JMLR: Workshop and Conference Proceedings 29:72–86, 2013 ACML 2013

Accelerated Coordinate Descent with Adaptive Coordinate Frequencies

Tobias Glasmachers TOBIAS.GLASMACHERS@INI.RUB.DE
Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany

Ürün Dogan DOGANUBD@MATH.UNI-POTSDAM.DE
Institut für Mathematik, Universität Potsdam, Germany

Editor: Cheng Soon Ong and Tu Bao Ho

Abstract

Coordinate descent (CD) algorithms have become the method of choice for solving a number of machine learning tasks. They are particularly popular for training linear models, including linear support vector machine classification, LASSO regression, and logistic regression. We propose an extension of the CD algorithm, called the adaptive coordinate frequencies (ACF) method. This modified CD scheme does not treat all coordinates equally, in that it does not pick all coordinates equally often for optimization. Instead the relative frequencies of coordinates are subject to online adaptation. The resulting optimization scheme can result in significant speed-ups. We demonstrate the usefulness of our approach on a number of large scale machine learning problems.

Keywords: Coordinate Descent, Online Adaptation, Convex Optimization

1. Introduction

Coordinate Descent (CD) algorithms are becoming increasingly important for solving machine learning tasks. They have superseded other gradient-based approaches such as stochastic gradient descent (SGD) for solving certain types of problems, such as training of linear support vector machines as well as LASSO regression and other L_1 regularized learning problems (Friedman et al., 2007; Hsieh et al., 2008).

SGD is a natural competitor algorithm for solving the underlying convex optimization problems. An ubiquitous problem of SGD is the need to set a learning rate parameter, possibly equipped with a cooling schedule. There have been a number of proposals for adapting such parameters online for optimal progress (see Schaul et al. 2013 and references therein), which effectively removes the need to tune parameters to the problem at hand. Importantly, such techniques make it feasible to maintain a potentially large number of independent learning rates for different parameters, e.g., one per coordinate.

Inspired by online adaptation schemes for SGD learning rates we propose a related technique for speeding up CD algorithms. The quantity in CD corresponding to learning rates in SGD is the frequency for selecting a coordinate for optimization. We argue that touching all coordinates equally often is a choice that remains arbitrary, and that can be expected to perform sub-optimally in many cases.

We propose to model and adapt the relative frequencies for coordinate selection explicitly in the CD algorithm. Modeling the frequencies allows to pick important coordinates

more often than others and thus to make faster progress to the optimum. Consequently, adaptation of coordinate frequencies is driven by the progress made on the single-coordinate sub-problem. We refer to this technique as *Adaptive Coordinate Frequencies* (ACF), and to the resulting coordinate descent scheme as ACF-CD.

Our approach is related to previous work. First of all, the formulation of our method is most natural in the context of random coordinate descent as proposed by Nesterov (2012). It is closest in spirit to the Adaptive Coordinate Descent algorithm by Loshchilov et al. (2011) that adapts a coordinate system of descent directions online with the goal to make steps independent. This algorithm maintains a number of state variables (directions) that are subject to online adaptation. This is actually a common characteristic of many direct search heuristics (see e.g. Hansen and Ostermeier 2001) that is also shared by the already mentioned SGD methods with online adaptation of learning rates (Schaul et al., 2013) and by the resilient backpropagation (Rprop) algorithm by Riedmiller and Braun (1993).

Our approach is also loosely related to sequential minimal optimization (SMO, Platt 1998; Bottou and Lin 2007), in particular when applied with single-element working sets to training problems of non-linear support vector machine (SVM) without equality constraint (Steinwart et al., 2011). However, modern versions of these algorithms rely on elaborate coordinate selection heuristics (Fan et al., 2005; Bottou and Lin, 2007) that are computationally prohibitive in the context of this paper.

The remainder of this paper is organized as follows. First we review the basic coordinate descent algorithm and its use for solving convex optimization problems in machine learning. Then we generalize this algorithm to variable coordinate frequencies and introduce our novel adaptive coordinate frequencies CD scheme. The new algorithm is benchmarked against established CD solvers on a number of large scale machine training tasks. We close with brief conclusions.

2. Coordinate Descent Machine Training

We consider an n-dimensional convex optimization problem. The basic coordinate descent scheme for solving this problem iteratively is presented in algorithm 1. In general, CD methods can be efficient if single steps are cheap to compute, which alleviates for the fact that CD steps do not account for interrelations between variables. Convergence properties of CD iterates and their values have been established, e.g, by Luo and Tseng (1992); Tseng (2001); Shalev-Shwartz and Zhang (2013).

Algorithm 1 Coordinate Descent (CD) algorithm.

INPUT: $x^{(0)} \in \mathbb{R}^n$
$t = 0$
repeat
  select active coordinate $i^{(t)} \in \{1, \ldots, n\}$
  solve the optimization problem with additional constraints $x_j^{(t)} = x_j^{(t-1)}$ for all $j \neq i^{(t)}$
  $t \leftarrow t + 1$
until stopping criterion is met

CD can come in a number of variations. Here we want to highlight the difference between deterministic and randomized CD, referring to the selection of coordinates. The standard CD algorithm is deterministic. The coordinates $i^{(t)}$ are chosen cyclically in order. Alternatively it may be randomized (Nesterov, 2012), with $i^{(t)}$ drawn independently in each iteration from the uniform distribution.

CD methods have been popularized in statistics and machine learning especially for regularized empirical risk minimization problem with sparse solutions. Sparsity is often a result of regularization, most prominently with an L_1 penalty on a linear model's weight vector, which is the case in least absolute shrinkage and selection operator (LASSO) models (Friedman et al., 2007). Logistic regression with L_1 regularization is another prominent example (Yuan et al., 2012). Alternatively, sparsity (of the dual solution) can result from the empirical risk term, e.g., in a support vector machine with hinge loss. CD training of linear SVMs has been demonstrated to outperform competing methods (Hsieh et al., 2008).

Given data $\{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ composed of inputs $x_i \in \mathbb{R}^d$, labels $y_i \in Y$, and a loss function $L : \mathbb{R} \times Y \to \mathbb{R}$, the (primal, unconstrained) training problem of the linear predictor $x \mapsto \langle w, x \rangle$ amounts to

$$\min_{w \in \mathbb{R}^d} \quad \frac{\lambda}{p} \|w\|_p^p + \frac{1}{\ell} \sum_{i=1}^{\ell} L\big(\langle w, x_i \rangle, y_i\big) \tag{1}$$

where p is typically either 1 or 2 and $\lambda > 0$ is a complexity control parameter. With binary classification labels $Y = \{-1, +1\}$, hinge loss $L(z, y) = \max\{0, 1 - yz\}$ and $p = 2$ we obtain the linear SVM. Hsieh et al. (2008) solve the corresponding dual problem

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^\ell} \quad & \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle - \sum_{i=1}^{\ell} \alpha_i \\ \text{s.t.} \quad & 0 \le \alpha_i \le C = \frac{1}{\lambda} \end{aligned} \tag{2}$$

with CD. The key technique for making CD iterations fast is to keep track of the vector $w = \sum_{i=1}^{\ell} \alpha_i y_i x_i$ during optimization. A CD step in this box-constrained quadratic program amounts to a one-dimensional, interval-constrained Newton step. The first derivative of the objective function w.r.t. α_i is $y_i \langle w, x_i \rangle - 1$, the second derivative is $\langle x_i, x_i \rangle$, which can be precomputed. The resulting CD step reads

$$\alpha_i^{(t)} = \left[\alpha_i^{(t-1)} - \frac{1 - y_i \langle w, x_i \rangle}{\langle x_i, x_i \rangle} \right]_0^C ,$$

where $[x]_a^b = \max\big\{a, \min\{b, x\}\big\}$ denotes truncation of the argument x to the interval $[a, b]$. With densely represented $w \in \mathbb{R}^d$ and sparse data x_i the complexity of a step is not only independent of the data set size ℓ, but even as low as the number of non-zero entries in x_i (and therefore often much lower than the data dimension d).

With regression labels $Y = \mathbb{R}$, squared loss $L(z, y) = \frac{1}{2}(y - z)^2$ and one-norm penalty $p = 1$ we obtain the LASSO problem. Friedman et al. (2007) propose to solve the primal problem directly with CD. The objective function (1) is piecewise quadratic and unconstrained, the one-dimensional CD sub-problem consists of only two quadratic parts. It can thus be solved

analytically with few case distinctions (refer to Friedman et al. 2007 for details). Again, the cost for the computation of the gradient and the update of the weight vector is dominated by the number of non-zero elements of x_i (for coordinate $j \in \{1, \ldots, d\}$ the complexity is determined by the number of training points x_i with non-zero component $(x_i)_j$).

Many other machine learning methods can be obtained as modifications of these two base cases. For example, replacing the squared loss with cross-entropy gives L_1-regularized logistic regression. The resulting one-dimensional sub-problem has no closed form solution and can be approximated with a number of Newton steps (refer to Yuan et al. 2012 for additional speed-up techniques).

3. Non-uniform Coordinate Descent

In this section we extend the CD algorithm to non-uniform selection of coordinates. The standard cyclic coordinate selection rule as well as the random CD variant select all coordinates equally often (averaged over a large number of iterations). Instead we propose to model the relative frequencies for selecting the different coordinates explicitly.

This is easiest in the random CD algorithm. Let p_i denote the probability for selecting coordinate i. Random CD defaults to $p_i = 1/n$ for all $i \in \{1, \ldots, n\}$. We relax this setting by allowing for arbitrary $p = (p_1, \ldots, p_n)$ in the probability simplex $S = \{p \in \mathbb{R}^n \,|\, p_i \geq 0, \sum_{i=1}^n p_i = 1\}$. One possibility to translate the parameter $p \in S$ into a deterministic CD algorithm is as follows: maintain an accumulator variable a_i for each coordinate, initialize it to $a_i^{(0)} = 0$ and progress as $a_i^{(t)} = a_i^{(t-1)} + p_i$. Pick $i^{(t)} = \arg\max_i \left\{a_i^{(t)}\right\}$ and lower the accumulator of the chosen coordinate by one: $a_{i^{(t)}}^{(t)} \leftarrow a_{i^{(t)}}^{(t)} - 1$.[1]

The reasoning behind this extension is that uniform p is a symmetric but otherwise arbitrary choice. With knowledge available on how much progress can be made with which coordinate a uniform choice is most probably sub-optimal. In the following we investigate a few standard applications to showcase this argument.

Consider the dual SVM training problem (2). The optimal solution α^* is sparse, since at the optimum many variables α_i are at the bounds. Once a variable is at the bound it may stay there for the rest of the optimization run. Therefore its selection probability p_i should be reduced to zero. This is achieved in standard solver software (Fan et al., 2008) with a so-called shrinking heuristic (Joachims, 1998) that removes coordinate i from the problem, or equivalently sets p_i to zero. However, some constraints may become inactive later during the optimization run. Although undoing the shrinking operation is relatively cheap it remains unclear when and how frequently the shrinking decision should be reconsidered. This calls for a small but non-zero setting for p_i, which becomes possible by allowing for general $p \in S$.

As a second example consider an unconstrained, quadratic optimization problem

$$\min_{w \in \mathbb{R}^d} \quad f(w) = \frac{1}{2}(w - w^*)^T Q (w - w^*) \tag{3}$$

with positive definite, symmetric matrix $Q \in \mathbb{R}^{d \times d}$. This models the late optimization phase for LASSO as well as for the SVM problem (2), restricted to the non-zero (in the SVM case: unbounded) variables. In this sense the model is suitable for the analysis of convergence

1. A computationally cheaper approximate scheme will be introduced later on.

rates with any twice continuously differentiable loss function (e.g., logistic regression), with L_1 and L_2 regularization, under mild technical assumptions.

On problem (3) the random CD algorithm is described by a Markov chain on w, and (under weak technical assumptions, say, for data and $p \in S$ in general position) the corresponding Markov chain $w/\|w\|$ converges to a stationary distribution. The speed of convergence to the optimum is determined by the expected progress in the stationary distribution. Both the stationary distribution as well as the average progress depend upon the choice of p, which should therefore be considered an important parameter that needs tuning, or better, online adaptation.

4. Online Adaptation of Coordinate Frequencies

Tuning of p to the problem at hand requires prior knowledge of which variables are important. However, the acquisition of this knowledge may be the reason for training the model in the first place, e.g., in case of LASSO. Thus, this knowledge is generally not available. Furthermore the optimal setting for p will typically change during the course of the optimization run, e.g., when constraints become active or inactive. For these reasons p should be subject to online adaptation rather than manual tuning by the user. In this section we develop a mechanism for adapting p online.

We start by investigating the optimal p in simple, tractable cases. The symmetric case $Q_{ii} = q_1 > q_2 = Q_{ij}$ for $i \neq j$ is easiest to handle: for symmetry reasons uniform p is optimal. Note that this covers weak as well as strong *uniform* couplings of coordinates. Nothing can be gained by adapting p in this case. Thus, an online adaptation strategy must pass the sanity check to drive p towards uniformity for such problems.

A more realistic case is a mixture of strong and weak couplings within a single problem. To study the resulting effects we investigate the following minimal model problem for $w \in \mathbb{R}^3$, which is an instance of problem (3):

$$Q(\varepsilon, \delta) = \begin{pmatrix} 1 & 1-\varepsilon & \delta \\ 1-\varepsilon & 1 & \delta \\ \delta & \delta & 1 \end{pmatrix} \quad . \tag{4}$$

For small ε and δ the variables w_1 and w_2 are strongly coupled, while w_3 is only weakly related to the others. Intuitively the best sequence of CD steps is to oscillate between the first two coordinates, interleaved with a step on w_3 from time to time.

We analyze the suitability of p for $Q(\varepsilon, \delta)$ in terms of the (linear convergence) progress rate

$$R = \mathbb{E}\left[\log\left(\frac{f(w^{(t-1)}) - f^*}{f(w^{(t)}) - f^*}\right)\right] \quad ,$$

(higher is better) with the expectation taken in the stationary distribution. $f^* = 0$ denotes the optimal value. The same definition restricted to steps on coordinate i gives the coordinate-specific progress rate R_i.

Table 1 lists progress rates for various coupling strengths ε and δ and CD strategies p. The first row of each block corresponds to the uniform strategy with $p_1 = p_2 = p_3 = 1/3$.

ε	δ	p_3	R	(R_1, R_2, R_3)
0.1	0.1	1/3	0.105	$(0.139, 0.139, 0.038)$
		0.0293	0.191	$(0.165, 0.165, 0.656)$
		0.0538	0.181	$(0.181, 0.181, 0.181)$
0.02	0.1	1/3	0.020	$(0.013, 0.013, 0.035)$
		0.0065	0.040	$(0.033, 0.033, 0.521)$
		0.03	0.037	$(0.037, 0.037, 0.038)$
0.1	0.02	1/3	0.105	$(0.157, 0.157, 0.002)$
		0.021	0.196	$(0.175, 0.175, 0.709)$
		0.0336	0.191	$(0.191, 0.191, 0.191)$
0.4	0.02	1/3	0.511	$(0.760, 0.760, 0.012)$
		0.078	0.775	$(0.710, 0.710, 1.228)$
		0.0999	0.764	$(0.764, 0.764, 0.764)$

Table 1: Overall and coordinate-wise progress rates for uniform, near-optimal, and rate-balancing strategies, estimated over $10,000,000$ CD steps. The full distribution p is obtained from p_3 as $p_1 = p_2 = (1 - p_3)/2$.

This too frequent choice of $i = 3$ results in a low progress rate R_3. The parameter p_3 in the second row of table 1 has been hand-tuned for near-optimal progress. The discrepancy in performance to the uniform case is clearly visible despite the low dimensionality of the model problem. Interestingly, the resulting progress rate R_3 exceeds R_1 and R_2 on all problems.

Until now we lack a performance indicator that

- can be monitored online,
- gives feedback on how to adapt p.

Assume for a moment that the coordinate-wise progress rate R_i was such an indicator.[2] The third row in each block in table 1 illustrates the progress made with p adjusted so that all coordinate-wise progress rates R_i coincide (and therefore also equal the overall progress rate R). We refer to this setting of p as the rate-balancing strategy. As expected from the above analysis of coordinate-wise progress rates, the rate-balancing strategy is in all cases in between uniform and optimal. Notably, in all cases the overall progress R remains reasonably close to the optimal progress rate, and far higher than for uniform p.

It turns out experimentally that the rate-balancing strategy is more stable than the optimal one w.r.t. slight variations of p. However, its most relevant advantage in the present context is that it can be identified by a simple online adaptation mechanism. This mechanism is based on the observation that R_i/R is a monotonically decreasing function of p_i (with sufficiently high probability). This is simply because more frequent steps mean less progress per step. Importantly, there is no need for modeling this relation in detail.

2. This is not the case since in practice the optimal value f^* is unknown. We will show below how to circumvent this problem.

Aiming for balanced coordinate-wise progress, a straightforward online adaptation strategy is to increase p_i if $R_i > R$ and to decrease p_i if $R_i < R$. As a side effect this rule also handles an immobile coordinate i correctly that is tied to a bound by an active constraint: the progress rate $R_i = 0$ vanishes, which has the desirable effect that the corresponding p_i is reduced. Finally, the adaptive rule passes our sanity check since for symmetric couplings adjusting all progress rates to the same value is achieved by setting all p_i to the same value, and thus p to the uniform distribution.

The above adaptation rule requires the online estimation of R and R_i, as well as a (rather arbitrary) quantitative rule for changing p_i in the right direction. As an additional precaution measure we avoid decreasing p_i too close to zero. Our implementation of these rules does not keep p normalized (to sum to one) and instead keeps track of $p_{\text{sum}} = \sum_{i=1}^{n} p_i$ so that the probability p_i/p_{sum} can be recovered at constant cost. Furthermore, equality of the progress rates R_i is equivalent to equality of f-progress $\Delta = f(x^{(t)}) - f(x^{(t-1)})$ for all coordinates. In contrast to the progress rate this quantity is directly observable since f^* does not enter. It is often a cheap by-product of the CD step.

The resulting *adaptive coordinate frequencies* (ACF) method is defined in algorithm 2. For a step on coordinate i let Δ denote the progress made by this step, while $\overline{\Delta}$ denotes a reference progress rate (a running average over many steps). Then the algorithm updates p_i (and consequently p_{sum}) with a learning rate c and the reference progress rate $\overline{\Delta}$ with an independent learning rate η. The learning rates as well as the lower and upper bounds $p_{\min}$ and $p_{\max}$ are parameters of the algorithm, all of which are set to default values according to appendix-A. The exponential update of p_i in algorithm 2 has the merit of keeping p_i positive. However, the same effect can be achieved by truncation, so that many other quantitative update rules would do. At the start of the optimization run the frequencies are initialized to $p_i = 1/n$ (and $p_{\text{sum}} = 1$). $\overline{\Delta}$ is initialized to the mean over a few (e.g., $\mathcal{O}(n)$) iterations.

Algorithm 2 Adaptive Coordinate Frequencies Update

$p_{\text{new}} \leftarrow \left[\exp\left(c \cdot (\Delta/\overline{\Delta} - 1)\right) \cdot p_i\right]_{p_{\min}}^{p_{\max}}$
$p_{\text{sum}} \leftarrow p_{\text{sum}} + p_{\text{new}} - p_i$
$p_i \leftarrow p_{\text{new}}$
$\overline{\Delta} \leftarrow (1 - \eta) \cdot \overline{\Delta} + \eta \cdot \Delta$

Algorithm 3 Creation of a sequence of coordinates according to p

$I \leftarrow \{\}$
for $i \in \{1, \ldots, n\}$ **do**
  $a_i \leftarrow a_i + n \cdot p_i/p_{\text{sum}}$
  $\lfloor a_i \rfloor$ times: append index i to list I
  $a_i \leftarrow a_i - \lfloor a_i \rfloor$
end for
shuffle list I

Finally, algorithm 3 realizes coordinate selection according to p in amortized constant time. This scheme is in between deterministic and random CD in that coordinate frequencies

are chosen according to their expected value w.r.t. p (and rounded) while the order is random. The algorithm pre-specifies a number of coordinate indices like in a cyclic sweep, but in random order. The algorithm outputs a sequence of on average n and at most $2 \cdot n$ coordinates at a time at a cost of $\Theta(n)$ operations while guaranteeing that each coordinate has a waiting time of at most

$$\lceil 1/(n \cdot p_i) \rceil \leq \lceil 1/(n \cdot p_{\min}) \rceil = N < \infty$$

sweeps for its next inclusion. This property guarantees convergence of the resulting CD algorithm with the same arguments as in the proof of theorem 1 by Hsieh et al. (2008), which is based on theorem 2.1 by Luo and Tseng (1992). Alternatively, in the terminology of Tseng (2001) algorithm 3 realizes an *essentially cyclic rule* for coordinate selection. Thus, our ACF-CD algorithm enjoys the same convergence guarantees as other CD schemes, e.g., with uniform selection of coordinates.

5. Empirical Evaluation

In this section we investigate the performance of the adaptive coordinate frequencies method. We have run algorithm 1 in a number of variants reflecting the state-of-the-art in the respective fields against the ACF-CD algorithm for solving a number of instances of problem (1). We adopt the stopping criterion applied by liblinear (Fan et al., 2008). The algorithm stops as soon as all components of the gradient of the objective function that do not correspond to active constraints fall below a threshold $\epsilon > 0$.[3]

The number of CD iterations is a straightforward performance indicator. For sparse data this indicator is not always reliable, since not all coordinates correspond to non-zero components of data points equally often. In fact, for some problems the cost of the inner products $\langle w, x_i \rangle$ that dominates the cost of a CD iteration varies widely and cannot be assumed to be roughly constant. Wall clock optimization time is the more relevant measure. However, experimental timings may be biased due to implementation details. We use optimization time for exactly comparable implementations and otherwise resort to the number of multiplication and addition required to compute the inner products $\langle w, x_i \rangle$, hereafter referred to as the *number of operations*. This quantity is a very good predictor of the actual optimization time, with the advantage of being independent of implementation, CPU, memory bandwidth, etc.

The data sets used in this study range from medium sized to extremely large. They have been obtained from the libsvm data website

`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` .

Table 2 lists number of data points (instances) and number of features. The dimensionality of the optimization problem is the first one for SVM training ($n = \ell$) and the latter one for LASSO training ($n = d$).

3. The source code of all experiments is publicly available at `http://www.ini.rub.de/PEOPLE/glasmtbl/code/acf-cd/`

Problem	Instances (ℓ)	Features (d)
cover type	581,012	54
kkd-a	8,407,752	20,216,830
kkd-b	19,264,097	29,890,095
news 20	19,996	1,355,191
rcv1	20,242	47,236
url	2,396,130	3,231,961
E2006-tfidf	16,087	150,360

Table 2: Benchmark problems used in this study.

5.1. Linear SVM Training

We compared ACF-CD in an extensive experimental study to the liblinear SVM solver (Fan et al., 2008; Hsieh et al., 2008). This is an extremely strong and widely used baseline. The liblinear CD solver sweeps over random permutations of coordinates in epochs. In addition it applies a shrinking heuristic that removes bounded variables from the problem.

Both algorithms return accurate solutions to the SVM training problem. The test errors coincide exactly. The algorithms don't differ in the quality of the solution (dual objective values are extremely close; often they coincide to 10 significant digits), but only in the time it takes to compute this solution.

Comparing training times in a fair way is non-trivial. This is because the selection of a good value of the regularization parameter C requires several runs with different settings, often performed in a cross validation manner. The computational cost of finding a good value of C can easily exceed that of training the final model, and even a good range for C is hard to guess without prior knowledge. The focus of the present study is on optimization. Therefore we don't fix a specific model selection procedure and instead report training times over a range of values, namely $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$. Training times of both algorithms are comparable since we have implemented ACF-CD directly into the liblinear code.

The results are reported in figures 1 and 2. The figure also includes three-fold cross validation performance which gives an indication of which C values are most relevant. The best value is contained in the interior of the tested range in all cases. For completeness, timings and iteration numbers are listed in Appendix B.

In most cases the ACF-CD algorithm is faster than liblinear. For large values of C it can outperform the baseline by more than an order of magnitude (note the logarithmic scale in the figures). The cover type problem is an exception. This problem is special for its low feature dimensionality of only 54, which means that the 581,012 dual variables are highly redundant. In this case the optimal solution can be represented with many possible subsets of variables α_i which makes adaptation of coordinate frequencies superfluous.

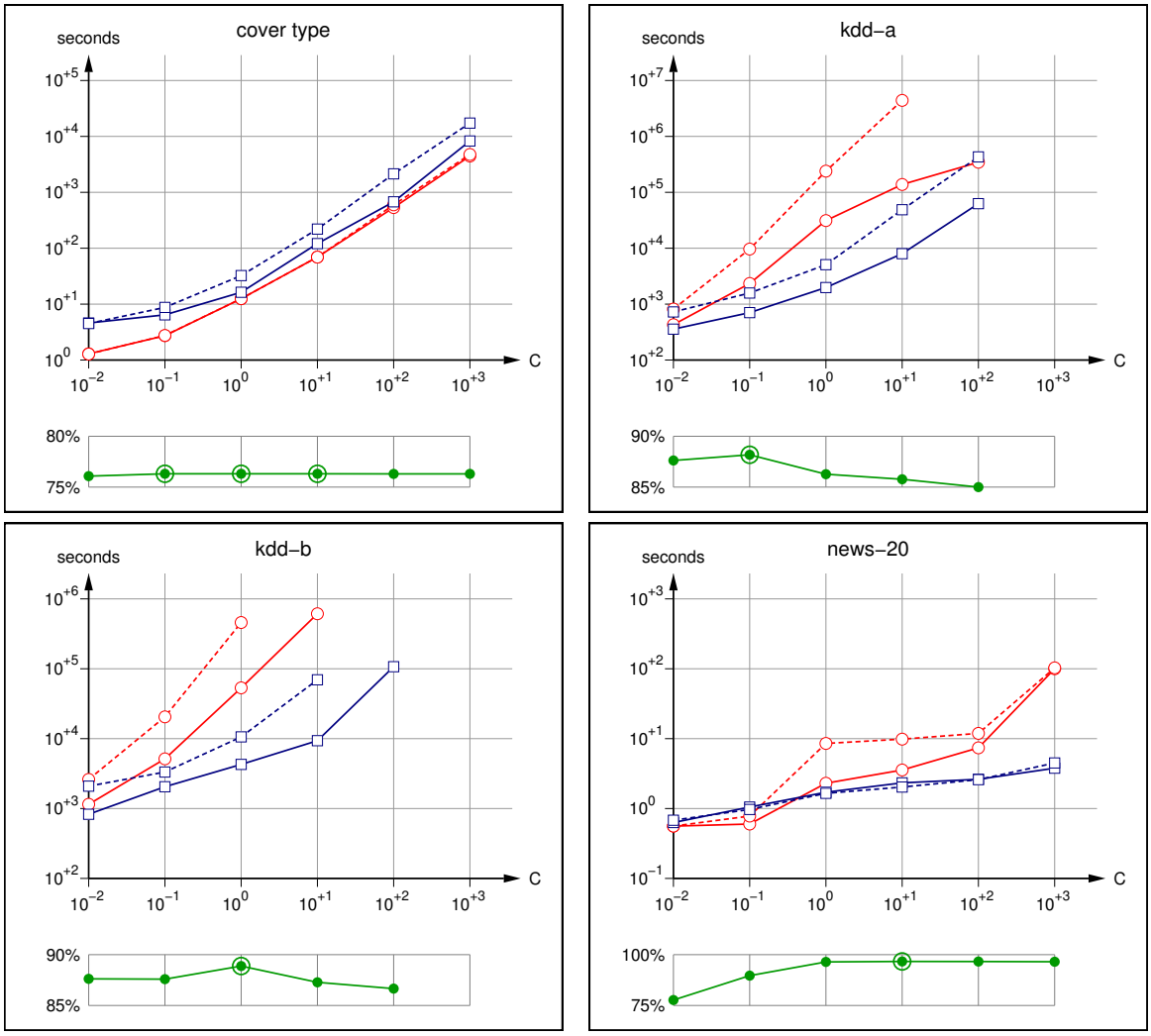

Figure 1: Training times with the original liblinear algorithm (red circles) and with ACF-CD (blue squares) as a function of the regularization parameter C. The target accuracy is $\varepsilon = 0.01$ for the solid curve and $\varepsilon = 0.001$ for the dashed curves. For reference, three-fold cross validation performance (percent correct) is plotted below the curves in green, with best configurations circled. In all cases the best value(s) are contained in the interior of the chosen parameter range.

5.2. LASSO Regression

To demonstrate the versatility of our approach we furthermore compared ACF-CD to the LASSO solver proposed by Friedman et al. (2007). This is a straightforward deterministic CD algorithm, iterating over all coordinates in order. We have varied the parameter λ in a range so that the resulting number of non-zero features varies between very few (less than 10) and many (more than $10,000$). This gives a rather complete picture of the relative

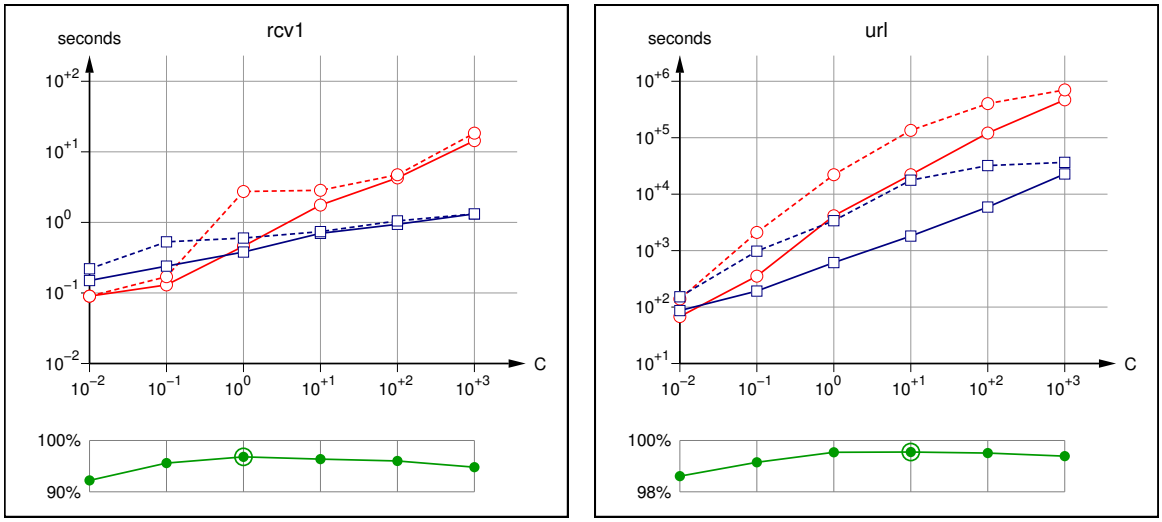

Figure 2: Further results. See figure 1 for details.

problem	λ	**uniform**		**ACF**	
		iterations	operations	iterations	operations
rcv1	0.001	$7.06 \cdot 10^8$	$2.24 \cdot 10^{10}$	$7.50 \cdot 10^7$	$4.63 \cdot 10^9$
	0.01	$9.21 \cdot 10^7$	$2.92 \cdot 10^9$	$1.86 \cdot 10^7$	$1.36 \cdot 10^9$
	0.1	$4.95 \cdot 10^7$	$1.57 \cdot 10^9$	$4.14 \cdot 10^6$	$4.43 \cdot 10^8$
	1	$2.36 \cdot 10^7$	$7.48 \cdot 10^8$	$1.53 \cdot 10^6$	$2.24 \cdot 10^8$
	10	$5.38 \cdot 10^6$	$1.71 \cdot 10^8$	$1.21 \cdot 10^6$	$1.79 \cdot 10^8$
	100	$4.25 \cdot 10^5$	$1.35 \cdot 10^7$	$2.36 \cdot 10^5$	$8.20 \cdot 10^6$
news 20	0.1	$2.64 \cdot 10^9$	$1.78 \cdot 10^{10}$	$3.88 \cdot 10^7$	$1.49 \cdot 10^9$
	1	$1.47 \cdot 10^9$	$9.89 \cdot 10^9$	$3.19 \cdot 10^7$	$7.50 \cdot 10^8$
	10	$3.78 \cdot 10^8$	$2.54 \cdot 10^9$	$2.30 \cdot 10^7$	$1.98 \cdot 10^8$
	100	$6.78 \cdot 10^6$	$4.55 \cdot 10^7$	$9.49 \cdot 10^6$	$6.42 \cdot 10^7$
E2006-tfidf	0.001	$2.38 \cdot 10^9$	$3.16 \cdot 10^{11}$	$4.08 \cdot 10^7$	$2.57 \cdot 10^{10}$
	0.01	$3.40 \cdot 10^8$	$4.51 \cdot 10^{10}$	$8.37 \cdot 10^6$	$4.02 \cdot 10^8$
	0.1	$2.59 \cdot 10^7$	$3.44 \cdot 10^9$	$5.70 \cdot 10^6$	$1.38 \cdot 10^9$
	1	$2.56 \cdot 10^6$	$3.40 \cdot 10^8$	$2.71 \cdot 10^6$	$3.75 \cdot 10^8$

Table 3: Performance of uniform CD (baseline) and the ACF-CD algorithm for LASSO training (with stopping accuracy $\epsilon = 0.001$).

performance of both algorithms over a wide range of relevant optimization problems. The results are summarized in table 3. Since LASSO is a regression method we have included the E2006-tfidf problem in the comparison.

The ACF-CD algorithm is never significantly slower than uniform CD and in some cases faster by one to two orders of magnitude, while obtaining solutions of equal quality (as indicated by the objective function value). This marks a significant speed-up of ACF-CD over uniform CD.

6. Conclusion

We have explored a new degree of freedom of coordinate descent optimization, namely (relative) coordinate frequencies. Non-uniform coordinate frequencies are advantageous on problems with constraints and with a mixture of weak and strong couplings of variables, which is arguably a property expected in many of real-world optimization problems.

We propose the Adaptive Coordinate Frequencies algorithm, an online adaptation scheme for the frequencies of coordinate selection. It adapts frequencies online so as to balance coordinate-wise progress rates. To the best of our knowledge this is the first coordinate descent algorithm with non-uniform coordinate frequencies (with the rather trivial exception of shrinking techniques). Standard CD convergence results carry over to the new technique.

The algorithm was benchmarked against strong baselines on various large scale machine learning problems. It demonstrated excellent performance with speed-ups up to a factor of 10 and more. We therefore consider adaptive coordinate frequency techniques a valuable tool for coordinate descent optimization.

References

L. Bottou and C. J. Lin. Support Vector Machine Solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, Large scale kernel machines, pages 1–28. MIT Press, 2007.

R. E. Fan, P. H. Chen, and C. J. Lin. Working Set Selection Using Second Order Information for Training Support Vector Machines. Journal of Machine Learning Research, 6:1889–1918, 2005.

R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. The Journal of Machine Learning Research, 9:1871–1874, 2008.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise Coordinate Optimization. The Annals of Applied Statistics, 1(2):302–332, 2007.

N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation, 9(2):159–195, 2001.

C. J. Hsieh, K. W. Chang, C. J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In Proceedings of the 30th International Conference on Machine learning (ICML), volume 951, pages 408–415, 2008.

T. Joachims. Making Large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods – Support Vector Learning, chapter 11, pages 169–184. MIT Press, 1998.

I. Loshchilov, M. Schoenauer, and M. Sebag. Adaptive Coordinate Descent. In N. Krasnogor, editor, Genetic and Evolutionary Computation Conference (GECCO). ACM, 2011.

Z.-Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. Journal of Optimization Theory and Applications, 72(1): 7–35, 1992.

Y. Nesterov. Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems. SIAM Journal on Optimization, 22(2):341–362, 2012.

J. C. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods – Support Vector Learning, chapter 11, pages 185–208. MIT Press, 1998.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Neural Networks, 1993., IEEE International Conference on, pages 586–591. IEEE, 1993.

T. Schaul, S. Zhang, and Y. LeCun. No More Pesky Learning Rates. In Proceedings of the 25th International Conference on Machine Learning (ICML), volume 28, pages 343–351, 2013.

S. Shalev-Shwartz and T. Zhang. Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization. The Journal of Machine Learning Research, 14:567–599, 2013.

I. Steinwart, D. Hush, and C. Scovel. Training SVMs Without Offset. The Journal of Machine Learning Research, 12:141–202, 2011.

P. Tseng. Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization. Journal of optimization theory and applications, 109(3):475–494, 2001.

G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for L1-regularized Logistic Regression. The Journal of Machine Learning Research, 13:1999–2030, 2012.

Appendix A. Appendix A

parameter	**value**
c	$1/5$
$p_{\min}$	$1/20$
$p_{\max}$	20
η	$1/n$

Default settings for the ACF algorithm parameters.

Appendix B. Appendix B

Linear SVM training results: runtime in seconds and number of iterations (small numbers below) liblinear and ACF-CF, trained for a range of values of C, with target accuracies $\varepsilon = 0.01$ (first table) and $\varepsilon = 0.001$ (second table). Runs marked with "—" did not finish after several weeks of training.[4]

Data Set	Solver	$C = 0.01$	$C = 0.1$	$C = 1$	$C = 10$	$C = 100$	$C = 1000$
cover type	liblinear	1.29	2.73	12.5	69.5	533	4,450
		$3.31 \cdot 10^6$	$7.41 \cdot 10^6$	$3.38 \cdot 10^7$	$1.80 \cdot 10^8$	$1.37 \cdot 10^9$	$1.14 \cdot 10^{10}$
	ACF	4.50	6.43	16.3	121	676	8280
		$8.92 \cdot 10^6$	$1.29 \cdot 10^7$	$3.31 \cdot 10^7$	$1.92 \cdot 10^8$	$1.49 \cdot 10^9$	$1.41 \cdot 10^{10}$
kkd-a	liblinear	429	2,340	31,200	138,000	345,000	—
		$3.07 \cdot 10^8$	$1.57 \cdot 10^9$	$1.88 \cdot 10^{10}$	$8.77 \cdot 10^{10}$	$2.35 \cdot 10^{11}$	
	ACF	357	705	1,980	7,990	62,700	—
		$2.93 \cdot 10^8$	$5.56 \cdot 10^8$	$1.49 \cdot 10^9$	$6.55 \cdot 10^9$	$5.21 \cdot 10^{10}$	
kkd-b	liblinear	1,150	5,140	53,300	612,000	—	—
		$6.92 \cdot 10^8$	$2.86 \cdot 10^9$	$3.11 \cdot 10^{10}$	$3.42 \cdot 10^{11}$		
	ACF	828	2,050	4,280	9,350	107,000	—
		$6.50 \cdot 10^8$	$1.11 \cdot 10^9$	$2.91 \cdot 10^9$	$7.13 \cdot 10^9$	$8.04 \cdot 10^{10}$	
news 20	liblinear	0.56	0.60	2.30	3.56	7.39	100
		$8.03 \cdot 10^4$	$1.22 \cdot 10^5$	$4.04 \cdot 10^5$	$6.38 \cdot 10^5$	$1.38 \cdot 10^6$	$2.47 \cdot 10^7$
	ACF	0.63	1.05	1.71	2.33	2.62	3.78
		$1.20 \cdot 10^5$	$2.03 \cdot 10^5$	$3.37 \cdot 10^5$	$3.82 \cdot 10^5$	$4.82 \cdot 10^5$	$7.37 \cdot 10^5$
rcv1	liblinear	0.09	0.13	0.46	1.76	4.27	14.1
		$9.36 \cdot 10^4$	$1.46 \cdot 10^5$	$4.77 \cdot 10^5$	$1.70 \cdot 10^6$	$4.19 \cdot 10^6$	$1.43 \cdot 10^7$
	ACF	0.15	0.24	0.38	0.70	0.94	1.32
		$1.62 \cdot 10^5$	$2.78 \cdot 10^5$	$4.62 \cdot 10^5$	$7.98 \cdot 10^5$	$1.05 \cdot 10^6$	$1.51 \cdot 10^6$
url	liblinear	67.9	353	4,140	22,100	121,000	469,000
		$4.05 \cdot 10^7$	$1.93 \cdot 10^8$	$2.22 \cdot 10^9$	$1.45 \cdot 10^{10}$	$8.04 \cdot 10^{10}$	$2.74 \cdot 10^{11}$
	ACF	86.7	192	614	1,810	5,910	22,800
		$6.24 \cdot 10^7$	$1.30 \cdot 10^8$	$4.24 \cdot 10^8$	$1.16 \cdot 10^9$	$4.34 \cdot 10^9$	$1.73 \cdot 10^{10}$

4. An outer loop iteration limit of 1000 is hard-coded into the liblinear code, version 1.9.2. We have removed this limit for the sake of a fair comparison.

Problem	Solver	$C = 0.01$	$C = 0.1$	$C = 1$	$C = 10$	$C = 100$	$C = 1000$
cover type	liblinear	1.28	2.75	12.5	69.5	597	4,750
		$3.31 \cdot 10^6$	$7.41 \cdot 10^6$	$3.38 \cdot 10^7$	$1.80 \cdot 10^8$	$1.78 \cdot 10^9$	$1.44 \cdot 10^{10}$
	ACF	4.50	8.74	32.4	220	2.140	17,300
		$8.92 \cdot 10^6$	$1.80 \cdot 10^7$	$6.45 \cdot 10^7$	$4.62 \cdot 10^8$	$4.32 \cdot 10^9$	$3.71 \cdot 10^{10}$
kkd-a	liblinear	817	9,660	239,000	4,410,000	—	—
		$1.11 \cdot 10^9$	$9.16 \cdot 10^9$	$1.59 \cdot 10^{11}$	$1.66 \cdot 10^{12}$		
	ACF	725	1,580	5,080	48,800	430,000	—
		$4.99 \cdot 10^8$	$8.90 \cdot 10^8$	$4.00 \cdot 10^9$	$3.32 \cdot 10^{10}$	$2.67 \cdot 10^{11}$	
kdd-b	liblinear	2,610	20,500	459,000	—	—	—
		$1.94 \cdot 10^9$	$1.17 \cdot 10^{10}$	$2.73 \cdot 10^{11}$			
	ACF	2,090	3,330	10,600	69,500	—	—
		$1.05 \cdot 10^9$	$1.77 \cdot 10^9$	$6.99 \cdot 10^9$	$4.28 \cdot 10^{10}$		
news 20	liblinear	0.56	0.78	8.54	9.84	11.9	103
		$8.03 \cdot 10^4$	$1.54 \cdot 10^5$	$1.55 \cdot 10^6$	$1.87 \cdot 10^6$	$2.90 \cdot 10^6$	$2.50 \cdot 10^7$
	ACF	0.68	0.97	1.65	2.03	2.59	4.49
		$1.20 \cdot 10^5$	$2.03 \cdot 10^5$	$3.37 \cdot 10^5$	$3.82 \cdot 10^5$	$4.82 \cdot 10^5$	$8.80 \cdot 10^5$
rcv1	liblinear	0.09	0.17	2.74	2.85	4.73	18.4
		$9.40 \cdot 10^4$	$1.93 \cdot 10^5$	$3.36 \cdot 10^6$	$3.36 \cdot 10^6$	$5.63 \cdot 10^6$	$2.14 \cdot 10^7$
	ACF	0.22	0.53	0.60	0.74	1.05	1.32
		$2.64 \cdot 10^5$	$6.09 \cdot 10^5$	$7.33 \cdot 10^5$	$9.14 \cdot 10^5$	$1.30 \cdot 10^6$	$1.70 \cdot 10^6$
url	liblinear	139	2,100	22,100	135,000	402,000	703,000
		$8.27 \cdot 10^7$	$1.18 \cdot 10^9$	$1.46 \cdot 10^{10}$	$7.61 \cdot 10^{10}$	$2.35 \cdot 10^{11}$	$3.78 \cdot 10^{11}$
	ACF	152	978	3,390	17,700	32,100	36,600
		$9.66 \cdot 10^7$	$5.92 \cdot 10^8$	$2.24 \cdot 10^9$	$9.76 \cdot 10^9$	$2.34 \cdot 10^{10}$	$2.25 \cdot 10^{10}$