Statistics and Its Interface Volume 6 (2013) 167–173

A cocktail algorithm for solving the elastic net penalized Cox's regression in high dimensions

Yi Yang* and Hui Zou†,‡

We introduce a cocktail algorithm, a good mixture of coordinate decent, the majorization-minimization principle and the strong rule, for computing the solution paths of the elastic net penalized Cox's proportional hazards model. The cocktail algorithm enjoys a proven convergence property. We have implemented the cocktail algorithm in an R package `fastcox`. Numerical examples show that `cocktail` is comparable to `coxnet` [11] in speed and often delivers better quality solutions.

Keywords and phrases: Cox's model, Coordinate descent, Elastic Net, MM principle, Strong rule.

1. INTRODUCTION

Cox's proportional hazards model [2] is a standard statistical model for studying the relation between survival time and a set of covariates. Consider the standard survival data of the form $(y_i, \mathbf{x}_i, d_i)_{i=1}^n$, where y_i is the survival time, $1-d_i$ is the censoring indicator ($d_i = 1$ indicates no censoring and $d_i = 0$ indicates right censoring) and $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^\intercal$ represents the p-dimension covariates. For the sake of simplicity, assume there are no tied failure times and the censoring is non-informative. Denote by $t_1 < t_2 < \cdots < t_S$ the distinct failure times and let R_s be the risk set at time $t_s - 0$. Cox's proportional hazards model assumes that the hazard function at time t given predictor values $\mathbf{x}$ is $h(t|\mathbf{x}) = h_0(t)\exp(\mathbf{x}^\intercal\boldsymbol{\beta}^*)$ where $h_0(t)$ represents the baseline hazard function. Statistical inference of Cox's model is through the partial likelihood function:

$$L(\boldsymbol{\beta}) = \prod_{s=1}^{S} \frac{\exp(\mathbf{x}_{i_s}^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})},$$

where i_s is the index of the failure at time t_s. The usual Cox's estimator of $\boldsymbol{\beta}$ is obtained by maximizing the partial likelihood.

With the advances in modern technology, high-dimensional data frequently appear in fields such as medical and biological sciences, finances and economics, etc. The maximum partial likelihood estimator does not work well in the presence of high-dimensional covariates. To combat the high-dimensionality, sparse penalized Cox's models have been considered in the literature. Let $P_\lambda(\boldsymbol{\beta})$ be a penalty function that is non-differentiable at zero. Consider the penalized partial likelihood estimator

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \frac{1}{n}\left[\sum_{s=1}^{S} -\mathbf{x}_{i_s}^\intercal\boldsymbol{\beta} + \log\left(\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})\right)\right] + P_\lambda(\boldsymbol{\beta}).$$

For example, Tibshirani [13] used the lasso penalty [12], $P_\lambda(\boldsymbol{\beta}) = \lambda\sum_{i=j}^{p}|\beta_j|$, to fit a lasso penalized Cox's regression model.

Zou and Hastie [19] proposed the elastic net penalty as an improved variant of the lasso for high-dimensional data. The elastic net penalty is defined as

$$P_{\lambda,\alpha}(\boldsymbol{\beta}) = \sum_{j=1}^{p}\lambda\left(\alpha|\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2\right),$$

with $\lambda > 0$ and $0 < \alpha \le 1$. The ℓ_1 part of the elastic net is responsible for achieving sparsity. The lasso can be viewed as a special case of the elastic net with $\alpha = 1$. By using its quadratic part, the elastic net improves upon the lasso in two aspects. First, it can better handle the correlated covariates which are very common in high-dimensional data. Second, the solution paths are more stable due to the quadratic regularization and hence it leads to improved prediction. Park and Hastie [10] developed a predictor-corrector algorithm for computing the elastic net penalized Cox's model, see the `glmpath` package available from the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/package=glmpath.

Goeman [7] discussed a gradient descent algorithm for solving the ℓ_1 penalized Cox's model and implemented his algorithm in an R package `penalized`, available from CRAN at http://cran.r-project.org/web/packages/penalized. Simon et al. [11] recently developed `coxnet` for computing the elastic net penalized Cox model. Their function is included in the `glmnet` package, available from CRAN at http://cran.r-project.org/package=glmnet. By numerical examples, Simon et al. [11] showed that `coxnet` is much faster than `coxpath` and `penalized`.

It is important to note that `coxnet` uses some heuristic arguments to approximate the Hessian matrix of the log-partial likelihood in order to boost computation speed. As

*Ph.D candidate.
†Corresponding author.
‡Associate Professor. The authors thank the editor and referees for their helpful comments.

a result, it is unclear whether `coxnet` always converges to the right solution, although Simon et al. [11] reported that they did not encounter any convergence problem. In this paper, we introduce a new principled fast algorithm for computing the elastic net penalized Cox model. Our algorithm combines the strengths of three optimization ideas: coordinate descent, the majorization-minimization principle and the strong rule. It is thus named a cocktail algorithm. We show that the cocktail algorithm always converges to the right solution. We build an R package `fastcox` to implement the cocktail algorithm and we show that `cocktail` is comparable to `coxnet` in speed and often gives higher quality solutions.

In section 2 we show how to combine the majorization-minimization principle and coordinate descent into a new coordinate-majorization-descent algorithm (CMD). In section 3 we further show how to integrate the strong rule and the CMD algorithm, which leads to the final cocktail algorithm for computing the solution paths of the elastic net penalized Cox's regression. Numerical examples are presented in section 4.

2. COORDINATE MAJORIZATION DESCENT

2.1 Derivation

In this section we derive the coordinate-majorization-descent (CMD) algorithm to minimize the following objective function

$$G(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}) + \sum_{j=1}^{p} \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right], \tag{1}$$

where

$$\ell(\boldsymbol{\beta}) = n^{-1} \sum_{s=1}^{S} -\mathbf{x}_{i_s}^{\intercal} \boldsymbol{\beta} + \log \left(\sum_{i \in R_s} \exp(\mathbf{x}_i^{\intercal} \boldsymbol{\beta}) \right).$$

Note that including the non-negative weights w_js allows for more flexible estimation. If we want to always include x_j in the final model then we typically do not impose a sparse penalty on β_j, which can be easily done by setting $w_j = 0$. Often, the adaptively weighted lasso [18] is preferred over the lasso for variable selection.

We begin with the observation that $\ell(\boldsymbol{\beta})$ is a smooth convex function of $\boldsymbol{\beta}$ and the penalty function is convex and separable. This observation suggests that we can try the coordinate descent algorithm for minimizing the objective function in (1) [15]. Recently, coordinate descent has been successfully used to solve the lasso-type penalized models [4, 5, 16]. Define the objective function for fixed λ and α and β_k where $k \neq j$ to be

$$g(\beta_j) = \ell(\beta_j | \beta_k = \tilde{\beta}_k, k \neq j) + \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right], \tag{2}$$

the coordinate descent algorithm proceeds as follows:

- Initialize $\tilde{\boldsymbol{\beta}}$.
- Cyclic coordinate descent: for $j = 1, \ldots, p$, update the estimator by
$$\tilde{\beta}_j \leftarrow \arg\min_{\beta_j} g(\beta_j).$$
- Repeat the coordinate descent cycle till convergence.

Coordinate descent is efficient for the ℓ_1-penalized least squares because each coordinate descent update can be computed by a simple soft-thresholding rule. However, the univariate minimization problem in (2) does not have a simple closed-form solution. The same computational difficulty appears in many applications of the Expectation-Maximization algorithm where the maximization step does not have a simple computational form. To alleviate such difficulty, Dempster, Laird and Rubin [3] proposed to increase the objective function rather than maximize it at each maximization step, which results in the generalized Expectation-Maximization algorithm. We borrow the same idea to overcome the computational difficulty in (2). Our solution makes use of the majorization-minimization/maximization (MM) principle. For some good review papers on the MM principle, the readers are referred to Lange, Hunter and Yang [9], Hunter and Lange [8] and Wu and Lange [17].

Instead of minimizing (2), we propose to find an update of $\tilde{\beta}_j$ such that the univariate function in (2) is decreased. It turns out that such an update can be computed by a soft-thresholding rule. To write the updating formula we need some additional notation. Define

$$D_j = \sum_{s=1}^{S} \frac{1}{4n} \left(\max_{i \in R_s}(x_{ij}) - \min_{i \in R_s}(x_{ij}) \right)^2,$$

for $j = 1, 2, \ldots, p$, and let $\ell_j'(\boldsymbol{\beta})$ denote the partial derivative of the negative log partial likelihood with respect to β_j. We have

$$\ell_j'(\boldsymbol{\beta}) = n^{-1} \sum_{s=1}^{S} \left[-x_{i_s,j} + \frac{\sum_{i \in R_s} x_{i,j} \exp(\mathbf{x}_i^{\intercal} \boldsymbol{\beta})}{\sum_{i \in R_s} \exp(\mathbf{x}_i^{\intercal} \boldsymbol{\beta})} \right],$$

for $j = 1, 2, \ldots, p$. We approximate (2) by a penalized quadratic function defined as

$$q(\beta_j | \tilde{\boldsymbol{\beta}}) = \ell(\tilde{\boldsymbol{\beta}}) + \ell_j'(\tilde{\boldsymbol{\beta}})(\beta_j - \tilde{\beta}_j) + \frac{D_j}{2}(\beta_j - \tilde{\beta}_j)^2 + \lambda \left[\alpha w_j |\beta_j| + \frac{1}{2}(1-\alpha)\beta_j^2 \right]. \tag{3}$$

The proposed update is the minimizer of (3)

$$\tilde{\beta}_j^{\text{new}} = \frac{S(D_j \tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda \alpha w_j)}{D_j + \lambda(1-\alpha)},$$

where $S(z,t) = (|z| - t)_+ \text{sign}(z)$ is the soft-thresholding operator.

Algorithm 1 CMD for the elastic net penalized Cox's regression

1. Initialize $\tilde{\boldsymbol{\beta}}$.
2. For $j = 1, \ldots, p$,
$$\tilde{\beta}_j^{\text{new}} = \frac{S(D_j\tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda\alpha w_j)}{D_j + \lambda(1-\alpha)}.$$
3. Update $\tilde{\boldsymbol{\beta}} = \tilde{\boldsymbol{\beta}}^{\text{new}}$.
4. Repeat steps 2–3 till convergence of $\tilde{\boldsymbol{\beta}}$.

With the help of the MM principle, we can use an iterative soft-thresholding procedure, see Algorithm 1, for solving the elastic net penalized Cox's regression, which is much like the iterative soft-thresholding procedure for the elastic net penalized least squares problem.

2.2 The descent property of CMD

We now prove the descent property of Algorithm 1 which can be seen from the following two lemmas.

Lemma 1. $\tilde{\beta}_j^{\text{new}} = \frac{S(D_j\tilde{\beta}_j - \ell_j'(\tilde{\boldsymbol{\beta}}), \lambda\alpha w_j)}{D_j + \lambda(1-\alpha)}$ *minimizes* $q(\beta_j|\tilde{\boldsymbol{\beta}})$ *defined in* (3).

Lemma 2. $\ell_j'' \le D_j$ *for all* $\boldsymbol{\beta} \in \mathbb{R}^p$.

Lemma 1 basically shows the soft-thresholding solution to the univariate lasso regression. We only prove Lemma 2.

Proof of lemma 2. We first compute

$$\ell_j'' = \frac{1}{n}\sum_{s=1}^{S}\left[\frac{\sum_{i\in R_s} x_{i,j}^2 \exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})} - \left(\frac{\sum_{i\in R_s} x_{i,j}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}\right)^2\right].$$

Inside $[\cdots]$ can be regarded as the variance of a discrete random variable Z whose distribution is $P(Z = x_{i,j}) = \frac{\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}{\sum_{i\in R_s}\exp(\mathbf{x}_i^\intercal\boldsymbol{\beta})}$. The variance is maximized when Z has a two-point distribution on $\max_{i\in R_s}(x_{ij})$, $\min_{i\in R_s}(x_{ij})$ with equal probability. □

From Lemmas 1 and 2 we have the following inequalities

$$q(\tilde{\beta}_j^{\text{new}}|\tilde{\boldsymbol{\beta}}) \le q(\tilde{\beta}_j|\tilde{\boldsymbol{\beta}}),$$
$$g(\beta_j) \le q(\beta_j|\tilde{\boldsymbol{\beta}}) \quad \forall\beta_j \in \mathbb{R}.$$

Therefore, we can conclude

$$g(\tilde{\beta}_j^{\text{new}}) \le q(\tilde{\beta}_j^{\text{new}}|\tilde{\boldsymbol{\beta}}) \le q(\beta_j|\tilde{\boldsymbol{\beta}}) = g(\beta_j),$$

which justifies the descent property of the CMD algorithm.

2.3 Solution path implementation

For each given α, we use the CMD algorithm to compute the solutions of the elastic net penalized Cox's regression model at a grid of decreasing λ values. The default number is 100. We use the commonly used warm-start and active set tricks, as done in `glmnet`.

To use warm-start we first need to find the largest λ value, denoted by $\lambda_{\max}$, that is defined as the smallest λ that shrinks all β_js to zero. By the KKT conditions it is easy to show for $w_j \neq 0$

$$\lambda_{\max} = n^{-1}\alpha^{-1}\max_j\left\{\frac{1}{|w_j|}\left|\sum_{s=1}^{S}(-x_{i_s,j} + \frac{\sum_{i\in R_s} x_{i,j}}{|R_s|})\right|\right\}.$$

For the ordinary $n \ge p$ data we let $\lambda_{\min} = 0.0001\lambda_{\max}$. When $p > n$ we let $\lambda_{\min} = 0.01\lambda_{\max}$. We choose a grid of 100 points uniformly in the log scale on $(\lambda_{\min}, \lambda_{\max})$. Let $\lambda[1] = \lambda_{\max}$ and $\lambda[100] = \lambda_{\min}$. Warm-start takes the solution at the k-th grid point $\lambda[k]$ as the initial value for computing the solution at $\lambda[k+1]$.

The active set is defined as the collection of variables whose current coefficient estimates are nonzeros. After a complete cycle through all p coefficients, we only repeat the coordinate descent on the active set till convergence. Then we run another complete cycle to check whether the active set is changed. If the active set remains unchanged, the algorithm is done, otherwise the process is repeated.

3. CMD WITH THE STRONG RULE

Tibshirani et al. [14] recently introduced the strong rule for improving the computational speed of `glmnet`. To implement CMD in the `fastcox` package, we have also used the strong rule on top of the CMD algorithm. Suppose that we have already computed the solution $\hat{\boldsymbol{\beta}}[\lambda_k]$ at $\lambda = \lambda_k$, before computing the solution at λ_{k+1}, we first compute $\ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k])$ for $j = 1, 2, \ldots, p$. The strong rule claims that the variables that satisfy the condition

$$\left|\ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k])\right| \ge \alpha(2\lambda_{k+1} - \lambda_k)w_j$$

are very likely to have nonzero coefficients at λ_{k+1}. Let V be the collection of such variables and write V^c as its complement. If the strong rule guesses correctly, we only need to focus on solving $\hat{\boldsymbol{\beta}}_V$ by calling Algorithm 1 on a reduced dataset $(y_i, \mathbf{x}_{iV}, d_i)_{i=1}^n$, where $\mathbf{x}_{iV} = (\cdots\mathbf{x}_{i,j}\cdots)$ and j is an index in the set V. Suppose that $\hat{\boldsymbol{\beta}}_V$ is computed and the next step is to check whether the strong rule indeed guesses correctly. For that we just need to check whether $\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$ satisfies the KKT condition at $\lambda = \lambda_{k+1}$. In other words, if for each $j \in V^c$ the following KKT condition holds:

$$\left|\ell_j'\left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})\right)\right| \le \alpha\lambda_{k+1}w_j.$$

Then $\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$ is the solution at $\lambda = \lambda_{k+1}$. Otherwise, we update V by $V = V\bigcup U$ where

$$U = \{j : j \in V^c \text{ and } \left|\ell_j'\left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})\right)\right| > \alpha\lambda_{k+1}w_j\}.$$

Algorithm 2 The CMD algorithm with the strong rule for the solution at λ_{k+1}

1. Initialize $\tilde{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}[\lambda_k]$.
2. Screen p variables using the strong rule, create an initial survival set V such that for $j \in V$
$$\left|\ell_j'(\hat{\boldsymbol{\beta}}[\lambda_k])\right| \geq \alpha(2\lambda_{k+1} - \lambda_k)w_j.$$
3. Call Algorithm 1 on a reduced dataset $(y_i, \mathbf{x}_{iV}, d_i)_{i=1}^n$ to solve $\hat{\beta}_V$.
4. Compute a set U as the part of V^c that failed KKT check:
$$U = \{j : j \in V^c \text{ and } \left|\ell_j'\left(\boldsymbol{\beta} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})\right)\right| > \alpha\lambda_{k+1}w_j\}.$$
5. If $U = \varnothing$ then stop the loop and return $\hat{\boldsymbol{\beta}} = (\hat{\boldsymbol{\beta}}_V, \mathbf{0})$. Otherwise update $V = V \bigcup U$ and go to step 3.

Note that the V set can only grow larger after each update and hence the strong rule iteration will always stop after a finite number of updates, which means the strong rule will eventually guess correctly.

For penalized least squares and logistic regression, the strong rule has a magic property in that it almost always guesses correctly and we rarely need to actually update V by adding U. See Tibshirani et al. [14] for more detailed discussion. We have observed that this incredible phenomenon continues to hold for the penalized Cox regression model. Algorithm 2 shows how we use the strong rule on top of the CMD algorithm. Algorithm 2 is indeed the pseudocode of `cocktail`.

4. NUMERICAL STUDIES

We have implemented the CMD algorithm with the strong rule in a publicly available R package `fastcox`. Simon et al. [11] have showed that `coxnet` is much faster than `coxpath` and `penalized`. Hence we only compare `cocktail` with `coxnet` which is a part of R package `glmnet` 1.7.1. All computations were carried out on an 2.4GHz Intel Core i5 processor. In `coxnet`, the convergence criterion is $\max_j D_j(\tilde{\beta}_j^{\text{old}} - \tilde{\beta}_j^{\text{new}})^2 < \epsilon^2$. We used the same convergence criterion in `cocktail` and $\epsilon = 10^{-5}$ in all examples presented in this section.

4.1 Timing comparison

We considered the simulation model by Simon et al. [11]. We generated data with n observations and p predictors from the following model

$$Y^{true} = \exp\left(\sum_{j=1}^{p} X_j\beta_j + k \cdot N(0,1)\right),$$

where Y^{true} is the "true" survival time and the correlation between predictors X_j and $X_{j'}$ is ρ, with ρ ranges from zero to 0.95, and $\beta_j = (-1)^j \exp(-(2j-1)/20)$, k is chosen such that the signal-to-noise ratio is 3.0. Likewise, censoring times are generated by $C = \exp(k \cdot N(0,1))$. The recorded survival time is $Y = \min(Y^{true}, C)$. The observation is censored if $C < Y^{true}$.

Table 1. Timings (in seconds) for `coxnet` *and* `cocktail`. *Total time for the same λ sequence of 100 values, averaged over 20 independent runs*

$n = 100, p = 5,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
`coxnet`	23.64	21.74	23.80	25.93	16.89	15.17
`cocktail`	11.35	9.91	12.78	18.89	26.97	30.44
$\alpha = 0.2$						
`coxnet`	21.29	22.83	21.05	24.42	14.67	13.07
`cocktail`	9.41	12.15	11.05	19.52	20.73	27.77
$\alpha = 0.5$						
`coxnet`	18.11	17.12	17.24	16.94	13.19	10.15
`cocktail`	11.98	13.52	13.61	17.66	23.08	18.42
$\alpha = 0.8$						
`coxnet`	10.42	10.86	10.59	13.26	11.16	16.61
`cocktail`	13.14	15.48	15.88	22.27	25.96	24.55
$\alpha = 1$						
`coxnet`	9.55	9.27	9.56	10.73	11.06	17.95
`cocktail`	35.06	31.19	28.75	31.77	35.19	44.40

In Table 1, $n = 100$, $p = 5,000$, for each α in $(0.1, 0.2, 0.5, 0.8, 1)$, we computed the solution paths of the penalized Cox's model for the same sequence of λ values using `coxnet` and `cocktail`. We repeated the process 20 times on 3 independent data sets and reported the average running time. We also performed a similar timing comparison for $n = 200$, $p = 10,000$ in Table 2. We see that `cocktail` has better speed performance with small α and low correlation data while `coxnet` is faster for large α and high correlation data. When `cocktail` is the winner, it can be 2 times faster than `coxnet` and vice versa. Thus, it is fair to say that both packages are comparable in speed.

4.2 Quality comparison

We show that with the same convergence criterion, `cocktail` can provide a more accurate solution than `coxnet` does. We test the accuracy of solutions by checking their KKT conditions. Theoretically, $\boldsymbol{\beta}$ is the solution to (1) if and only if the following conditions hold

$$\begin{aligned} \ell_j'(\boldsymbol{\beta}) + \lambda(1-\alpha)\beta_j + \alpha\lambda w_j \cdot \text{sgn}(\beta_j) = 0 \qquad & \text{if } \beta_j \neq 0, \\ |\ell_j'(\boldsymbol{\beta})| \leq \alpha\lambda w_j \qquad & \text{if } \beta_j = 0, \end{aligned}$$

where $j = 1, 2, \ldots, p$. Numerically, we declare β_j passes the KKT condition check if

$$\begin{aligned} |\ell_j'(\boldsymbol{\beta}) + \lambda(1-\alpha)\beta_j + \alpha\lambda w_j \cdot \text{sgn}(\beta_j)| \leq \epsilon \qquad & \text{if } \beta_j \neq 0, \\ |\ell_j'(\boldsymbol{\beta})| \leq \alpha\lambda w_j + \epsilon \qquad & \text{if } \beta_j = 0, \end{aligned}$$

Table 2. Timings (in seconds) for `coxnet` *and* `cocktail`*. Total time for the same λ sequence of 100 values, averaged over 20 independent runs*

$n = 200, p = 10,000$						
Correlation	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
`coxnet`	96.5	70.9	79.7	113.6	77.9	42.7
`cocktail`	45.6	45.0	51.1	71.7	90.9	138.0
$\alpha = 0.2$						
`coxnet`	94.1	89.0	114.5	77.4	44.6	49.2
`cocktail`	35.7	43.5	64.8	62.6	69.3	124.3
$\alpha = 0.5$						
`coxnet`	52.6	70.4	65.3	62.5	49.3	52.1
`cocktail`	45.2	59.9	51.9	78.0	102.4	133.6
$\alpha = 0.8$						
`coxnet`	38.9	41.3	35.8	39.8	32.1	57.8
`cocktail`	54.1	66.6	74.4	69.2	76.0	76.0
$\alpha = 1$						
`coxnet`	39.5	40.4	42.8	52.9	30.3	48.5
`cocktail`	150.2	135.2	127.8	224.6	145.9	164.6

for a small $\epsilon > 0$. For the solutions computed in section 4.1, we calculated the average number of coefficients that violated the KKT condition check at each λ value. Then this number was averaged over the 100 values of λs. This process was repeated 3 times on 3 independent datasets. As shown in Table 3 and Table 4, `cocktail` has much smaller violation counts than `coxnet` in most cases. Only for $\alpha = 1$, which corresponds to the lasso case, `coxnet` is slightly better than `cocktail`, but the KKT violation counts are very small in this case. Overall, it is clear that `cocktail` is numerically more accurate than `coxnet`.

4.3 Real data analysis

In this section we use the lung cancer data from Beer et al. [1] to examine timings and accuracies of `coxnet` and `cocktail`. The data is from a microarray experiment investigating survival of cancer patients with lung adenocarcinomas. The data set contains expression data for 86 patients with 7,129 probe sets. We chose α from $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)$ and computed the solution paths of the penalized Cox's model for the same λ sequence using `coxnet` and `cocktail`. The process was repeated 20 times. For space consideration we only report the timing and accuracy results of $\alpha = 0.1, 0.2, 0.3, 0.5, 0.8, 1$ in Table 5. One can see that the two algorithms are comparable in terms of the running time: `cocktail` has better speed performance with small α while `coxnet` is faster with large α. In terms of solution quality which is measured by the average number of coefficients that violated the KKT condition check, `cocktail` is always the winner. This is consistent with the simulation results.

In Figure 1 we also plot the solution paths of the tuned elastic net penalized Cox's model for the lung cancer data.

Table 3. Reported numbers are the average number of coefficients among 5,000 coefficients that violated the KKT condition check (rounded down to the next smaller integer) using `coxnet` *and* `cocktail`*. Results are averaged over the λ sequence of 100 values and averaged over 20 independent runs*

$n = 100, p = 5,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
`coxnet`	556	497	517	434	285	153
`cocktail`	6	11	19	59	136	116
$\alpha = 0.2$						
`coxnet`	313	325	292	233	185	98
`cocktail`	6	5	12	72	90	76
$\alpha = 0.5$						
`coxnet`	146	141	124	102	88	54
`cocktail`	5	6	9	25	51	40
$\alpha = 0.8$						
`coxnet`	50	50	36	23	41	32
`cocktail`	5	6	7	13	28	28
$\alpha = 1$						
`coxnet`	5	2	4	7	20	24
`cocktail`	7	6	6	12	20	24

Table 4. Reported numbers are the average number of coefficients among 10000 coefficients that violated the KKT condition check (rounded down to the next smaller integer) using `coxnet` *and* `cocktail`*. Results are averaged over the λ sequence of 100 values and averaged over 20 independent runs*

$n = 200, p = 10,000$						
ρ	0	0.1	0.2	0.5	0.8	0.95
$\alpha = 0.1$						
`coxnet`	921	729	731	669	573	197
`cocktail`	12	14	107	136	227	140
$\alpha = 0.2$						
`coxnet`	471	527	545	404	247	129
`cocktail`	7	12	61	95	93	95
$\alpha = 0.5$						
`coxnet`	203	251	184	123	170	80
`cocktail`	3	11	20	66	88	62
$\alpha = 0.8$						
`coxnet`	90	69	42	60	82	47
`cocktail`	10	14	13	29	48	38
$\alpha = 1$						
`coxnet`	1	2	3	24	38	37
`cocktail`	9	12	13	30	44	42

One can see that the two solution path plots are virtually identical. We did a 2-dimension search using 5-fold cross-validation to find the best pair of (α, λ) that incurs maximal log partial likelihood. The fitted penalized Cox model selected 39 genes. It took `cocktail` 4.1 seconds to complete the solution path calculation, while it took `coxnet` 5.3 sec-

Table 5. Timings (in seconds) and KKT check for `coxnet` *and* `cocktail` *for the lung cancer data from Beer et al. [1]. Reported values for KKT check are the average number of coefficients among 7,129 coefficients that violated the KKT conditions (rounded down to the next smaller integer) using* `coxnet` *and* `cocktail`*. Total time for the same λ sequence, averaged over 20 runs*

Lung cancer ($n = 86, p = 7,129$)						
α	0.1	0.2	0.3	0.5	0.8	1
Timings						
`coxnet`	6.30	5.33	5.31	4.57	4.42	5.49
`cocktail`	4.14	3.60	4.10	4.30	5.58	7.77
KKT Check						
`coxnet`	42	20	10	1	0	0
`cocktail`	0	1	1	0	0	0

onds to get the results.

5. DISCUSSION

By combining the strengths of the MM principle, cyclic coordinate descent and the strong rule, we have derived a fast cocktail algorithm for solving the elastic net penalized Cox's model. Our algorithm is comparable in speed to the fastest software for solving the elastic net penalized Cox's model in the literature.

It is attempting to directly apply the Newton-Raphson algorithm to solve the penalized Cox's model, as done for the penalized logistic regression model in Friedman, Hastie and Tibshirani [4]. The difficulty is, as pointed out in Simon et al. [11], the computation of the Hessian matrix of the log partial likelihood is very expensive in the Newton-Raphson loop. To avoid the difficulty Simon et al. [11] only computed the diagonals of the Hessian matrix with respect to $\mathbf{x}^\intercal\boldsymbol{\beta}$ and pretended the Hessian matrix is a diagonal matrix. However, it is unclear whether their treatment always guarantees `coxnet` to converge to the right solution. It is important to point out that even when we compute the exact Hessian matrix in each Newton-Raphson iteration, the Newton-Raphson update does not always guarantee convergence. A more careful Newton-Raphson algorithm involves step-size adjustment by techniques such as trust-region methods Genkin, Lewis and Madigan [6]. Such issues do not exist in the cocktail algorithm. Thanks to the MM principle, the cocktail algorithm has a proven convergence property. The R package `fastcox` that implements our cocktail algorithm is available on CRAN and `http://code.google.com/p/fastcox/`.

Received 14 December 2011

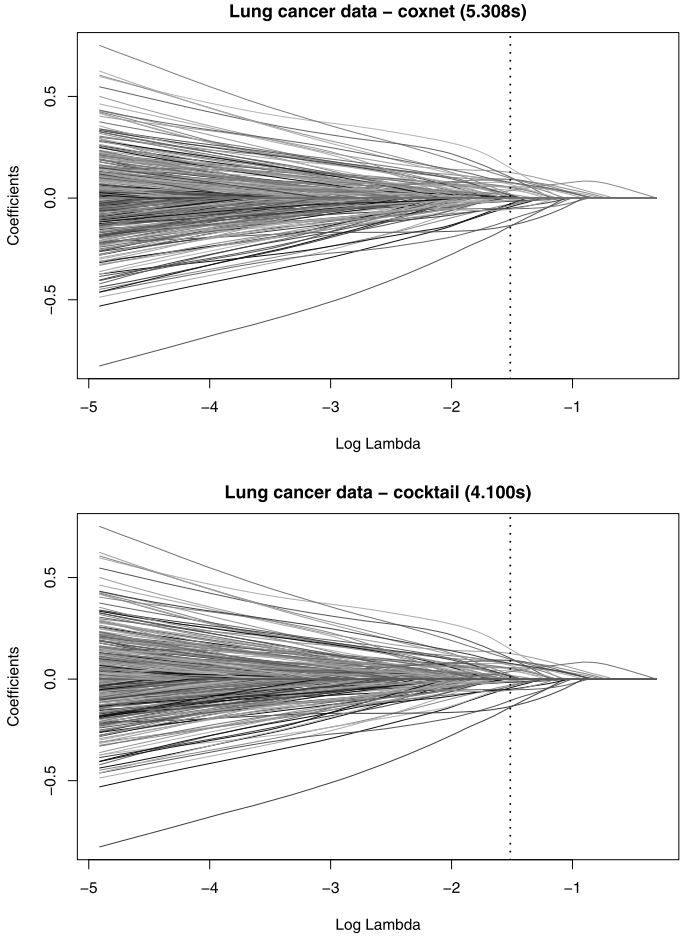

Figure 1. Solution paths and timings of the elastic net penalized Cox's model on the lung cancer data from Beer et al. [1] with 86 observations and 7,129 predictors. The top panel shows the solution paths computed by `coxnet` *in 5.308 seconds; the bottom panel shows the solution paths computed by* `cocktail` *in 4.100 seconds. The optimal $(\alpha, \log(\lambda))$ pair is $(0.3, -1.51)$. The elastic net penalized Cox's model, which is indicated by the vertical dotted line, selects 39 genes.*

REFERENCES

[1] Beer, D. G., Kardia, S. L. R., Huang, C. C., Giordano, T. J., Levin, A. M., Misek, D. E., Lin, L., Chen, G., Gharib, T. G., Thomas, D. G. et al. (2002). Gene-expression profiles predict survival of patients with lung adenocarcinoma. *Nature Medicine* **8** 816–824.

[2] Cox, D. (1972). Regression models and life tables (with discussion). *Journal of the Royal Statistical Society, Series B* **74** 187–220. MR0341758

[3] Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977). Maximum likelihood form incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B* **39** 1–38. MR0501537

[4] Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularized paths for generalized linear models via coordinate descent. *Journal of Statistical Software* **33** 1–22.

[5] Friedman, J., Hastie, T., Hoefling, H. and Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics* **2**(1) 302–322. MR2415737

[6] Genkin, A., Lewis, D. and Madigan, D. (2007). Large-scale bayesian logistic regression for text categorization. *Technometrics* **49**(3) 291–304. MR2408634

[7] Goeman, J. J. (2010). L1 penalized estimation in the cox proportional hazards model. *Biometrical Journal* **52** 70–84. MR2756594
[8] Hunter, D. R. and Lange, K. (2004). A tutorial on MM algorithms. *The American Statistician* **58** 30–37. MR2055509
[9] Lange, K., Hunter, D. and Yang, I. (2000). Optimization transfer using surrogate objective functions (with discussion). *Journal of Computational and Graphical Statistics* **9** 1–20. MR1819865
[10] Park, M. Y. and Hastie, T. (2007). L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **69** 659–677. MR2370074
[11] Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011). Regularization paths for Cox's proportional Hazards model via coordinate descent. *Journal of Statistical Software* **39** 1–13.
[12] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B* **58** 267–288. MR1379242
[13] Tibshirani, R. (1997). The lasso method for variable selection in the Cox model. *Statistics in Medicine* **16** 385–395.
[14] Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, R. J. (2010). Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.
[15] Tseng, P. (2001). Convergence of block coordinate descent method for nondifferentiable maximization. *Journal of Optimization Theory and Applications* **109**(3) 474–494. MR1835069
[16] Wu, T. T. and Lange, K. (2008). Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics* **2** 224–244. MR2415601
[17] Wu, T. T. and Lange, K. (2010). The MM alternative to EM. *Statistical Science* **25** 492–505. MR2807766
[18] Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association* **101** 1418–1429. MR2279469
[19] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B* **67** 301–320. MR2137327

Yi Yang
School of Statistics
University of Minnesota
Minneapolis, MN 55455
USA
E-mail address: yiyang@umn.edu

Hui Zou
School of Statistics
University of Minnesota
Minneapolis, MN 55455
USA
E-mail address: zouxx019@umn.edu