpha_i^1 = 0\} = \{i \mid \alpha_i^2 = 0\}$. Therefore, a face means a partition of $\{1, \ldots, l\}$ to three sets where corresponding components of α are free, upper-bounded, and lower-bounded.

Based on these properties, we conjecture that for large C, optimal solutions are at similar faces. Therefore, if α^1 is an optimal solution at $C = C_1$, then $\alpha^1 C_2 / C_1$ can be a very good initial point for solving (1.2) with $C = C_2$. This technique, called alpha seeding, was originally proposed for SVM model selection (DeCoste and Wagstaff 2000) where several (1.2) with different C have to be solved.

Earlier work which focus on nonlinear SVMs mainly use alpha seeding as a heuristic. Now for linear SVMs, $\alpha^1 C_2 / C_1$ is at the same face as α^1 so most likely it is at a similar face of one optimal solution of $C = C_2$. This strongly supports its use as the initial solution.

Next, we conduct some comparisons between the proposed and the original implementations. Here, we consider two-class problems only. Some statistics of the data sets used are in Table 3.2. The four small problems are from the statlog collection (Michie, Spiegelhalter, and Taylor 1994). The problem adult is compiled by Platt (1998) from the UCI "adult" data set (Blake and Merz 1998). Problem web is also from Platt. Problem ijcnn is from the first problem of IJCNN challenge 2001 (Prokhorov 2001). Note that we use the winner's transformation of the raw data (Chang and Lin 2001a).

We train linear SVMs with $C = [2^{-8}, 2^{-7.5}, \ldots, 2^{8}]$. That is, $[2^{-8}, 2^{8}]$ is discretized to 33 points with equal ratio. Table 3.1 presents the total number of iterations of training 33 linear SVMs using the alpha seeding approach. We also individually solve them by LIBSVM and list the number of iterations (total, $C = 2^{7.5}$, and $C = 2^{8}$). The alpha seeding implementation, based on LIBSVM, will be described in detail in Section 4. For the new approach, we also list the approximate C^* for which linear SVMs with $C \geq C^*$ have the same decision function. In addition, the constant $w^T w$ after $C \geq C^*$ is also given. For some problems (e.g. web and adult), $w^T w$ has not reached a constant until C is very large so we indicate them as "unstable" in Table 3.1.

It can be clearly seen that the alpha seeding approach performs so well to the point that its total number of iterations is much less than solving one single linear SVM with the original decomposition implementation. Therefore, even if we intend to solve one linear SVM with a particular C, using the proposed alpha seeding method starting from a smaller C may be more efficient.

Our experimental results indicate that the slow convergence of decomposition methods causes a huge number of iterations for changing some initial zero components to the upper bound C. On the other hand, the new approach starts from a small C where an optimal solution can be easily obtained. Then, as the next C is only slightly increased, the optimal solution face is not changed much. Hence, using the previous solution multiplied by the increase of C as an initial point, the saving on the number of iterations is dramatic. Furthermore, since we have solved linear SVMs with different C, model selection by cross-validation is already done. To be more precise, we can randomly separate data to different folds first. If one fold is singled out as the validation set, we sequentially train the rest and predict the validation set using different C.

To demonstrate that alpha seeding is much more effective for linear than nonlinear SVMs, Table 3.2 presents the number of iterations using the RBF kernel

$K(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$ with $1/2\sigma^2 = 1/n$. It can be clearly seen that the saving of iterations by using alpha seeding is marginal. In addition, comparing to the "total iter." column in Table 3.1, we confirm again the slow convergence for linear SVMs if without alpha seeding.

To further justify the alpha seeding approach for linear SVMs, we prove the following theorem:

Theorem 2 *Assume that any two parallel hyperplanes in the feature space do not contain more than $n+1$ points of $\{x_i\}$ on them. We have*

1. *For any optimal solution of (1.2), it has no more than $n+1$ free components.*
2. *There is C^* such that after $C \geq C^*$, all optimal solutions of (1.2) share at least the same $l-n-1$ upper and lower-bounded α variables.*

The proof is in Appendix A. This result indicates that if $n \ll l$, starting from small C, most components of optimal solutions are at bounds. From any C_1 to C_2, even if all free components are different, two solutions share at least $l - 2(n+1)$ bounded variables. If C_2 is not far away from C_1, it is less likely that an upper-bounded (lower-bounded) component at C_1 becomes lower-bounded (upper-bounded) at C_2. Thus, it is highly possible that the initial solution by alpha seeding has correctly identify at least $l-2(n+1)$ components of an optimal solution.

Theorem 2 also helps to explain why web is the most difficult problem for results in Table 3.1. Its large number of attributes might lead to more free variables during iterations or at the final solution. Thus, alpha seeding is less effective.

Another result which supports the use of alpha seeding is the following theorem:

Theorem 3 *There are two vectors A, B, and a number C^* such that for any $C \geq C^*$, $AC + B$ is an optimal solution of (1.2).*

The proof is in Appendix B. This theorem extends the result in (Keerthi and Lin 2003) which shows only that for any $\bar{C} > C^*$, there are optimal solutions α which form a linear function of C on the interval $[C^*, \bar{C}]$. Clearly $A_i \geq 0$, for $i = 1, \ldots, l$, so we can consider the following three situations of vectors A and B:

Table 3.1: Comparison of iterations (linear kernel); with and without alpha seeding.

	α-seeding			without α-seeding		
Problem	#iter	C^*	w^Tw	#total iter	#iter($C = 2^{7.5}$)	#iter($C = 2^8$)
heart	27231	$2^{3.5}$	5.712	2449067	507122	737734
australian	79162	$2^{2.5}$	2.071	20353966	3981265	5469092
diabetes	33264	$2^{6.5}$	16.69	1217926	274155	279062
german	277932	2^{10}	3.783	42673649	6778373	14641135
web	24044242	unstable	unstable	$\geq 10^8$	74717242	$\geq 10^8$
adult	3212093	unstable	unstable	$\geq 10^8$	56214289	84111627
ijcnn	590645	2^6	108.6	41440735	8860930	13927522

Table 3.2: Comparison of iterations (RBF kernel); with and without alpha seeding.

Problem	l	n	α-seeding	without α-seeding
heart	270	13	43663	56792
australian	690	14	230983	323288
diabetes	768	8	101378	190047
german	1000	24	191509	260774
web	49749	300	633788	883319
adult	32561	123	2380265	4110663
ijcnn	49990	22	891563	1968396

1. $0 < A_i \leq 1$,

2. $A_i = 0, B_i = 0$,

3. $A_i = 0, B_i > 0$.

For the second case, $\alpha_i^1 \frac{C_2}{C_1} = A_i C_2 + B_i = 0$. For the first case, $A_i C \gg B_i$ after C is large enough. Thus, $\alpha_i^1 \frac{C_2}{C_1} = A_i C_2 + B_i \frac{C_2}{C_1} \approx A_i C_2 + B_i$. Using Theorem 2, there are few ($\leq n+1$) components satisfying the third case. This analysis also shows the effectiveness of using alpha seeding.

4 Implementation

Though the concept is so simple, to have an efficient and elegent implementation, there are many considerations. In this section, we will have detailed discussions.

4.1 Stopping Condition of Alpha Seeding

Remember that there is C^* such that after $C \geq C^*$, the optimal w is the same. Though C^* is not known a priori, if a linear SVM with $C \geq C^*$ is already solved, it is possible to stop the alpha seeding procedure earlier. As $w^T w$ is an increasing function of C (e.g. using a similar proof as Lemma 4 of (Chang and Lin 2001c)), we may think that checking whether $w^T w$ becomes a constant is an effective method. Our experience indicates that in general it is. However, some concerns remain as follows:

1. As $w^T w$ is not strictly increasing, it is possible that $w^T w$ is a constant on an interval of C and start to increase later.

2. Now $w^T w = \alpha^T Q \alpha$ is a constant after C is large enough. As we solve the dual problem whose optimal α contains large components, the numerical error on calculating $\alpha^T Q \alpha$ may not be negligible. Note that though α has large values, $\alpha^T Q \alpha$ can be quite small. For example, problem australian has C^* at around 4 but after $C \geq C^*$, $w^T w = 2.071$, a small number. We find out that if only single precision is used for kernel evaluations, the obtained $w^T w$ can be erroneous.

Therefore, as the proposed procedure is very efficient, simply specifying a range of C and directly solving them may be more reliable in practice.

4.2 Initial Gradient

Though the proposed approach largely reduces the number of iterations, if we directly solve the sequence of linear SVMs using existing software, the computational time can be still huge. The main efforts will be on calculating the initial gradient $Q\alpha - e$. Existing implementations start from zero so the initial gradient $-e$ is easily obtained. Now the initial α may have many nonzero elements. To handle this bottleneck, roughly we can consider two types of implementations: with or without gradient reuse:

1. If α is the final solution of the linear SVM with $C = C_1$, then the gradient of using $\alpha C_2/C_1$ as initial solution for $C = C_2$ is $(C_2/C_1)Q\alpha - e$. In other

words, the final gradient of solving the previous linear SVM can be easily used for finding the initial gradient of the next linear SVM. We refer to it the "gradient reuse" strategy.

2. As we are solving linear SVMs, $Q\alpha = X^T(X\alpha)$ can be efficiently obtained in $O(ln)$ time. This plays a small part of the whole decomposition method provided $n \ll l$.

The difference of the two approaches on the training time is generally small. However, the first approach requires an additional array to convey $Q\alpha$ from one SVM to another.

4.3 With or Without Cache

For nonlinear SVMs, storing recently used Q_{ij} in cache can save many kernel evaluations. As mentioned in Section 2, if q components of α are updated and the corresponding q columns of Q are available, the computational cost for updating the gradient is reduced to $O(lq)$ from $O(lqn)$. As the change of C does not affect the kernel matrix, we can pass the cache of solving the previous SVM to the next one. On the other hand, cache may not be needed as Section 2 has also said that for linear SVMs, using $Q(\alpha^{k+1} - \alpha^k) = X^T(X(\alpha^{k+1} - \alpha^k))$, the cost is $O(ln)$. In the following we discuss some possible advantages and disadvantages of using cache:

1. If $q < n$, and problems are small, the whole kernel is cached so the program is n/q times faster.

2. As we expect that solutions are at similar faces, cache left from solving the provious linear SVMs can be very useful for the next problem.

3. If one of the q columns is not available in the cache, $O(ln)$ time is already needed to construct this column. This can happen very often for large problems. In the worst case, the cost can be $O(lqn)$, q times higher than without using the cache. In other words, the cache is associated with the matrix Q so we cannot take the advantage of linear SVMs. This clearly shows for large linear problems, cache should not be used.

4. Retaining cache from solving one problem to another certainly complicates the implementation. For example, shrinking may have cause cached columns to have different length. Thus, we must be careful on passing the cache to another problem.

4.4 Overall Considerations

Above discussions lead us to conclude that:

1. Implementation of decomposition methods for linear and nonlinear SVMs are quite different. We should consider them separately in some sense.

2. The implementation of alpha seeding for linear and nonlinear SVMs are also different. For nonlinear SVMs, gradient reuse for the initial calculation is a must but linear SVMs do not necessarily have to do so.

Here, we implement the simplest way: without gradient reuse and without cache. It may not be particularly efficient for small problems but is very competitive with other options for large problems.

5 Comparison with Other Approaches

Table 5.1: Comparison of different approaches for linear SVMs (time in second; q: size of the working set of decomposition methods).

	Decomposition methods with alpha seeding				Methods for linear SVMs	
	$q = 2$ (LIBSVM)		$q = 30$ (BSVM)		ASVM	LSVM
problem	total iter.	time	total iter.	time	time	time
australian	79162	2.87	145	0.85	3.79	0.83
heart	27231	0.39	65	0.14	0.40	0.39
diabetes	33264	1.48	118	0.3	0.73	0.86
german	277932	21.43	144	0.65	2.80	3.50
ijcnn	590645	1119.99	1409	46.71	215.37	633.66
adult	3212093	1123.18	129440	153.52	896.56	4605.17
web	24044242	4154.53	1559664	1140.09	4277.12	44468.48

It is interesting to compare the proposed approach with other methods. In particular, there are approaches which are mainly suitable for linear SVMs. In this section, we consider Active SVM (ASVM) (Mangasarian and Musicant 2000) and Lagrangian SVM (LSVM) (Mangasarian and Musicant 2001).

LIBSVM, the software used in Section 3, implements an SMO-type decomposition method where in each iteration two variables are updated. For problems with more free variables at final solutions, many such updates (i.e. iterations) are needed. We suspect that a decomposition method with a larger working set can finish in even fewer iterations. Thus, here we implement the alpha seeding apporach in another decomposition software BSVM (Hsu and Lin 2002b) which allows an arbitrary size of the working set.

However, BSVM, ASVM, and LSVM all solve slightly different formulations from (1.1). Thus, it is very difficult to conduct a fair comparison. Our goal here is only to demonstrate that decomposition methods, which are originally unsuitable for solving linear SVMs, can be competative with other linear-SVM methods, using the proposed implementation. In the following we briefly describe the three implementions.

BSVM modifies the objective function of (1.1) to be

$$\min_{w,b,\xi} \quad \frac{1}{2}(w^Tw + b^2) + C\sum_{i=1}^{l}\xi_i.$$

Then, the dual problem does not have the linear constraint $y^T\alpha = 0$. Its working set selection is slightly different from that of LIBSVM and SVM^{light}. Details are in (Hsu and Lin 2002b).

ASVM and LSVM both consider a square error term in the objective function:

$$\min_{w,b,\xi} \quad \frac{1}{2}(w^Tw + b^2) + C\sum_{i=1}^{l}\xi_i^2. \tag{5.1}$$

With b^2 included as well, the dual problem of (5.1) is

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + yy^T + \frac{I}{2C})\alpha - e^T\alpha \\ \text{subject to} \quad & 0 \le \alpha_i, \quad i = 1, \ldots, l., \end{aligned} \tag{5.2}$$

where I is the identity matrix. The solution of (5.2) has far more free components than that of (1.2) as upper-bounded variables of (1.2) are likely to be free now.

The optimality condition of (5.2) is very simple: for any α_i of a final solution, either

$$\begin{aligned} &\alpha_i > 0 \quad \text{and} \quad (\bar{Q}\alpha)_i - 1 = 0 \text{ or} \\ &\alpha_i = 0 \quad \text{and} \quad (\bar{Q}\alpha)_i - 1 \geq 0, \end{aligned}$$

where $\bar{Q} \equiv (Q + yy^T + \frac{I}{2C})$. ASVM iteratively guesses more variables to be at zero: if α^k is the solution of the kth iteration, they define

$$B^k \equiv \{i \mid \alpha_i^k > 0\}$$

and solve

$$\alpha_{B^k}^{k+1} = (\bar{Q}_{B^k B^k}^{-1} e_{B^k})_+, \tag{5.3}$$

where $(\cdot)_+ \equiv \max(\cdot, 0)$. Other elements of α^{k+1} (i.e., α_i^{k+1}, $i \notin B^k$) are kept zero. Thus, α^{k+1} contains more zero variables than α^k. If α^k is already at a correct face, the solution of (5.3) is optimal. However, such guessing may be wrong sometimes. Additional checks and corrections are required and the whole procedure is more complicated. The matrix inversion in (5.3) is very difficult when B^k is $\{1, \ldots, l\}$ in the beginning. However, for linear SVMs with $n \ll l$, using Sherman-Morrison-Woodbury (SMW) formula and $Q = XX^T$, we only have to invert a smaller n by n matrix.

LSVM is an iterative procedure where

$$\alpha^{k+1} = \bar{Q}^{-1}(e + ((\bar{Q}\alpha^k - e) - \mu\alpha^k)_+), \tag{5.4}$$

and $\bar{Q} = (Q + yy^T + \frac{I}{2C})$. The parameter μ is chosen to satisfy

$$0 < \mu < \frac{1}{C}.$$

Here, following (Mangasarian and Musicant 2001), we choose it as $\frac{0.95}{C}$. The same paper also proves that (5.4) linearly converges to the solution of (5.2). Similar to ASVM, the matrix inversion in (5.4) is difficult for non-linear SVMs but is easy for linear SVMs with $n \ll l$.

We try to use similar stopping criteria for the four approaches. If $f(\alpha)$ is the objective function, the stopping condition of LIBSVM is

$$\begin{aligned} &\max\{-y_i \nabla f(\alpha)_i \mid y_i = 1, \alpha_i < C \text{ or } y_i = -1, \alpha_i > 0\} - \\ &\min\{-y_i \nabla f(\alpha)_i \mid y_i = -1, \alpha_i < C \text{ or } y_i = 1, \alpha_i > 0\} \leq \epsilon, \end{aligned} \tag{5.5}$$

where $\epsilon = 0.001$. This is from the Karush-Kuhn-Tucker (KKT) condition (i.e. the optimality condition) of (1.2): α is optimal if and only if α is feasible and there is a number b and two nonnegative vectors λ and μ such that

$$\nabla f(\alpha) + by = \lambda - \mu,$$
$$\lambda_i \alpha_i = 0, \mu_i (C - \alpha)_i = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1, \ldots, l,$$

where $\nabla f(\alpha) = Q\alpha - e$ is the gradient of $f(\alpha)$, the objective function of (1.2). This can be rewritten as

$$\begin{aligned} \nabla f(\alpha)_i + by_i \leq 0 &\quad \text{if } \alpha_i > 0, \\ \nabla f(\alpha)_i + by_i \geq 0 &\quad \text{if } \alpha_i < C. \end{aligned}$$

Using $y_i = \pm 1$ and reformulating (5.6) as upper and lower bounds of b and introducing a stopping tolerance 0.001, we have (5.5). For BSVM, without the linear constraint $y^T \alpha = 0$, (5.5) can be simplified to

$$\max_{\alpha_i > 0} \nabla f(\alpha)_i - \min_{\alpha_i < C} \nabla f(\alpha)_i \leq \epsilon. \tag{5.6}$$

For ASVM, the $\alpha_i \leq C$ constraints are removed so (5.6) is further reduced to

$$\max_{\alpha_i > 0} \nabla f(\alpha)_i - \min_{i} \nabla f(\alpha)_i \leq \epsilon. \tag{5.7}$$

However, (5.7) is not suitable for LSVM as it does not keep $\alpha_i \geq 0$ throughout all iterations. Thus, we modify (5.7) to be

$$\max_{\alpha_i > \epsilon/100} \nabla f(\alpha)_i - \min_{i} \nabla (\alpha)_i \leq \epsilon$$

and

$$\alpha_i \geq -\epsilon/100, \forall i,$$

where $\epsilon = 0.001$, as the stopping condition of LSVM. This has been used in (Lin and Lin 2003) which implements LSVM for solving Reduced SVM (Lee and Mangasarian 2001).

Table 5.1 presents a comparison of the four implementations. We use the same benchmark problems as in Section 3. The computational experiments were done on a Pentium III-1000 with 1024MB RAM using the gcc compiler.

For each problem, 33 linear SVMs with $C = 2^{-8}, 2^{-7.5}, \cdots, 2^{8}$ are solved. In addition to the total computational time, we also list the number of iterations of the two decomposition implementations. Clearly for problems with small n, for each C, BSVM takes very few iterations. The computational time is also less than that of LIBSVM. This is consistent with our earlier statement that for linear SVMs, SMO-type decomposition methods are less favorable than general decomposition methods with larger working sets.

Since alpha seeding is not applied to ASVM and LSVM, we admit that their computational time can be improved. Results here also serves as the first comparison between ASVM and LSVM. Clearly ASVM is faster. Moreover, due to the huge computational time, we set the maximal iterations of LSVM to be 1,000. For problems adult and web, after C is large, the limit of iterations is reached before stopping conditions are satisfied.

We set the size of the working set of BSVM to be 30. The code of BSVM follows from the same structure of LIBSVM so the timing comparison really reflects their algorithmic difference. For ASVM, we directly use the authors' C++ implementation available at `http://www.cs.wisc.edu/dmi/asvm`. The authors of LSVM provide only MATLAB programs so we implement it by modifying LIBSVM. We must exercise the caution that quite a few implementation details affect the computational time. For example, each iteration of ASVM and LSVM involves several matrix-vector multiplications. Hence, it is possible to use finely-tuned dense linear algebra subroutines. For the LSVM implementation here, by using ATLAS (Whaley, Petitet, and Dongarra 2000), for large problems, the time is reduced by two third. Thus, it is possible to further reduce the time of ASVM in Table 5.1 though we find it too complicated to modify the authors' program. Using such tools also means X is considered a dense matrix. On the other hand, for LIBSVM and BSVM where each iteration requires two matrix-vector multiplications $X(X^T(\alpha^{k+1} - \alpha^k))$, currently X is considered a sparse matrix. This creates some overheads when data are dense.

6 Experiments on Model Selection

If the RBF kernel

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$$

is used, (Keerthi and Lin 2003) proposes the following model selection procedure for finding good C and σ^2:

Algorithm 1 *Two-line model selection*

1. *Search for the best* C *of linear SVMs and call it* $\tilde{C}$.

2. *Fix* $\tilde{C}$ *from step 1 and search for the best* (C, σ^2) *satisfying* $\log \sigma^2 = \log C - \log \tilde{C}$ *using the RBF kernel.*

That is, we solve a sequence of linear SVMs first and then a sequence of nonlinear SVMs with the RBF kernel. The advantage of this approach over an exhausted search of the parameter space is that only parameters on two lines are considered. If the same decomposition method is used for both linear and nonlinear SVMs here, due to the huge number of iterations, solving the linear SVMs becomes the bottleneck. Our goal in this section is to show that using the new approach for linear SVMs, the computational time spent on the linear part becomes similar to that on the nonlinear SVMs.

Earlier in (Keerthi and Lin 2003), due to the difficulty on solving linear SVMs, only small two-class problems are tested. Here, we would like to evaluate this approach on large multi-class datasets. We consider problems: dna, satimage, letter, and shuttle. They were originally from the statlog collection (Michie, Spiegelhalter, and Taylor 1994) and were used in (Hsu and Lin 2002a). Except problem dna which has binary attributes, we scale all training data to be in $[-1, 1]$. Then, test data are adjusted using the same linear transformation.

We search for $\tilde{C}$ by five-fold cross-validation on linear SVMs using uniformly spaced $\log_2 \tilde{C}$ value in $[-10, 10]$ (with grid space 1). As LIBSVM considers $\gamma = 1/2\sigma^2$ as the kernel parameter, the second step is to search for good (C, γ) satisfying

$$-1 - \log_2 \gamma = \log_2 C - \log_2 \tilde{C}. \tag{6.1}$$

We discretize $[-10, 4]$ as values of $\log_2 \gamma$ and calculate $\log_2 C$ from (6.1). To avoid that $\log_2 C$ locates in an abnormal region, we consider only points with $-2 \leq \log_2 C \leq 12$ so the second step may solve less SVMs than the first step. The implementation is modified from the python interface of LIBSVM (Chang and Lin 2001b). The same computational environment as that for Section 5 is used.

Since this model selection method is based on the analysis of binary SVMs, a multi-class problem has to be decomposed to several binary SVMs. We employ the so called "one-against-one" approach: if there are k classes of data, we consider all $k(k-1)/2$ combinations of any two classes. For any two classes of data, the model selection is conducted to have the best (C, σ^2). With the $k(k-1)/2$ decision functions, a voting strategy is used for the final prediction. Note that now each decision function has its own (C, σ^2). In Table 6.1, we compare this approach with a complete grid search. For any two classes of data, five-fold cross-validation is conducted on 225 points, a discretization of the $(\log_2 C, \log_2 \gamma) = [-2, 12] \times [-10, 4]$ space. This is different from the cross-validation procedure adopted by LIBSVM where a list of (C, σ^2) is selected first and then for each (C, σ^2), one-against-one method is used for estimating the cross-validation accuracy of the whole multi-class data. Therefore, for the final optimal model, $k(k-1)/2$ decision functions share the same C and σ^2. Clearly, methods not based on the analysis of binary SVMs can take both approaches. The complete grid search by cross-validation is an example. Since the same number of nonlinear SVMs are trained, the time for the two complete grid searches is exactly the same but the performance (test accuracy) may be different. Since there is no comparison so far, we also present a preliminary investigation here.

Table 6.1 presents experimental results. For each problem, we compare test accuracy by two complete grid searches and the model selection method by Keerthi and Lin. The two grid searches are represented as "1 (C, σ^2)" and "$k(k-1)/2$ (C, σ^2)," respectively in Table 5.1 depending on how many (C, σ^2) used by the decision functions. The performance of the three approaches are very similar. However, the total model selection time of the method by Keerthi and Lin is much shorter. We achieve this because of using the alpha seeding. Otherwise, time for solving linear SVMs is a lot more so the proposed model selection does not possess any advantage.

We then investigate the stability of the new model selection approach. Due to timing restriction, we consider two smaller problems banana and adult_small tested in (Keerthi and Lin 2003). Note that the adult_small is a subset of the adult used in Section 3. It is a binary problem with 1,605 examples. Table 6.2 shows the means and standard deviations of parameters and accuracies using 10-time the "$k(k-1)/2$ (C, σ^2)" grid search and the method by Keerthi and Lin, respectively. For the two-line method, we list only $\tilde{C}$'s variances because the variances of parameters C and σ^2, which are computed from (6.1), are less meaningful. Note that different parameters as well as accuracy by applying the same method 10 times are due to the randomness of cross-validation.

From Table 6.2, we can see that although the performance (testing accuracy and consumed time) of the model selection method proposed by (Keerthi and Lin 2003) is good, it might be less stable. That is, the variance of accuracies is significantly larger than that of the complete grid search method while the variances of parameters are both large. We think that in the complete grid search method, the cross-validation estimation bounds the overall error. Thus, the variances of gained parameters do not affect the testing performance. However, in the two-line search method (Algorithm 1), two-stage cross-validations are utilized. Thus, the variance in the first stage may affect the best performance of the second stage.

Table 6.1: Comparison of different model selection methods (time in second).

	Complete grid search			Method by Keerthi and Lin			
	1 (C, σ^2)	$k(k-1)/2$ (C, σ^2)		Time	Time	Time	Accuracy
Problem	Accuracy	Time	Accuracy		(linear)	(non-linear)	
dna	95.62	15238	95.11	1007	586	421	94.86
satimage	91.9	24505	92.2	3244	2378	866	91.55
letter	97.9	135881	97.72	15716	10028	5688	96.54
shuttle	99.92	243375	99.94	7887	5267	2620	99.81

7 Discussion and Conclusion

It is arguable that we may have used a too strict stopping condition in the decomposition method when C is large. If $C\epsilon$ instead of ϵ is used in (5.5), the number of iterations (without alpha seeding) would be much smaller. Thus, directly solving

Table 6.2: Mean and Standard deviation of two model selection methods (each method applied 10 times).

	Complete grid search						Method by Keerthi and Lin			
	$\log_2 C$		$\log_2 \gamma$		Accuracy		$\log_2 \tilde{C}$		Accuracy	
Problem	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
banana	7	4.45	-0.4	1.51	87.91	0.47	-1.9	2.18	76.36	12.21
adult_small	5.4	2.37	-7.6	1.71	83.82	0.27	0.3	4.08	83.20	1.28
dna	5.4	3.34	-5	0	95.56	0.19	-	-	94.85	0.20
satimage	2.5	0.71	0.1	0.57	91.74	0.24	-	-	91.19	0.28

Table 7.1: Comparison of the running time for linear SVMs with stopping tolerance $C\epsilon$ (time in second).

Problem	with alpha seeding	without alpha seeding
australian	2.30	6.90
heart	0.36	1.16
diabetes	0.75	7.97
german	17.58	46.24
ijcnn	1052.07	25779.50
adult	923.76	40059.15
web	2311.79	7547.94

linear SVMs with large C becomes possible. However, as shown in Table 7.1, with alpha seeding the decomposition method still makes the computational time several times faster than without alpha seeding, especially for large datasets.

The stopping condition has long been a controversial issue for SVM software design. So far we have not found out a satisfactory way which is not too loose or too strict for most problems. This is why in most software, ϵ is left to be adjusted by users. For example, although using $C\epsilon$ takes a significant advantage on running time, especially for solving linear SVMs, it may lead us to a wrong optimal solution when C is too large. An extreme scenario is as follows: Let the initial solution $\alpha = 0$ and

$$\nabla f(\alpha) = Q\alpha - e = e.$$

The stopping condition (5.5) becomes

$$\begin{aligned} &\max\{-y_i e_i \mid y_i = 1, \alpha_i < C \text{ or } y_i = -1, \alpha_i > 0\} - \\ &\min\{-y_i e_i \mid y_i = -1, \alpha_i < C \text{ or } y_i = 1, \alpha_i > 0\} \leq 2. \end{aligned}$$

If $C\epsilon > 2$, the initial α already satisfies the stopping condition. Then, the optimization procedure stops with $w = 0$, an obviously wrong solution. In addition, in Section 5 we have shown that other methods work well under the same strict condition. Since a large stopping tolerance may cause a fake convergence, decomposition methods should also be efficient enough under the same strict setting.

In conclusion, we hope that based on this work, SVM software using decomposition methods can be suitable for all types of problems, no matter $n \ll l$ or $n \gg l$.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 90-2213-E-002-111.

References

Blake, C. L. and C. J. Merz (1998). UCI repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA. Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Chang, C.-C. and C.-J. Lin (2001a). IJCNN 2001 challenge: Generalization ability and text decoding. In *Proceedings of IJCNN.*

Chang, C.-C. and C.-J. Lin (2001b). *LIBSVM: a library for support vector machines.* Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chang, C.-C. and C.-J. Lin (2001c). Training ν-support vector classifiers: Theory and algorithms. *Neural Computation 13*(9), 2119–2147.

Cortes, C. and V. Vapnik (1995). Support-vector network. *Machine Learning 20*, 273–297.

DeCoste, D. and K. Wagstaff (2000). Alpha seeding for support vector machines. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD-2000).*

Fine, S. and K. Scheinberg (2001). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research 2*, 243–264.

Hsu, C.-W. and C.-J. Lin (2002a). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks 13*(2), 415–425.

Hsu, C.-W. and C.-J. Lin (2002b). A simple decomposition method for support vector machines. *Machine Learning 46*, 291–314.

Joachims, T. (1998). Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Keerthi, S. S. and C.-J. Lin (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*. To appear.

Keerthi, S. S., S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy (2000). A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks 11*(1), 124–136.

Lee, Y.-J. and O. L. Mangasarian (2001). RSVM: Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining*.

Lin, C.-J. (2001). Linear convergence of a decomposition method for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

Lin, K.-M. (2002). Reduction techniques for training support vector machines. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University.

Lin, K.-M. and C.-J. Lin (2003). A study on reduced support vector machines. *IEEE Transactions on Neural Networks*. To appear.

Mangasarian, O. L. and D. R. Musicant (2000). Active set support vector machine classification. In *Advances in Neural Information Processing Systems*, pp. 577–583.

Mangasarian, O. L. and D. R. Musicant (2001). Lagrangian support vector machines. *Journal of Machine Learning Research 1*, 161–177.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (1994). *Machine Learning, Neural and Statistical Classification.* Englewood Cliffs, N.J.: Prentice Hall. Data available at `http://www.ncc.up.pt/liacc/ML/statlog/datasets.html`.

Osuna, E., R. Freund, and F. Girosi (1997). Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, New York, NY, pp. 130–136. IEEE.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Prokhorov, D. (2001). IJCNN 2001 neural network competition. Slide presentation in IJCNN'01, Ford Research Laboratory. `http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf` .

Whaley, R. C., A. Petitet, and J. J. Dongarra (2000). Automatically tuned linear algebra software and the ATLAS project. Technical report, Department of Computer Sciences, University of Tennessee.

A Proof of Theorem 2

For the first result, if it were wrong, there was an optimal α with more than $n+1$ free components. From the optimality condition (5.6) and the primal-dual relation (1.4), for those $0 < \alpha_i < C$,

$$(Q\alpha + by)_i = y_i(w^T x_i + b) = 1.$$

Thus, more than $n+1$ x_i are at two parallel hyperplanes. This contradicts the assumption.

Next, we consider from Theorem 1 and define

$$A \equiv \{i \mid \text{for all optimal solutions with } C > C^*, \alpha_i \text{ is at the upper (lower) bound}\}.$$

The second result of this theorem will hold if we can prove $|A| \geq l - n - 1$. If its result is wrong, we can assume all those $i \notin A$ are from $\alpha^1, \ldots, \alpha^s$ which are optimal at $C = C_1, \ldots, C_s$, $C^* < C_1 < \cdots < C_s$ and $s > n + 1$. Thus, for any $i \notin A$, it is impossible that

$$\alpha_i^1 = \cdots = \alpha_i^s = 0 \text{ or } \alpha_i^1 = C_1, \ldots, \alpha_i^s = C_s. \tag{A.1}$$

For any convex combination with weights $0 < \lambda_j < 1$ and $\sum_{j=1}^s \lambda_j = 1$,

$$0 < \sum_{j=1}^s \lambda_j \alpha_i^j < \sum_{j=1}^s \lambda_j C_j, \quad \forall i \notin A. \tag{A.2}$$

Remember from Theorem 1, for all $C \geq C^*$, the optimal w is the same. If we define

$$\bar{\alpha} \equiv \sum_{j=1}^s \lambda_j \alpha^j,$$

then

$$\sum_{i=1}^l y_i \bar{\alpha}_i x_i = \sum_{j=1}^s \lambda_j \sum_{i=1}^l y_i \alpha_i^j x_i = \sum_{j=1}^s \lambda_j w = w.$$

Therefore, we have constructed an $\bar{\alpha}$ which is optimal at $C = \sum_{j=1}^s \lambda_j C_j > C^*$ with more than $n + 1$ free components (from (A.1) and (A.2)). This contradicts the first part of this theorem so the proof is complete.

B Proof of Theorem 3

First we note that for any given C, if $A_i C + B_i, i = 1, 2, \ldots$ are all optimal solutions of (1.2), and there are vectors A and B such that

$$\lim_{i \to \infty} A_i C + B_i = AC + B, \tag{B.1}$$

then $AC + B$ is an optimal solution as well. This is from the continuity of (1.2)'s objective function and the compactness of its feasible region.

From Theorem 3 of (Keerthi and Lin 2003), there is a C^* such that for any $\bar{C} > C^*$, there exists a linear function $AC + B$ which is optimal for (1.2) when $C \in [C^*, \bar{C}]$. Thus, we can consider $C^1 < C^2 < \cdots$ with $\lim_{i \to \infty} C^i = \infty$ and functions $A_i C + B_i$ which are optimal solutions at $C \in [C^*, C^i]$.

Since

$$0 \leq A_i C^1 + B_i \leq C^1 \text{ and } 0 \leq A_i C^2 + B_i \leq C^2,$$

are bounded for all i, there is an infinity set I such that

$$\lim_{i \to \infty, i \in I} A_i C^1 + B_i = \alpha^1 \text{ and } \lim_{i \to \infty, i \in I} A_i C^2 + B_i = \alpha^2. \tag{B.2}$$

If $\alpha^1 \neq \alpha^2$, then two different points uniquely determine vectors A and B such that

$$\alpha^1 = AC^1 + B \text{ and } \alpha^2 = AC^2 + B. \tag{B.3}$$

For any $C > C^2$, there is $0 < \lambda < 1$ such that $C^2 = \lambda C^1 + (1 - \lambda)C$ so

$$A_i C^2 + B_i = \lambda(A_i C^1 + B_i) + (1 - \lambda)(A_i C + B_i). \tag{B.4}$$

Taking the limit, (B.2), (B.3), and (B.4) imply

$$AC^2 + B = \lambda(AC^1 + B) + (1 - \lambda) \lim_{i \in I, i \to \infty} (A_i C + B_i).$$

Thus,

$$\lim_{i \in I, i \to \infty} (A_i C + B_i) = AC + B, \tag{B.5}$$

so (B.1) is valid. The situation for $C^* \leq C \leq C^2$ is similar. Thus $AC + B$ is optimal for (1.2), for all $C \geq C^*$. On the other hand, if $\alpha^1 = \alpha^2$, since $C^2 > C^1$,

$$\lim_{i \in I, i \to \infty} A_i (C^2 - C^1) = 0$$

from (B.2) implies

$$\lim_{i \in I, i \to \infty} A_i = 0 \text{ and } \lim_{i \in I, i \to \infty} B_i = \alpha^1 = \alpha^2.$$

By defining $A \equiv 0$ and $B \equiv \alpha^1$, for any $C \geq C^*$, (B.1) also holds so $AC + B$ is optimal for (1.2). Thus, the proof is complete. □