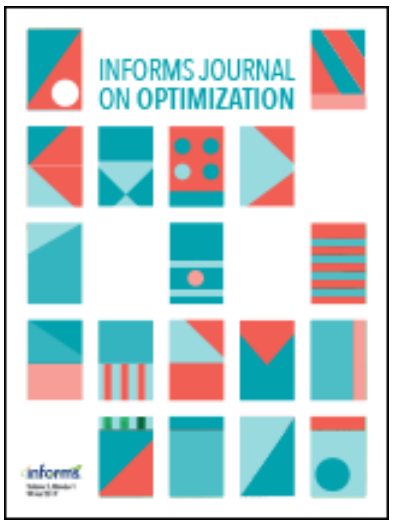


INFORMS Journal on Optimization

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

REPR: Rule-Enhanced Penalized Regression

Jonathan Eckstein, Ai Kagawa, Noam Goldberg

To cite this article:
Jonathan Eckstein, Ai Kagawa, Noam Goldberg (2019) REPR: Rule-Enhanced Penalized Regression. INFORMS Journal on Optimization 1(2):143-163. https://doi.org/10.1287/ijoo.2019.0015

http://pubsonline.informs.org/journal/ijoo/

INFORMS JOURNAL ON OPTIMIZATION
Vol. 1, No. 2, Spring 2019, pp. 143–163
ISSN 2575-1484 (print), ISSN 2575-1492 (online)

REPR: Rule-Enhanced Penalized Regression

Jonathan Eckstein,[a] Ai Kagawa,[b] Noam Goldberg[c]

[a] Department of Management Science and Information Systems and RUTCOR, Rutgers Business School Newark and New Brunswick, Rutgers University, Piscataway, New Jersey 08854; [b] Computational Science Initiative, Brookhaven National Laboratory, Upton, New York 11973; [c] Department of Management, Bar Ilan University, 5290002 Ramat Gan, Israel

Contact: jeckstei@business.rutgers.edu, http://orcid.org/0000-0002-6871-2691 (JE); aik@bnl.gov, http://orcid.org/0000-0001-8339-6175 (AK); noam.goldberg@biu.ac.il, http://orcid.org/0000-0002-1340-7569 (NG)

Received: January 15, 2018
Revised: December 6, 2018
Accepted: February 24, 2019
Published Online in Articles in Advance: May 14, 2019

https://doi.org/10.1287/ijoo.2019.0015

Abstract. This article describes a new rule-enhanced penalized regression procedure for the generalized regression problem of predicting scalar responses from observation vectors in the absence of a preferred functional form. It enhances standard L_1-penalized regression by adding dynamically generated rules, that is, new 0-1 covariates, corresponding to multidimensional "box" sets. In contrast to prior approaches to this class of problems, we draw heavily on standard (but non-polynomial-time) mathematical programming techniques, enhanced by parallel computing. We identify and incorporate new rules using a form of classical column generation and solve the resulting pricing subproblem, which is *NP*-hard, either exactly by a specialized parallel branch-and-bound method or by a greedy heuristic based on Kadane's algorithm. The resulting rule-enhanced regression method can be computation intensive when we solve the subproblems exactly, but our computational tests suggest that it outperforms prior methods at making accurate and stable predictions from relatively small data samples. Through selective use of our greedy heuristic, we can make our method's run time generally competitive with some established methods, without sacrificing prediction performance. We call our method's pricing subproblem *rectangular maximum agreement*.

History: This paper has been accepted for the *INFORMS Journal on Optimization* Special Issue on Machine Learning and Optimization.

Keywords: regression • machine learning • boosting • column generation • branch and bound • greedy heuristics

1. Introduction

Consider a general learning problem with m observation vectors $X_1, \ldots, X_m \in \mathbb{R}^n$ and matching response values $y_1, \ldots, y_m \in \mathbb{R}$. Each response y_i is presumed to be a possibly noisy evaluation of some unknown function $f : \mathbb{R}^n \to \mathbb{R}$ at X_i. The goal is to estimate f by some $\hat{f} : \mathbb{R}^n \to \mathbb{R}$ such that $|\hat{f}(X_i) - y_i|$ tends to be small and use $\hat{f}$ to predict the response value y corresponding to newly encountered observations $x \in \mathbb{R}^n$ through the prediction $\hat{y} = \hat{f}(x)$. We assume that prior knowledge of a concise functional form for f is unavailable. By way of notation, we let X denote the matrix whose rows are $X_1^\top, \ldots, X_m^\top$ and whose columns we denote by $x_1, \ldots, x_n$; we also let $y = (y_1, \ldots, y_m) \in \mathbb{R}^m$. We may then express a problem instance by a pair (X, y). We also let x_{ij} denote the $(i,j)^{\text{th}}$ element of this matrix, that is, the value of variable j in observation i.

Our approach to the kind of learning problem described here, like a number before it, involves constructing new covariates from axis-parallel box sets. Specifically, given two vectors $a, b \in \mathbb{R}^n$ with $a \le b$, we define the rule function $r_{(a,b)} : \mathbb{R}^n \to \{0, 1\}$ to be

$$r_{(a,b)}(x) = \begin{cases} 1 & \text{if } a_j \le x_j \le b_j \ \forall\, j = 1, \ldots, n, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Let K be some set of pairs $(a, b) \in \mathbb{R}^n \times \mathbb{R}^n$ with $a \le b$, constituting a catalog of all the possible rules of the form (1) that we wish to be available to our regression model. The set K is typically extremely large: restricting each a_j and b_j to values that appear as x_{ij} for some i, which is sufficient to describe all possible distinct behaviors of rules of the form (1) on the data set X, there are still $\prod_{j=1}^n \ell_j(\ell_j + 1)/2 \ge 3^n$ possible choices for (a, b), where $\ell_j = |\cup_{i=1}^m \{x_{ij}\}|$ is the number of distinct values for x_{ij} (we assume that $\ell_j \ge 2$ for each j because, otherwise, variable j may simply be dropped from the data set). The predictors $\hat{f}$ that our method constructs are of the form

$$\hat{f}(x) = \beta_0 + \sum_{j=1}^{n} \beta_j x_j + \sum_{k \in K} \gamma_k r_k(x) \tag{2}$$

for some $\beta_0, \beta_1, \ldots, \beta_m, (\gamma_k)_{k \in K} \in \mathbb{R}$. Typically, the vast majority of the coefficients $(\gamma_k)_{k \in K} \in \mathbb{R}$ are zero. Compared with learning approaches, such as kernel-based methods and neural networks, rule-based approaches have the advantage of generally being considered more accessible and easier to interpret by domain experts. In classification settings, as opposed to regression, rule-based methods also have a considerable history as in, for example, Weiss and Indurkhya (1993), Cohen and Singer (1999), and Dembczyński et al (2008a).

An earlier example of a regression method constructing models of the form (2) is RuleFit (Friedman and Popescu 2008), which uses a two-phase procedure: first, it determines a regression tree ensemble, and then it decomposes these trees into rules and determines the regression model coefficients (including for the rules).

The method of random forests (Breiman 2001) may be considered to be of the same form. This algorithm constructs $\hat{f}$ by training regression trees on multiple random subsamples of the data and then averaging the resulting predictors. The resulting model is equivalent to one of the form (2) with $\beta_1 = \beta_2 = \cdots = \beta_n = 0$.

The random forest and RuleFit methods both derive box rules (implicitly or explicitly, respectively) from an initial ensemble of regression trees. The approach of Dembczyński et al (2008b) generates rules more directly within a gradient-boosting (Friedman 2001) procedure for nonregularized regression. In this scheme, a greedy procedure generates the rules within a gradient-descent method that runs for a predetermined number of iterations. Aho et al. (2012) extended RuleFit to solve more general multitarget regression problems. For the special case of single-target regression, however, their experiments suggest that random forests and RuleFit outperform several other methods, including their own extended implementation and the algorithm of Dembczyński et al (2008b).

This paper describes an alternative procedure for constructing models of the form (2). Unlike the approaches described, our methodology draws more heavily on long-established exact non-polynomial-time optimization techniques arising from the field of operations research. Specifically, we iteratively orchestrate the generation of new rules using the classical mathematical programming technique of column generation in a manner reminiscent of the LPBoost ensemble classification method of Demiriz et al. (2002). We call our regression method rule-enhanced penalized regression (REPR); a condensed presentation of an earlier version of the algorithm appears in the short conference paper by Eckstein et al. (2017). Although REPR can be quite computationally intensive in some cases, its prediction performance appears promising in comparison with competing methods, especially when the amount of available training data are relatively small.

The pricing subproblem in our column-generation procedure is an *NP*-hard combinatorial problem we call *rectangular maximum agreement* (RMA), which generalizes the maximum mononial agreement (MMA) problem as formulated and solved in Eckstein and Goldberg (2012). We solve the RMA problem by a similar branch-and-bound method, implemented using parallel computing techniques. We have also devised a greedy heuristic approach to solving RMA: although not guaranteed to obtain the exact optimal solution, it is much faster and seems to work well empirically within our regression procedure.

Compared with our earlier conference proceedings publication (Eckstein et al. 2017), the version of REPR presented here contains numerous enhancements that significantly improve its performance and were enabled by major improvements to the RMA subproblem solver. In particular, Section 3.2 describes a new, recursive method for discretizing the input data to the RMA subproblem. We also developed more efficient data structures for the RMA solver as described at the end of Section 3.5. Further, Section 3.7 describes a new greedy heuristic that we use either as a substitute for our branch-and-bound method or to initialize its incumbent. As described in Section 3.8, we also developed a second, simpler greedy heuristic to be applied to each node of the branch-and-bound search. In our computational experiments with REPR, we now use cross-validation to select the regularization parameters in our regression model, specifically C and E in (3); this approach is described in Section 5.1 and became practical because of the RMA performance improvements resulting from our newly streamlined data structures. We also employ a new normalization procedure for the REPR input data as described in Section 5.2.

In addition, the presentation here is more detailed and comprehensive than Eckstein et al. (2017), and the computational results are more extensive. Among the additional topics discussed are the dual of our master optimization problem (see the end of Section 2), whose analysis demonstrates the relationship to forward selection methods in linear regression discussed in Section 3.1. Section 6.3 also describes REPR's relationship to other boosting regression methods and the role of our model's regularization parameter E in controlling overfitting. Experimentally, we test several more variants of the (improved) algorithm and one more competitor (gradient boosting). We also conducted computational tests on some larger data sets. The computer code used for our experiments is available on GitHub (Kagawa and Eckstein 2019).

2. A Penalized Regression Model with Rules

Finding a predictor function $\hat{f}$ of the form (2) may be considered as a matter of linear regression with coefficients in a space with the potentially very high dimension of $1 + n + |K|$. As is now customary in regression models in which the number of explanatory variables potentially outnumbers the number of observations, we employ a LASSO-class model in which all explanatory variables except the constant term have L_1 penalties. For $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}^{|K|}$, let $\hat{f}_{\beta_0,\beta,\gamma}(\cdot)$ denote the predictor function in (2). We then propose to estimate β_0, β, γ by (approximately) solving

$$\min_{\beta_0,\beta,\gamma} \left\{ \sum_{i=1}^{m} \left| \hat{f}_{\beta_0,\beta,\gamma}(X_i) - y_i \right|^p + C \|\beta\|_1 + E \|\gamma\|_1 \right\}, \tag{3}$$

where $p \in \{1, 2\}$ and $C, E > 0$ are scalar parameters. For $p = 2$, this model is essentially the classic LASSO as originally proposed by Tibshirani (1996). The decision variables are $\epsilon \in \mathbb{R}^m$, $\beta_0 \in \mathbb{R}$, $\beta \in \mathbb{R}^n$, and $\gamma \in \mathbb{R}^{|K|}$. To put (3) into a form more suitable for column generation, we split the regression coefficient vectors into positive and negative parts, so that $\beta = \beta^+ - \beta^-$ and $\gamma = \gamma^+ - \gamma^-$ with $\beta^+, \beta^- \in \mathbb{R}^n_+$ and $\gamma^+, \gamma^- \in \mathbb{R}^{|K|}_+$. Introducing one more vector of variables $\epsilon \in \mathbb{R}^m_+$, we obtain the equivalent problem formulation

$$\min_{\substack{\beta_0 \in \mathbb{R},\, \beta^+, \beta^- \geq 0 \\ \gamma^+, \gamma^- \geq 0,\, \epsilon \geq 0}} \quad \sum_{i=1}^{m} \epsilon_i^p + C \sum_{j=1}^{n} (\beta_j^+ + \beta_j^-) + E \sum_{k \in K} (\gamma_k^+ + \gamma_k^-) \tag{4a}$$

$$\text{s.t.} \quad \beta_0 + X_i^\top (\beta^+ - \beta^-) + \sum_{k \in K} r_k(X_i)(\gamma_k^+ - \gamma_k^-) - \epsilon_i \leq y_i, \qquad i = 1, \ldots, m \tag{4b}$$

$$-\beta_0 - X_i^\top (\beta^+ - \beta^-) - \sum_{k \in K} r_k(X_i)(\gamma_k^+ - \gamma_k^-) - \epsilon_i \leq -y_i, \qquad i = 1, \ldots, m. \tag{4c}$$

This model is constructed so that, in any optimal solution, $\epsilon_i = |\hat{f}_{\beta_0,\beta,\gamma}(X_i) - y_i|$ for $i = 1, \ldots, m$. For each $i = 1, \ldots, m$, constraints (4b) and (4c), respectively, tie ϵ_i to the overestimation or underestimation of y_i because they respectively reduce to

$$\hat{f}_{\beta_0,\beta^+,\beta^-,\gamma^+,\gamma^-}(X_i) - \epsilon_i \leq y_i \quad \Leftrightarrow \quad \hat{f}_{\beta_0,\beta^+,\beta^-,\gamma^+,\gamma^-}(X_i) - y_i \leq \epsilon_i, \tag{5a}$$

$$-\hat{f}_{\beta_0,\beta^+,\beta^-,\gamma^+,\gamma^-}(X_i) - \epsilon_i \leq -y_i \quad \Leftrightarrow \quad \hat{f}_{\beta_0,\beta^+,\beta^-,\gamma^+,\gamma^-}(X_i) - y_i \geq -\epsilon_i. \tag{5b}$$

If $p = 1$, problem (4) is a linear program, and if $p = 2$, it is a linearly constrained convex quadratic program. In either case, there are $2m$ constraints (other than nonnegativity), but the number of variables is $1 + m + 2n + 2|K|$. Because of this potentially unwieldy number of variables, we propose to solve (4) by using the classical technique of column generation, which dates back to Ford and Fulkerson (1958) and Gilmore and Gomory (1961); see, for example, section 7.3 of Griva et al. (2009) for a recent textbook treatment. As usual, our column-generation algorithm cycles between solving two optimization problems, the restricted master problem and the pricing problem. In our case, the restricted master problem is the same as (4) but with K replaced by some (presumably far smaller) $K' \subseteq K$; we initially choose $K' = \emptyset$. Solving the restricted master problem yields optimal Lagrange multipliers $\mu \in \mathbb{R}^m_+$ and $\nu \in \mathbb{R}^m_+$ for constraints (4b) and (4c), respectively. Using the structure of (4), it is easily deduced that the reduced costs of γ_k^+ and γ_k^- are, respectively,

$$\text{rc}[\gamma_k^+] = E - \sum_{i=1}^{m} r_k(X_i)\nu_i + \sum_{i=1}^{m} r_k(X_i)\mu_i, \tag{6a}$$

$$\text{rc}[\gamma_k^-] = E + \sum_{i=1}^{m} r_k(X_i)\nu_i - \sum_{i=1}^{m} r_k(X_i)\mu_i. \tag{6b}$$

Therefore, the column with the most negative reduced cost may be found by solving

$$z^* = \max_{k \in K} \left| \sum_{i=1}^{m} r_k(X_i)(\nu_i - \mu_i) \right|, \tag{7}$$

and the natural stopping condition for column generation is $z^* \leq E$. Problem (7) is an instance of the RMA problem, whose solution we discuss in the next section. If the optimal solution of (7) corresponds to a positive value of $\sum_{i=1}^{m} r_k(X_i)(\nu_i - \mu_i)$, then the corresponding entering variable is γ_k^+, whereas a negative value means that the entering variable is γ_k^-.

In terms of the respective Lagrange multipliers $\mu, \nu \in \mathbb{R}^m_+$ of the master problem constraints (4b) and (4c), the dual functional of (4) is

$$q(\mu,\nu) = \min_{\substack{\beta_0\in\mathbb{R},\, \beta^+,\beta^-\geq 0\\ \gamma^+,\gamma^-\geq 0,\, \epsilon\geq 0}} \sum_{i=1}^m \epsilon_i^p + C\sum_{j=1}^n(\beta_j^+ + \beta_j^-) + E\sum_{k\in K}(\gamma_k^+ + \gamma_k^-) \tag{8a}$$

$$+ \sum_{i=1}^m \mu_i\left[\beta_0 + X_i{}^T(\beta^+ - \beta^-) + \sum_{k\in K} r_j(X_i)(\gamma_j^+ - \gamma_j^-) - \epsilon_i - y_i\right] \tag{8b}$$

$$+ \sum_{i=1}^m \nu_i\left[-\beta_0 - X_i{}^T(\beta^+ - \beta^-) - \sum_{k\in K} r_j(X_i)(\gamma_j^+ - \gamma_j^-) - \epsilon_i + y_i\right]. \tag{8c}$$

The dual problem is to maximize $q(\mu\,\nu)$ over $\mu, \nu \in \mathbb{R}^+_m$. When $p = 1$, this dual problem takes the form

$$\max_{\mu,\nu\geq 0} \quad \sum_{i=1}^m y_i(\nu_i - \mu_i) \tag{9a}$$

$$\text{s.t.} \quad \sum_{i=1}^m(\mu_i - \nu_i) = 0 \tag{9b}$$

$$\mu_i + \nu_i \leq 1, \qquad i = 1,\ldots,m \tag{9c}$$

$$\left|\sum_{i=1}^m \nu_i x_{ij} - \sum_{i=1}^m \mu_i x_{ij}\right| \leq C, \qquad j = 1,\ldots,n \tag{9d}$$

$$\left|\sum_{i=1}^m \nu_i r_k(X_i) - \sum_{i=1}^m \mu_i r_k(X_i)\right| \leq E, \qquad k \in K. \tag{9e}$$

When $p = 2$, we instead obtain the dual problem

$$\max_{\mu,\nu\geq 0} \quad \sum_{i=1}^m y_i(\nu_i - \mu_i) - \frac{1}{4}\sum_{i=1}^m(\nu_i + \mu_i)^2 \tag{10a}$$

$$\text{s.t.} \quad \sum_{i=1}^m(\mu_i - \nu_i) = 0 \tag{10b}$$

$$\left|\sum_{i=1}^m \nu_i x_{ij} - \sum_{i=1}^m \mu_i x_{ij}\right| \leq C, \qquad j = 1,\ldots,n \tag{10c}$$

$$\left|\sum_{i=1}^m \nu_i r_k(X_i) - \sum_{i=1}^m \mu_i r_k(X_i)\right| \leq E, \qquad k \in K. \tag{10d}$$

Constraints (9b) and (10b) correspond to the primal variable β_0, dual constraints (9d) and (10c) correspond to the primal variables β_j^+ and β_j^-, and the dual constraints (9e) and (10d) correspond to the primal variables γ_k^+ and γ_i^-. The primal variable ϵ_i corresponds to constraint (9c) in the $p = 1$ case and to the quadratic objective function term in the $p = 2$ case.

Because every restricted master problem contains the variables $\beta_0 \in \mathbb{R}$ and $\epsilon_i, \beta_j^+, \beta_j^- \in \mathbb{R}_+$, the dual constraints other than (9e)/(10d) are satisfied at every column-generation iteration. However, the constraints (9e)/(10d) are not necessarily satisfied, because not all the possible variables γ_k^+, γ_k^- are present in the restricted master. Much as is customary in column generation, the pricing problem may be viewed as finding the most violated dual constraint of the form (9e)/(10d).

We next consider how to solve problems such as (7).

3. The RMA Problem

3.1. Problem Definition and Relation to Column Generation

Suppose we are given a data matrix X following the same notation as earlier, with each observation $i = 1,\ldots,m$ having a (possibly negative) "weight" $w_i \in \mathbb{R}$. For any set $S \subseteq \{1,\ldots,m\}$, let $w(S) = \sum_{i\in S} w_i$. Given two vectors $a, b \in \mathbb{R}^n$, we let $\mathrm{Cvr}_X(a,b)$ denote of the indices of the observations falling between a and b, that is,

$$\mathrm{Cvr}_X(a,b) = \{i \in \{1,\ldots,m\} \mid a \leq X_i \leq b\} = \left\{i \in \{1,\ldots,m\} \mid r_{(a,b)}(X_i) = 1\right\}.$$

We then define the RMA problem on the problem instance (X, w) to be

$$\begin{array}{ll}\max\limits_{a,b\in\mathbb{R}^n} & \left|w(\mathrm{Cvr}_X(a,b))\right| \\ \text{s.t.} & a,b\in\mathbb{R}^n.\end{array}$$

Essentially implicit in this formulation is the constraint that $a \le b$ because, if $a \not\le b$, then $\mathrm{Cvr}_X(a,b) = \emptyset$, and the objective value is zero. The previously mentioned MMA problem is the special case of RMA in which all the observations are binary, that is, $X \in \{0,1\}^{m\times n}$. Because the MMA problem is *NP*-hard (Eckstein and Goldberg 2012), so is RMA.

If we take K to be the set of all possible boxes on $\mathbb{R}^n$, the pricing problem (7) may be reduced to RMA by setting $w_i = \mu_i - \nu_i$ for each $i = 1,\dots,m$, where $\mu, \nu \in \mathbb{R}^m_+$ are the respective dual variables for constraints (4b) and (4c) of the restricted master problem.

The RMA subproblems generated by our column-generation procedure have some special structure: for any given $i = 1,\dots,m$, only one of the constraints (4b) or (4c) can be binding except in the situation that $\hat{f}_{\beta_0,\beta^+,\beta^-,\gamma^+,\gamma^-}(X_i) = y_i$ and $\epsilon_i = 0$, that is, the model exactly fits the response y_i. Otherwise, either the current model overestimates y_i, in which case constraint (4c) is slack and $\nu_i = 0$, or conversely, y_i is underestimated, and constraint (4b) is slack and $\mu_i = 0$.

Consider now the case $p = 1$. The dual constraints (9c) correspond to the primal variables ϵ_i, so for any observation y_i not being estimated exactly, complementary slackness requires that constraint (9c) be binding. Thus, we deduce that

$$\begin{array}{ll} w_i = \mu_i = 1 & \text{when observation } i \text{ is overestimated,} \\ w_i = -\nu_i = -1 & \text{when observation } i \text{ is underestimated.}\end{array}$$

Next, consider the case $p = 2$. If $(\epsilon, \beta^+, \beta^-, \gamma^+, \gamma^-, \beta_0)$ is an optimal primal solution and (μ, ν) is an optimal dual solution, then for any $i = 1,\dots,m$, we should have from the optimality conditions of the minimization within (8) that $(\partial/\partial\epsilon_i)[\epsilon_i^2 - \mu_i\epsilon_i - \nu_i\epsilon_i]$, which reduces to $2\epsilon_i = \mu_i + \nu_i$. We, therefore, conclude that

$$\begin{array}{ll} w_i = \mu_i = 2\epsilon_i & \text{when observation } i \text{ is overestimated,} \\ w_i = -\nu_i = -2\epsilon_i & \text{when observation } i \text{ is underestimated.}\end{array} \tag{11}$$

Combining all these observations and supposing that no observations are fitted exactly, we conclude that, in the $p = 1$ case, the pricing problem is to locate a box in which overestimated observations most outnumber underestimated ones or vice versa. When $p = 2$, the objective of the pricing problem is instead to find a box in which the sum of overestimation errors most greatly exceeds the sum of underestimation errors or vice versa.

In the $p = 2$ case, the relationships in (11) mean that the column-generation pricing problem has a natural interpretation in terms of forward selection and boosting methods in regression. Specifically, (11) implies that each weight w_i is directly proportional to the residual for observation i in the current regression model. If we treat w and some rule r_k as samples of random variables, their covariance is

$$\sigma_{w,r_k} = \sum_{i=1}^m \left(w_i - \frac{1}{m}\sum_{j=1}^m w_j\right)\left(r_k(X_i) - \frac{1}{m}\sum_{j=1}^m r_k(X_j)\right) = \sum_{i=1}^m w_i r_k(X_i),$$

where the second equality follows because (10b) is equivalent to $\sum_{i=1}^m w_i = 0$. Therefore, the pricing problem (7) is equivalent finding the rule having maximum absolute covariance with the current model residuals. This choice closely resembles that of traditional forward selection regression procedures, which build up models by iteratively adjoining the candidate explanatory variable with maximum absolute *correlation* with the current residuals; see, for example, Miller (1990, section 3.2) or Weisberg (2005, section 10.3). In such procedures, it is typically assumed that a preprocessing procedure has normalized each covariate to have zero mean and standard deviation one, in which case correlation and covariance become identical. Our procedure dynamically generates new explanatory variables, so it is not clear how to apply such a normalization. Specifically, modifying our procedure to use correlation instead of covariance would be equivalent to replacing (7) by

$$\max_{k\in K}\left\{\frac{\left|\sum_{i=1}^m r_k(X_i)w_i\right|}{\sqrt{\sum_{i=1}^m\left(r_k(X_i) - \frac{1}{m}\left(\sum_{j=1}^m r_k(X_j)\right)\right)^2}}\right\}, \tag{12}$$

the denominator being the standard deviation of the rule r_k (we omit the standard deviation of w from this formula because it is constant within each subproblem). As a practical matter, it is unclear how to incorporate the inverse-standard-deviation weighting in (12) into the pricing problem we shortly describe in Sections 3.3–3.5. Further, in comparison with using covariance as in (7), the objective in (12) favors rules r_k with unbalanced coverage of the data set, that is, those that return zero for most observations or one for most observations. Such a weighting may encourage overfitting, so we hypothesize that covariance-based selection may be preferable in our context, in which candidate variables are dynamically generated from a large set K and already have a natural 0-1 scaling.

3.2. Preprocessing and Restriction to $\mathbb{N}$

Any RMA problem instance may be converted to an equivalent instance in which all the observation data are integer. Essentially, for each coordinate $j = 1, \ldots, n$, one may simply record the distinct values of x_{ij} and replace each x_{ij} with its ordinal position among these values. Algorithm 1, with its parameter δ set to zero, performs exactly this "discretization" procedure, outputting an equivalent data matrix $\overline{X} \in \mathbb{N}^{m\times n}$ and a vector $\ell \in \mathbb{N}^n$ whose j^{th} element is $\ell_j = |\cup_{i=1}^m \{x_{ij}\}|$ as defined in the previous section. Algorithm 1's output values $\bar{x}_{ij}$ for attribute j vary between zero and $\ell_j - 1$.

Algorithm 1 (Discretization Algorithm)

1: **Input:** $X \in \mathbb{R}^{m\times n}$, $\delta \geq 0$
2: **Output:** $\overline{X} \in \mathbb{N}^{m\times n}$, $\ell \in \mathbb{N}^n$
3: ProcessData:
4: **for** $j = 1$ **to** n **do**
5: $\quad \ell_j \leftarrow 0$
6: $\quad$ Sort $x_{1j}, \ldots, x_{mj}$ and set $(k_1, \ldots, k_m)$ such that $x_{k_1 j} \leq x_{k_2 j} \leq \cdots \leq x_{k_m j}$
7: $\quad \bar{x}_{k_1,j} \leftarrow 0$
8: $\quad$ **for** $i = 1$ **to** $m - 1$ **do**
9: $\quad\quad$ **if** $x_{k_{i+1}j} - x_{k_i j} > \delta \cdot \mathrm{CI}_{95\%}(x_j)$ **then** $\ell_j \leftarrow \ell_j + 1$
10: $\quad\quad \bar{x}_{k_{i+1}j} \leftarrow \ell_j$
11: $\quad$ **end for**
12: $\quad \ell_j \leftarrow \ell_j + 1$
13: **end for**
14: **return** $(\overline{X}, \ell)$.

The number of distinct values ℓ_j for each explanatory variable j strongly influences the difficulty of an RMA problem instance. To obtain easier problem instances at the cost of implicitly searching a smaller set K of possible box rules in our column-generation application, we can set the parameters of Algorithm 1 to combine nearby values into common "bins." Essentially, if $\delta > 0$, the algorithm bins together consecutive values x_{ij} that are within relative tolerance δ, resulting in a smaller number of distinct values ℓ_j for each explanatory variable j.

The comparison between successive variable values on line 1 of Algorithm 1 uses δ scaled by the 95% central confidence interval of x_j, which we denote by $\mathrm{CI}_{95\%}(x_j)$ and define to be the difference range between the 97.5% and 2.5% quantiles of x_j (we could also easily substitute any similar quantile range). This comparison procedure provide some problem-adaptive scaling but limits the influence of extreme values of explanatory variables.

In cases in which an attribute has a large number of closely spaced values, Algorithm 1 can aggregate the input data into excessively large bins. So, after running the algorithm, we inspect the bins it has produced, and whenever the values in any bin span a range greater than $\rho\mathrm{CI}_{95\%}(x_j)$ for some parameter ρ, we break it up into subbins by recursively applying Algorithm 1 to just the data within the bin. We repeat this process recursively until no subbin spans more than a fraction ρ of the 95% central confidence interval of its containing bin; currently, we use the value $\rho = 0.1$. The resulting recursive discretization procedure works efficiently yet allows the RMA procedure to produce better quality boxes than the simpler single-level discretization procedure that we employed earlier in Eckstein et al. (2017); see Kagawa (2018) for details.

Within the context of our REPR method, we discretize the data X using using the process described previously for some (small) parameter value δ, set the RMA weight vector to $w = \mu - \nu$, solve the RMA problem, and then translate the resulting boxes back to the original, prediscretized coordinate system. We perform this translation by expanding box boundaries to lie halfway between the boundaries of the clusters of points created by the discretization procedure except when the lower boundary of the box has the lowest

possible value or the upper boundary has the largest possible value. In these cases, we expand the box boundaries to $-\infty$ or $+\infty$, respectively. More precisely, for each observation variable j and $v \in \{0,\dots,\ell_j - 1\}$, let $x_{j,v}^{\min}$ be the smallest value of x_{ij} assigned to the integer value v by Algorithm 1 and $x_{j,v}^{\max}$ be the largest. If $\hat{a}, \hat{b} \in \mathbb{N}^n$, $\hat{a} \le \hat{b}$ describes a discretized box arising from the solution of the preprocessed RMA problem, we choose the corresponding box boundaries $a, b \in \mathbb{R}^n$ in the original coordinate system to be given by, for $j = 1,\dots,n$,

$$a_j = \begin{cases} -\infty, & \text{if } \hat{a}_j = 0 \\ \frac{1}{2}\left(x_{j,\hat{a}_j-1}^{\max} + x_{j,\hat{a}_j}^{\min}\right), & \text{otherwise;} \end{cases} \qquad b_j = \begin{cases} +\infty, & \text{if } \hat{b}_j = \ell_j - 1 \\ \frac{1}{2}\left(x_{j,\hat{b}_j}^{\max} + x_{j,\hat{b}_j+1}^{\min}\right), & \text{otherwise.} \end{cases}$$

Overall, our procedure is equivalent to solving the pricing problem (7) over some set of boxes $K = \mathcal{K}_\delta(X)$. For $\delta = 0$, the resulting set of boxes $\mathcal{K}_0(X)$ is such that the corresponding set of rules $\{r_k | k \in \mathcal{K}_0(X)\}$ comprises every box-based rule distinguishable on the data set X. For small positive values of δ, the set of boxes $\mathcal{K}_\delta(X)$ excludes those corresponding to rules that "cut" between very closely spaced observations.

3.3. Branch-and-Bound Subproblems

In this and the following two sections, we describe the key elements of our branch-and-bound procedure for solving the RMA problem, assuming that the data X have already been preprocessed. For brevity, we omit some details which are instead covered in a forthcoming publication. In our method, each branch-and-bound subproblem P is characterized by four vectors $\underline{a}(P), \overline{a}(P), \underline{b}(P), \overline{b}(P) \in \mathbb{N}^n$, and represents a subset of the search space consisting of vector pairs (a, b) for which $\underline{a}(P) \le a \le \overline{a}(P)$ and $\underline{b}(P) \le b \le \overline{b}(P)$. Any valid subproblem conforms to $\underline{a}(P) \le \overline{a}(P)$, $\underline{b}(P) \le \overline{b}(P)$, $\underline{a}(P) \le \underline{b}(P)$, and $\overline{a}(P) \le \overline{b}(P)$. The root problem R of the branch-and-bound tree is $R = (\mathbf{0}, \ell - \mathbf{1}, \mathbf{0}, \ell - \mathbf{1})$, where $\ell \in \mathbb{N}^n$ is as output from Algorithm 1 and $\mathbf{0}$ and $\mathbf{1}$, respectively, denote the vectors $(0, 0, \dots, 0) \in \mathbb{N}^n$ and $(1, 1, \dots, 1) \in \mathbb{N}^n$.

3.4. Inseparability and the Bounding Function

Our bounding function uses an extension of the *inseparability* notion developed for the MMA problem in Eckstein and Goldberg (2012). Consider any subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$ and two observations i and i'. If $x_{ij} = x_{i'j}$ or $\overline{a}_j \le x_{ij}, x_{i'j} \le \underline{b}_j$ for each $j = 1,\dots,n$, then $x_i, x_{i'} \in \mathbb{N}^n$ are *inseparable* with respect to $\overline{a}, \underline{b} \in \mathbb{N}^n$ in the sense that any box with $a \le \overline{a}$ and $b \ge \underline{b}$ must either cover both of $x_i, x_{i'}$ or neither of them.

Inseparability with respect to $\overline{a}, \underline{b}$ is an equivalence relation among the indices $1,\dots,m$, and we denote its induced equivalence classes by $\mathcal{E}(\overline{a}, \underline{b})$. Our bound for subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$ is

$$g(\underline{a}, \overline{a}, \underline{b}, \overline{b}) = \max\left\{ \sum_{C \in \mathcal{E}(\overline{a}, \underline{b})} \left[w(C \cap \mathrm{Cvr}_X(\underline{a}, \overline{b}))\right]_+, \sum_{C \in \mathcal{E}(\overline{a}, \underline{b})} \left[w(C \cap \mathrm{Cvr}_X(\underline{a}, \overline{b}))\right]_- \right\}, \tag{13}$$

where $[d]_+ = \max\{d, 0\}$ and $[d]_- = \max\{-d, 0\}$ denote the positive and negative parts of a number, respectively. The reasoning behind this bound is that each possible box in the set specified by $(\underline{a}, \overline{a}, \underline{b}, \overline{b})$ must either cover or not cover the entirety of each $C \in \mathcal{E}(\overline{a}, \underline{b})$. The first argument to the "max" operation reflects the situation that every equivalence class C with a positive net weight is covered, and no classes with negative net weight are covered; this is the best possible situation if the box covers a higher weight of positive observations than of negative. The second "max" argument reflects the opposite situation, the best possible case in which the box covers a greater weight of negative observations than of positive ones.

3.5. Branching

The branching scheme of a branch-and-bound algorithm divides subproblems into smaller ones with the aim of improving their bounds. In our case, branching a subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$ involves choosing an explanatory variable $j \in \{1,\dots,n\}$ and a *cut point* $v \in \{\underline{a}_j, \dots, \overline{b}_j - 1\}$. Together, we call such a pair (j, v) a *branch point*.

There are three possible cases, the first of which is when $\underline{b}_j < \overline{a}_j$ and $v \in \{\underline{b}_j, \dots, \overline{a}_j - 1\}$. In this case, our scheme creates three children based on the disjunction that either $b_j \le v - 1$ (the box lies below v), $a_j \le v \le b_j$ (the box straddles v), or $a_j \ge v + 1$ (the box lies above v). The next case is that $v \in \{\underline{a}_j, \dots, \min\{\overline{a}_j, \underline{b}_j\} - 1\}$, in which case the box cannot lie below v and we split P into two children based on the disjunction that either $a_j \le v$ (the box straddles v) or $a_j \ge v + 1$ (the box is above v). The third case occurs when $v \in \{\max\{\overline{a}_j, \underline{b}_j\}, \dots, \overline{b}_j - 1\}$, in which case we split P into two children based on the disjunction that either $b_j \le v$ (the box is does not extend above v) or $b_j \ge v + 1$ (the box extends above v). If no v falling under one of these three cases exists for any dimension j,

then the subproblem represents a single possible box, that is, $\underline{a} = \overline{a}$ and $\underline{b} = \overline{b}$. Such a subproblem is a terminal node of the branch-and-bound tree, and in this case, we simply compute the RMA objective value for $a = \underline{a} = \overline{a}$ and $b = \underline{b} = \overline{b}$ as the subproblem bound.

When more than one possible branch point (j, v) exists, as is typically the case, our algorithm must select one. In the experiments described in this paper, we use a cut point caching strategy to accomplish this task, and this strategy draws on the standard technique of strong branching. Our RMA solver also includes several other branch-selection strategies that are relevant for data sets having very large numbers of distinct attribute values ℓ_j. However, these additional techniques are not relevant to the data sets tested here, and so we omit their description. The branch selection techniques manipulate internal representations of equivalence classes and use efficient techniques for predicting how the bound (13) will change in response to branching decisions; the details of these algorithms may be found in Kagawa (2018). Our implementations of these techniques now use an equivalence class representation in which each class C is stored as a sum of weights $\sum_{i \in C} w_i$ and a single "representative" observation $i \in C$. This simple code optimization made the implementation run more than twice as fast as the RMA solver we had developed in our earlier work (Eckstein et al. 2017).

3.5.1. Strong Branching. Strong branching is our simplest branch point selection strategy. In strong branching, we simply experiment with all applicable branch points (j, v) and select the one that minimizes the maximum bound of the resulting two or three children. This is a standard technique in branch-and-bound algorithms: it involves evaluating the bounds of all the potential children of the current search node and is the strategy used in Eckstein and Goldberg (2012) for the related MMA problem. To make the strong branching process as efficient as possible, we have developed specialized data structures for manipulating equivalence classes of the RMA problem, and we analyze the branching possibilities in a particular order dictated by this data structure.

3.5.2. Cutpoint Caching. Strong branching can be much more time-consuming for the RMA problem than for MMA because the number of potential branch points is often much larger. We therefore developed some alternative procedures. We tried randomly sampling branch points, but this approach greatly inflated the search tree. Eventually, we found that a technique we call "cut point caching" was effective. It is based on the tendency of strong branching to select the same branch points at many different points in the search tree. Essentially, every time we perform strong branching, we store the selected branch point in a cache. For each new subproblem, we check what fraction of its possible branch points are already in the cache. If this fraction is above a threshold parameter τ, we only check the cached branch points, comparing them using the same scoring method as for strong branching; otherwise, we perform strong branching and potentially add another branch point to the cache. This method significantly accelerated the branch-and-bound search: the branching selection is considerably faster than strong branching, on average, but the search tree did not inflate significantly. We obtained good results with $\tau = 0.001$. A detailed empirical analysis of the relative merits of cut point caching and strong branching may be found in Kagawa (2018).

3.6. Implementing the Branch-and-Bound Algorithm

We implemented our branch-and-bound algorithm using PEBBL (Eckstein et al. 2015), an open source C++ framework for branch-and-bound methods. PEBBL makes it relatively easy to move from a serial to a parallel implementation, using MPI-based parallelism (Gropp et al. 1994). In the initial phase of a parallel search, we take advantage of PEBBL's ability to support special "ramp-up" search procedures. At the beginning of the search, PEBBL can exploit parallelism within each subproblem rather than across the search tree, with all the processors synchronously searching an identical sequence of search nodes but sharing the work of evaluating each node. During this ramp-up phase, our implementation uses only strong branching and distributes the work of evaluating the various branch points, delegating a nearly equal number of branch points to each processor. This tactic efficiently parallelizes the branch selection procedure, the most time-consuming aspect of branch-and-bound algorithm. We trigger PEBBL's "crossover" to its standard asynchronous search mode when the number of nodes in the active subroblem pool becomes comparable to the number of processors or exceeds the number of feasible branch points of the current subproblem. After crossover, PEBBL asynchronously searches multiple nodes of the search tree in parallel with individual bounding and branching operations being handled by individual processors. Once we cross over, we enable any selected alternative to strong branching.

To support cut point caching during the asynchronous search phase, we developed a method to broadcast newly discovered cut points to other processors with relatively little redundancy. We use a hashing procedure to assign an "owning" processor to each branch point (j, v). Whenever a processor finds an apparently new

branch point, it sends it to the owning processor. If the owning processor has not encountered the branch point before, it broadcasts it to all processors. The broadcasts occur in an asyncronous, "background" manner so as not to disrupt the ongoing parallel search process.

3.7. A Greedy Heuristic for RMA

For several reasons, we supplemented our exact branch-and-bound method for the RMA problem by developing a heuristic solution procedure we call "greedy range search"; we developed this procedure after our preliminary work in Eckstein et al. (2017). We employ this heuristic in two ways: first, we use it to generate a good starting incumbent value at the root of the branch-and-bound search, pruning the search tree and, thus, saving memory as well as obviating possible nonessential work in parallel implementations. Second, we have also tried substituting the heuristic for exact solution of the pricing problem within our column-generation procedure, reducing run times at the possible cost of producing inferior rules.

We run our heuristic once to attempt to find a box with maximum (positive) covered weight and then run it again to try to find a box with minimum (negative) covered weight. In each case, the heuristic starts with the box $B = (a, b) = (\mathbf{0}, \ell - \mathbf{1})$, covering the entire (preprocessed) data set. Each iteration of the heuristic then greedily selects an attribute j to constrain, modifying the box lower bound a_j and upper bound b_j. To select j, a_j, and b_j at each iteration of our greedy heuristic, we make use of Kadane's algorithm, which is a well-known linear-time procedure for identifying a contiguous subarray of a one-dimensional array having the largest or smallest possible sum; see, for example, Bentley (1984).

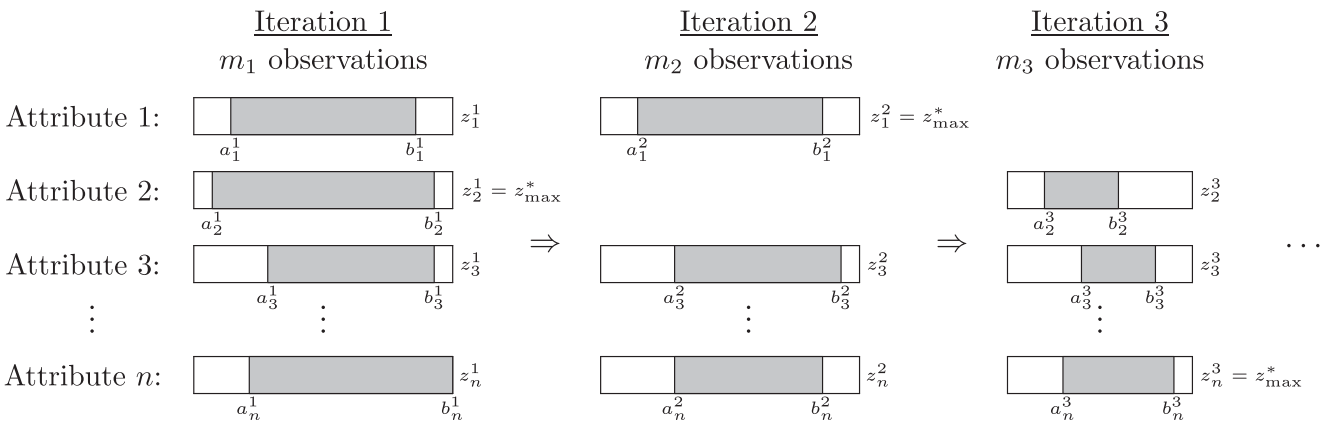
Consider the case in which we are attempting to maximize the covered weight of the box. We evaluate each possible attribute $j = 1, \ldots, n$ and use Kadane's algorithm on an array of length ℓ_j to identify the choice of a_j and b_j maximizing $w(B)$. We select the attribute j^* that maximizes the resulting covered weight and modify a_{j^*} and b_{j^*} accordingly.

We then iteratively repeat this procedure, considering at each iteration every attribute j except the j^* just selected in the prior iteration (changing the bounds on this attribute cannot improve the objective function unless we first change some other attribute bounds). At each iteration, we consider only narrowing the bounds on the attributes (that is, increasing a_j and/or decreasing b_j). We continue in this manner until the covered weight of the box cannot be increased by narrowing the bounds of any individual attribute j.

The case of attempting to find the most negative covered box weight is similar, substituting minimization for maximization at each step. Finally, we compare the results of the maximization and minimization heuristics, selecting whichever has the larger absolute value.

Figure 1 shows an example of the greedy range search method. For each attribute j in each iteration, the shaded portion of each rectangle shows the attribute values having the greatest weight (assuming maximization). In the first iteration, we initially have all $m_1 = m$ nonzero-weight observations. We apply Kadane's algorithm to each attribute $j = 1, \ldots, n$, obtaining a set of candidate objective values $z_1^1, \ldots, z_n^1$. In the example, we suppose that the largest of these values (in the case of maximization) is $z^*_{\max} = z_2^1$; we then modify (a_2, b_2) with the result that the box covers some smaller number of observations $m_2 < m_1 = m$ and repeat the procedure. In the second iteration, we consider further constraining each box dimension except for $j = 2$, which was just modified. Supposing for the case of the example that the next attribute we constrain is $j = 1$ (resulting in

Figure 1. An Example of the Operation of Our Greedy Heuristic



$m_3 < m_2$ covered observations), the third iteration considers every attribute except $j = 1$; it is once again possible to consider constraining attribute 2.

Algorithm 2 (Maximum Greedy Range Search)

1: **Input:** $n \in \mathbb{R}$, $\ell \in \mathbb{N}^n$ (the number of distinct values in each attribute),
$X \in \mathbb{R}^{m \times n}$ (data), $w \in \mathbb{R}^m$ (observation weights)
2: **Output:** $z^*_{\max} \in \mathbb{R}, a, b \in \mathbb{N}^n$
3: **MaxGreedyRange**(n, ℓ, w, X):
4: $z^*_{\max} \leftarrow \sum_{i=1}^m w_i; j^* \leftarrow -1; (a, b) \leftarrow (\mathbf{0}, \ell - \mathbf{1}); M = \{1, \ldots, m\}$; allocate $W \in \mathbb{R}^{\max\{\ell_1, \ldots, \ell_n\}}$
5: **while** true **do**
6: **for** $j \in \{1, \ldots, n\} \setminus j^*$ **do**
7: $z_{\max} \leftarrow z^*_{\max}$
8: **for** $v = a_j, \ldots, b_j$ **do** $W[v] = 0$
9: **for** $i \in M$ **do** $W[x_{ij}] \leftarrow W[x_{ij}] + w_i$
10: $(z, l, u) \leftarrow$ **MaxKadane**(W, a_j, b_j)
11: **if** $z > z_{\max}$ **then** $(j^*, z_{\max}, l^*, u^*) \leftarrow (j, z, l, u)$
12: **end for**
13: **if** $z^*_{\max} \geq z_{\max}$ **then break**
14: $a_{j^*} \leftarrow l^*; b_{j^*} \leftarrow u^*; z^*_{\max} \leftarrow z_{\max}$
15: $M \leftarrow$ **dropObsNotCovered**$(j^*, l^*, u^*, M, x_{j^*})$
16: **end while**
17: **return** $(z^*_{\max}, a, b)$.

Algorithm 2 summarizes, with some simplifications, the maximization instance of our greedy range search procedure, assuming that the data have already been discretized as described in Section 3.2. The initial objective value is the sum of weights of all training observations, $\sum_{i=1}^m w_i$, which must be zero in our REPR application because the dual constraint (9b)/(10b) must be satisfied. The function call to **MaxKadane** on line 10 applies Kadane's algorithm to the portion working array W between elements a_j and b_j, returning the maximum contiguous subarray sum in z and extent (l, u) of the corresponding subarray, where $a_j \leq l \leq u \leq b_j$. The subroutine **dropObsNotCovered** invoked on line 15 removes from the list of indices M any observations that are excluded if the bounds of the box B are narrowed to (l^*, u^*) in attribute j^*. Our actual implementation is somewhat more complicated because we effectively compress the working array W to represent only values of each attribute j that remain represented in the set of observations M.

Examining the complexity of Algorithm 2, the highest complexity steps in the body of the main **for** loop on lines 6–12 are lines 8–10; lines 8 and 10 have complexity $O(b_j - a_j) \subseteq O(\ell_j) \subseteq O(m)$, and line 9 has complexity $O(M) \subseteq O(m)$. It follows that each iteration of the main **for** loop requires $O(m)$ time. This **for** loop iterates over $\{1, \ldots, n\} \setminus j^*$, so each execution of the loop requires $O(mn)$ time. Finally, each iteration of the **while** loop except the last must remove at least one observation from the covered set M because otherwise there could be no objective improvement and the **break** condition would trigger to terminate the loop. The call to **dropObsNotCovered** on line 15 can be implemented in $O(M) \subseteq O(m)$ time, so it follows that the **while** loop executes at most m times and that the overall run time is $O(m^2 n)$ although this is a very loose bound. This polynomial complexity contrasts with the potentially exponential run time of our branch-and-bound procedure.

3.8. A Streamlined Heuristic at Each Subproblem

As our branch-and-bound method processes each subproblem, it attempts to update the incumbent. Our basic approach at each subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$ is to try the box $(\underline{a}, \overline{b})$, updating the incumbent if the objective value $|w(\mathrm{Cvr}_X(\underline{a}, \overline{b}))|$ is larger than the current incumbent value. Further, we also apply what is essentially one iteration of our greedy heuristic to $(\underline{a}, \overline{b})$: we use both the minimization and maximization versions of Kadane's algorithm on each attribute to find which way of narrowing the bounds on a single attribute improves the objective the most. The result replaces the current incumbent if it has larger objective value. This procedure allows the algorithm to find the eventual optimal solution more quickly with similar benefits to running the more elaborate heuristic of Section 3.7 at the root.

Algorithm 3 (REPR)

Input: data $X \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m$, penalty parameters $C, E \geq 0$, column-generation tolerance $\theta \geq 0$, integer $t \geq 1$, aggregation tolerance $\delta \geq 0$, and iteration limit S
Output: $\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^n, K' \subset \mathcal{K}_\delta(X), \gamma \in \mathbb{R}^{|K'|}$
REPR:
Apply the preprocessing procedure of Section 3.2 to the data set X
$K' \leftarrow \emptyset$
for $s = 1, \ldots, S$ **do**
  Solve the restricted master problem to obtain optimal primal variables $(\beta_0, \beta^+, \beta^-, \gamma^+, \gamma^-)$ and dual variables (ν, μ)
  Use the heuristic of Algorithm 2 to identify a possibly suboptimal solution k to the pricing problem (7), with objective value z
  if $z \leq E$ **then**
    Use the (parallel) branch-and-bound RMA algorithm of Sections 3.3–3.7 to identify an optimal solution k to (7), with objective value z
    if $z \leq E + \theta$ **break**
  end if
  $K' \leftarrow K' \cup \{k\}$
end for
return $(\beta_0, \beta := \beta^+ - \beta^-, K', \gamma := \gamma^+ - \gamma^-)$.

4. Full Algorithm and Implementation

The pseudocode in Algorithm 3 describes our full REPR column-generation procedure for solving (4), preprocessing the subproblem input data using the procedure described in Section 3.2. At each iteration, we first attempt to use the greedy heuristic of Algorithm 2 to identify a new box pattern to adjoin to the restricted master. If the greedy heuristic fails to find an entering column, then we use the branch-and-bound method to solve the pricing problem exactly. Several points bear mentioning: first, any observations for which $\nu_i = \mu_i$ have zero weight in the pricing problem and are temporarily removed from the subproblem instance before executing the greedy heuristic or the branch-and-bound method. Second, the nonnegative scalar parameter θ allows us to incorporate a tolerance into the column-generation stopping criterion, so that we terminate when all reduced costs exceed $-\theta$ instead of when all reduced costs are nonnegative. This kind of tolerance is customary in column-generation methods. The tolerance δ, on the other hand, controls the space of columns searched over. Furthermore, using some features already available in PEBBL, our implementation of the RMA branch-and-bound algorithm can identify any desired number $t \geq 1$ of the best possible RMA solutions as opposed to just one value of k attaining the maximum in (7). This t is also a parameter to our procedure, so at each iteration of Algorithm 3, we may adjoin up to t new rules to K'; however, we omit this complication in the description of Algorithm 3 in the interest of simplicity. Finally, the algorithm has a parameter S specifying a limit on the number of column-generation iterations, meaning that the output model contains at most St rules.

In addition to implementing the RMA branch-and-bound algorithm using C++ and PEBBL as mentioned, we implemented the remainder of Algorithm 3 in C++, using the Gurobi commercial optimizer (Gurobi Optimization 2016) to solve the restricted master problems, employing its LP or QP simplex method and "warm starting" from the optimal solution basis of the previous restricted master problem. If one is careful to warm-start the solution of the restricted master problem in this way, solving the RMA pricing problem tends to be, by far, the most time-consuming part of Algorithm 3, so we currently use true parallel computing only in that portion of the algorithm. The remainder of the algorithm, including solving the restricted master problems, was executed in serial and redundantly on all processors.

5. Computational Testing

5.1. Setting Parameters by Cross-Validation

To evaluate the performance REPR, we need to set its parameters, in particular the L_1-penalty coefficients C and E. Small values of C and E (along with a high iteration limit S) can lead to overfitting—improving apparent performance on training data while worsening performance on testing data. As is standard in the machine learning community, we use a cross-validation procedure to determine suitable parameters for each data set; this is in contrast to our earlier work in Eckstein et al. (2017), in which we used a single setting of C

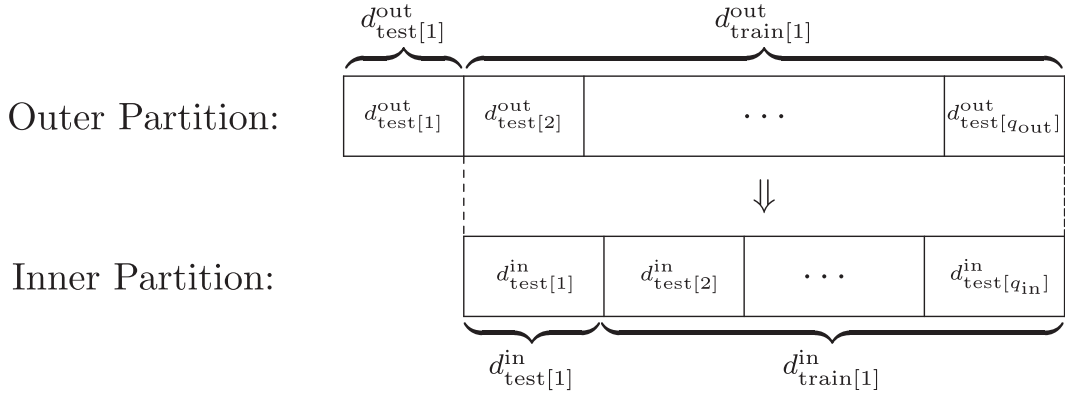
and E across all data sets. In our computational tests, this parameter-selection procedure is nested within an outer cross-validation procedure that evaluates the accuracy of our regression method.

Algorithm 4 (Bilevel (Nested) Cross-Validation Procedure to Evaluate REPR and Set Its Parameters)

Input: $q_{\text{out}}, q_{\text{in}}, S_{\text{out}}, S_{\text{in}} \in \mathbb{N}$,
$d = (X, y) \in (\mathbb{R}^{m\times n}, \mathbb{R}^m)$, $\Lambda \subset \mathbb{R}^T$ (parameter combinations to test)
Output: $\bar{\varepsilon}_{\text{our}}, \bar{\varepsilon}_{\text{other}}$
BilevelCrossValidation:
$(d^{\text{out}}_{\text{test}[1]}, \ldots, d^{\text{out}}_{\text{test}[p]}) \leftarrow$ **RandomPartition**(d, p)
for $i = 1, \ldots, q_{\text{out}}$ **do**
  $\varepsilon^* = \infty$
  $d^{\text{out}}_{\text{train}[i]} \setminus d^{\text{out}}_{\text{test}[i]}$
  $\left(d^{\text{in}}_{\text{test}[1]}, \ldots, d^{\text{in}}_{\text{test}[q]}\right) \leftarrow$ **RandomPartition**$\left(d^{\text{out}}_{\text{train}[i]}, q\right)$
  for each $\lambda \in \Lambda$ **do**
    for$j = 1, \ldots, q_{\text{in}}$ **do**
      $d^{\text{in}}_{\text{train}[j]} \leftarrow d^{\text{out}}_{\text{train}[i]} \setminus d^{\text{in}}_{\text{test}[j]}$
      $\mathcal{M}_{\text{our}} \leftarrow$ **TrainOurModel**$\left(d^{\text{in}}_{\text{train}[j]}, \lambda, S_{\text{in}}\right)$
      $\varepsilon_{\text{our}}[j] \leftarrow$ **EvaluteModel**$\left(\mathcal{M}_{\text{our}}, d^{\text{in}}_{\text{test}[j]}\right)$
  end for
  $\bar{\varepsilon}_{\text{our}} = (\epsilon_{\text{our}})$
  if $\bar{\varepsilon}_{\text{our}} < \varepsilon^*$ **then** $\lambda^* \leftarrow \lambda$; $\varepsilon^* \leftarrow \bar{\varepsilon}_{\text{our}}$
end for
$\mathcal{M}_{\text{our}} \leftarrow$ **TrainOurModel**$\left(d^{\text{out}}_{\text{train}[i]}, \lambda^*, S_{\text{out}}\right)$
$\varepsilon_{\text{our}}[i] \leftarrow$ **EvaluteModel**$\left(\mathcal{M}_{\text{our}}, d^{\text{out}}_{\text{test}[i]}\right)$
$\mathcal{M}_{\text{other}} \leftarrow$ **TrainOtherModel**$\left(d^{\text{out}}_{\text{train}[i]}\right)$
$\varepsilon_{\text{other}}[i] \leftarrow$ **EvaluateModel**$\left(\mathcal{M}_{\text{other}}, d^{\text{out}}_{\text{test}[i]}\right)$
end for
$\bar{\varepsilon}_{\text{our}} = \text{avg}(\varepsilon_{\text{our}})$
$\bar{\varepsilon}_{\text{other}} = \text{avg}(\varepsilon_{\text{other}})$
return $(\bar{\varepsilon}_{\text{our}}, \bar{\varepsilon}_{\text{other}})$.

Algorithm 4 outlines our full bilevel (nested) cross-validation procedure with q_{out} folds of outer cross-validation and q_{in} folds of inner cross-validation. The set Λ holds the various parameter combinations to be tested. The subroutine **RandomPartition**(d, s) randomly partitions an input data set $d = (X, y)$ into s smaller data sets of nearly equal size. The first call to **RandomPartition** produces a q_{out}-way partition that we use to evaluate the overall performance of the algorithm and its competitors. We evaluate each algorithm by its performance averaged over the q_{out} folds of this partition, using one element of the partition as the testing set and combining the remaining elements into the training set. Within each of these q_{out} folds, we use a q_{in}-way partition of the resulting training set to decide on the best choice of algorithm parameters λ from the set of

Figure 2. Training and Testing Data Sets in the Bilevel Cross-Validation Procedure in the Case $i = 1, j = 1$



candidates Λ. We use two different limits S_{in} and S_{out} on the number of column-generation iterations when training our model because the number of iterations needed to identify good parameter settings is typically less than that required to fit the data best. We use the possibly smaller number S_{in} when searching for the parameters and the possibly larger number S_{out} to evaluate modeling performance.

Figure 2 depicts the situation when the outer loop index i is one (we are testing the first of the q_{out} outer folds) and the inner loop index j is also one (we are testing the first of the q_{in} inner folds). The outer testing set is $d^{\text{out}}_{\text{test}[1]}$, the first element returned from the initial call to **RandomPartition**, and the outer training set is the remainder of the input data set. To choose the best parameters $\lambda \in \Lambda$, we partition this training set into q parts. In the first iteration of the inner loop (over j) the inner testing set is the first element of this inner partition with the remaining elements making up the inner testing set. We select the parameter setting λ^* that has the best performance averaged over the folds of the inner partition.

The end of Algorithm 2 invokes the subroutines **TrainOtherModel** and **EvaluateModel** to evaluate the performance of a competing model. In actuality, we evaluate multiple competing models, including RuleFit, random forests, gradient boosting, and simple linear regression. Some of these methods may perform their own internal parameter optimizations.

5.2. Small Test Data Sets and Data Standardization

We tested the REPR procedure on some small data sets from the University of California, Irvine (UCI) data repository (Lichman 2013) as described in Table 1. The column of this table shows $K_0(X)$, the total number of box descriptor pairs (a, b) that have distinguishable coverage on each data set.

Before applying REPR, we standardized the problem input data so that the response vector y and all columns x_j of X have mean zero and standard deviation one, as follows: given any nonstandardized problem input instance $(\hat{X}, \hat{y})$, we set

$$y_i = \frac{\hat{y}_i - \mu(\hat{y})}{\sigma(\hat{y})} \quad \forall\, i = 1, \ldots, m \quad x_{ij} = \frac{\hat{x}_{ij} - \mu(\hat{x}_j)}{\sigma(\hat{x}_j)} \quad \forall\, i = 1, \ldots, m \quad \forall\, j = 1, \ldots, n,$$

where $\mu(\cdot)$ and $\sigma(\cdot)$, respectively, denote computing the mean and (sample) standard deviation of a vector, and $\hat{x}_j$ denotes the jth column of $\hat{X}$. This standardization procedure makes the effect of the parameter C uniform across variables and also fixes the relative scaling of the parameters C and E, making it possible to try fewer values of these parameters during cross-validation; it was absent from our earlier work in Eckstein et al. (2017). For input data containing categorical attributes, we also convert each c-way categorical attribute into $c - 1$ binary attributes.

5.3. Numerical Results for Small Data Sets

We used our bilevel cross-validation procedure twice with $q_{\text{out}} = 5$ outer folds and $q_{\text{in}} = 3$ inner folds. This procedure gave us a performance sample of size 10 for each data set but each with an 80–20 data split between training and testing data. We selected the REPR's parameters as follows:

- We used our inner cross-validation procedure to select the parameters C and E. We experimented only with cases in which $C = E$, selecting the possible values from $\{0, 0.5, 1, 1.5, 2\}$.
- We did complete tests for both $p = 1$ (absolute deviation loss) and $p = 2$ (quadratic loss); our earlier work in Eckstein et al. (2017) only tested $p = 2$.
- After some initial experimentation, we used $\delta = 0.005$.
- The column-generation iteration limits were $S_{\text{in}} = 20$ (for parameter tuning) and $S_{\text{out}} = 150$ (for model evaluation).
- We set $t = 1$, so we only added one box rule per iteration of column generation.

Table 1. Summary of the Small Experimental Data Sets

Data set	m	n	$K_0(X)$
SERVO	167	10	9.8×10^5
CONCRETE	103	9	2.7×10^{29}
MACHINE	209	6	2.5×10^{15}
YACHT	308	6	2.6×10^{10}
MPG	392	7	3.3×10^{19}
COOL	768	8	1.1×10^{10}
HEAT	768	8	1.1×10^{10}
AIRFOIL	1503	5	1.0×10^{11}

In addition to testing the REPR procedure shown in Algorithm 3, we also tested two variants we call EREPR and GREPR. The EREPR variant solves all the pricing subproblems exactly without first attempting to use the greedy heuristic (however, heuristics are still used within the exact RMA algorithm). Conversely, the GREPR variant only uses the greedy heuristic and does not attempt to solve subproblems exactly by branch and bound. Our earlier work in Eckstein et al. (2017) used only what we now call EGREPR but with some other differences.

GREPR is not able to use the termination test in line 11 of Algorithm 3 because the greedy method solution may not necessarily correspond to the column with the most negative reduced cost. Instead, we simply terminate GREPR whenever the greedy method returns the same box rule on two successive calls (which would otherwise cause the method to enter an infinite loop) or when we encounter the iteration limit S.

We compared the performance of REPR, EREPR, and GREPR to RuleFit, random forests, gradient boosting, and classical linear regression. To implement RuleFit, random forests, gradient boosting, and classical linear regression, we used their publicly available R packages with default parameter settings; the variant of gradient boosting implemented is described in Ridgeway (2007). We allowed gradient boosting to run for up to 10,000 iterations and random forests to generate up to 10,000 trees.

Table 2 summarizes the testing-set MSE performance of the various methods, averaged over the 10 runs in our two invocations of the cross-validation procedure and scaled by $(1/m)\sum_{i=1}^{m} y_i^2$. The smallest average MSE for each data set is shown in bold. Figure 3 displays the same information as a bar chart, normalized so that the $p = 2$ REPR value has height one for each data set. For all the data sets except MPG and YACHT, the smallest average MSE is attained by one of the REPR-family methods, and the average MSE of the $p = 2$ version of REPR is lower than all the non-REPR methods. For YACHT, RuleFit produces the lowest average MSE, but the $p = 2$ REPR variants produce similar values, with EREPR coming the closest. For MPG, the $p = 2$ REPR variants, RuleFit, and gradient boosting all produce very similar average MSE values, but gradient boosting is the lowest. For each of the non-REPR methods, there are multiple data sets for which the average MSE is substantially worse than $p = 2$ REPR.

Table 3 summarizes the standard deviation of the sample of 10 calculated test MSE values, which we take as a measure of prediction stability. The REPR-class methods have the lowest standard deviation on six of the eight data sets, and the $p = 2$ REPR values are lower than those for the non-REPR methods for the same six data sets. We have also observed that REPR's prediction stability can be further improved by performing more inner cross-validation folds when searching for the best parameters C and E.

Table 4 summarizes the run time performance of all the algorithms in average seconds per outer fold. Each run was performed on a single 2.1GHz Xeon E5-2695 v4 node of one of our university computing resource clusters. For all the pricing subproblems for EREPR and for exact-case pricing in REPR, we used the 36 processor cores available on a single compute node. All other operations were serial, including the entirety of GREPR. Although EREPR is much slower than the other methods, the run times of REPR and GREPR are more competitive. GREPR is consistently faster than RuleFit, and REPR is faster than RuleFit for five out of the eight data sets. The MPG data set seems to produce particularly difficult RMA subproblems, as evidenced by the relatively long run times for REPR and EREPR.

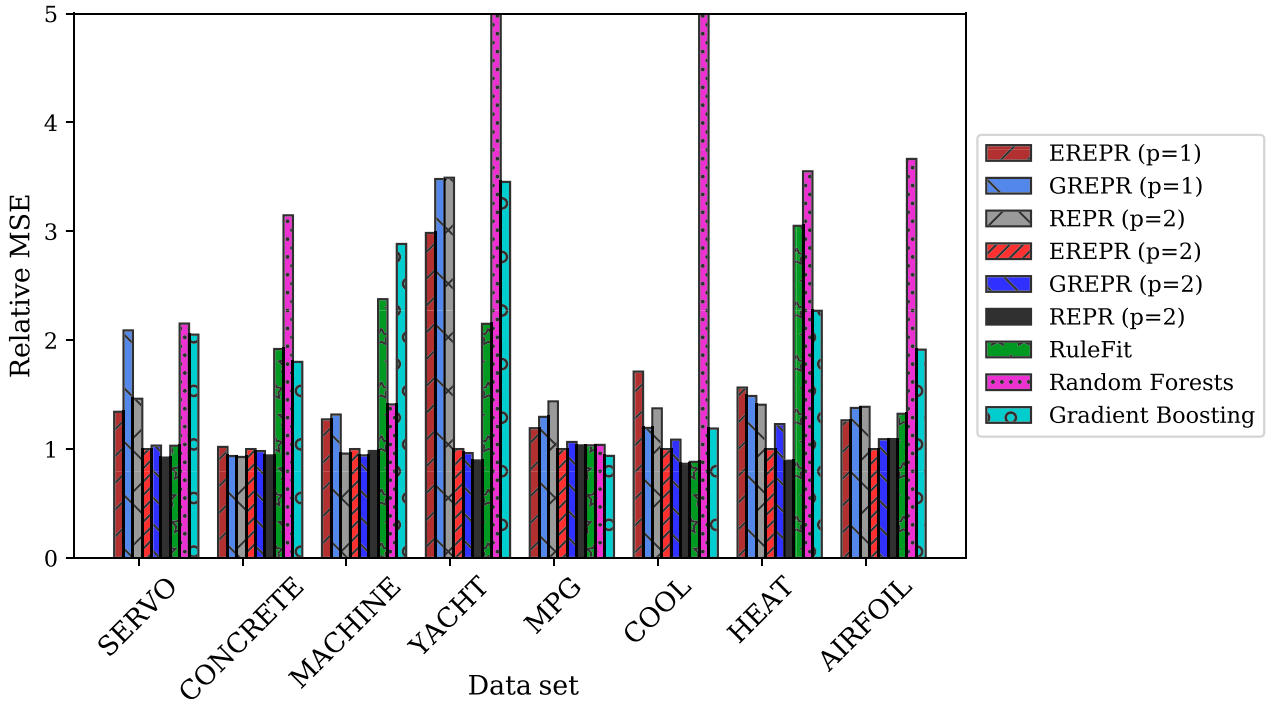
Table 5 compares the single-outer-fold run time of REPR with our earlier EREPR implementation from Eckstein et al. (2017). Our current implementation is approximately an order of magnitude faster.

Table 2. Average Test Mean Squared Error (MSE) over the Small Data Sets, Scaled by $(1/m)\sum_{i=1}^{m} y_i^2$

	Data set							
Method	SERVO	CONCRETE	MACHINE	YACHT	MPG	COOL	HEAT	AIRFOIL
EREPR ($p = 1$)	0.08080	0.00467	0.09562	0.00679	0.01332	0.00066	0.00262	0.00031
GREPR ($p = 1$)	0.12579	0.00428	0.09897	0.00582	0.01448	0.00046	0.00249	0.00033
REPR ($p = 2$)	0.08801	**0.00424**	**0.07202**	0.00817	0.01607	0.00053	0.00235	0.00034
EREPR ($p = 2$)	0.06019	0.00458	0.07513	**0.00283**	0.01118	0.00038	0.00167	**0.00024**
GREPR ($p = 1$)	0.06211	0.00449	0.07076	0.00361	0.01191	0.00042	0.00205	0.00026
REPR ($p = 2$)	**0.05549**	0.00431	0.07381	0.00336	0.01156	**0.00033**	**0.00149**	0.00026
RuleFit	0.06203	0.00878	0.17856	0.00265	0.01156	0.00034	0.00510	0.00032
Random forests	0.12956	0.01440	0.10593	0.04874	0.01161	0.00223	0.00594	0.00089
Gradient boosting	0.12345	0.00824	0.21669	0.02395	**0.01048**	0.00046	0.00379	0.00046
Linear regression	0.27954	0.00460	0.11723	0.23322	0.01860	0.01342	0.01818	0.00143

Note. The smallest value in each column is bolded.

Figure 3. (Color online) Visual Comparison of Average Testing-Set Mean Squared Error (MSE) Values with the $p = 2$ EREPR Value Normalized to One



The charts in Figure 4 give more detailed information for the small data sets. For the C and E parameter values selected by the inner cross-validation step, these charts show how the $p = 2$ REPR, EREPR, and GREPR testing-set prediction MSEs (in the original, nonstandardized coordinate system) evolve with each iteration with each data point averaged over the two fivefold cross-validations; the horizontal lines indicate the average MSE level for the competing procedures. MSE generally declines as REPR adds rules, with the exception of CONCRETE; some dimishing returns are also evident for MACHINE. The first iteration of the REPR models is equivalent to a simple LASSO model, which has similar performance to linear regression. We observed that GREPR tends to stop earlier than EREPR and REPR. Interestingly, the charts show only limited evidence of overfitting by the REPR-family algorithms even when large numbers of rules are incorporated into the model.

5.4. Preliminary Numerical Results with Larger Data Sets

We also made some preliminary experiments with the $p = 2$ GREPR method and the non-REPR competitors on four UCI data sets having similar small numbers of attributes n but considerably larger numbers of observations m, as summarized in Table 6; these data sets were not tested in our earlier work in Eckstein et al. (2017). We used

Table 3. Standard Deviation of Testing-Set MSE over the Small Data Sets, Scaled by $(1/m)\sum_{i=1}^{m}|y_i|$

	Data set							
Method	SERVO	CONCRETE	MACHINE	YACHT	MPG	COOL	HEAT	AIRFOIL
EREPR ($p = 1$)	1.07E-03	5.04E-04	8.79E-02	**8.53E-05**	1.65E-04	1.41E-05	1.28E-05	2.26E-06
GREPR ($p = 1$)	1.12E-03	4.29E-04	1.39E-01	2.60E-04	1.47E-04	1.93E-05	6.96E-06	2.97E-06
REPR ($p = 1$)	1.02E-03	**3.97E-04**	**4.54E-02**	2.21E-04	2.46E-04	2.37E-05	1.10E-05	3.37E-06
EREPR ($p = 2$)	**5.18E-04**	4.30E-04	6.54E-02	1.42E-04	9.99E-05	1.52E-05	1.00E-05	2.52E-06
GREPR ($p = 2$)	8.03E-04	4.30E-04	6.72E-02	9.38E-05	1.03E-04	1.92E-05	8.75E-06	**1.69E-06**
REPR ($p = 2$)	6.62E-04	3.50E-04	7.48E-02	8.65E-05	1.03E-04	**1.14E-05**	3.75E-06	1.77E-06
RuleFit	7.58E-04	1.15E-03	2.19E-01	1.61E-04	9.70E-05	1.66E-05	**2.71E-06**	3.72E-06
Random forests	1.20E-03	8.58E-04	1.51E-01	2.66E-03	8.91E-05	2.52E-05	1.68E-05	2.91E-06
Gradient boosting	1.78E-03	2.66E-03	4.27E-01	6.83E-04	**7.34E-05**	1.06E-04	2.08E-05	4.25E-05
Linear regression	1.25E-03	5.13E-04	9.94E-02	5.97E-03	1.08E-04	6.28E-05	3.70E-05	7.50E-06

Note. The smallest value in each column is bolded.

Table 4. Average Single-Outer-Fold Run Times in Seconds for the Small Data Sets

	Data set							
Method	SERVO	CONCRETE	MACHINE	YACHT	MPG	COOL	HEAT	AIRFOIL
EREPR ($p = 1$)	2.2	228.2	108.3	8.3	9878.4	23.5	22.0	519.2
GREPR ($p = 1$)	0.7	0.6	1.0	1.9	5.8	5.3	2.8	12.9
REPR ($p = 1$)	1.1	0.7	1.2	2.1	3.1	6.4	5.4	13.5
EREPR ($p = 2$)	1.8	106.0	90.2	5.3	12,019.5	23.0	26.9	81.4
GREPR ($p = 2$)	1.0	0.8	1.3	1.3	3.0	10.2	7.3	41.2
REPR ($p = 2$)	1.8	10.4	32.0	3.4	383.8	15.4	13.4	45.5
RuleFit	3.5	1.7	3.9	6.6	11.3	17.6	17.2	65.8
Random forests	0.4	0.3	0.5	0.6	1.0	1.6	1.7	4.7
Gradient boosting	0.7	0.4	0.8	1.0	1.5	2.6	2.6	3.6
Linear regression	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

the same testing methodology as in the previous section except that we let GREPR run for up to $S = 500$ iterations and set $\delta = 0$ in the discretization procedure.

The average testing-set MSE levels produced by the various methods are presented in Table 7. Here, the results appear less favorable to our class of methods. GREPR obtains the lowest MSE values for the two "SML" data sets, but the random forest method obtains the lowest MSE on the Parkinsons data sets. Table 8 presents the standard deviation of MSE values obtained on the larger data sets. Again, GREPR obtains the smallest values on the two SML data sets, but random forests obtain the lowest value for the Parkinsons data sets. Finally, Figure 5 shows the evolution of the average MSE with the number of GREPR iterations similarly to Figure 4. Interestingly, there is no evidence of overfitting through 500 iterations.

6. Conclusions and Discussion

6.1. Assessment of REPR and Possible Enhancements

The results on smaller data sets in Section 5.3 indicate that our new regression procedure is of considerable value when a relatively small number of observations are available. For the small data sets, REPR generally outperforms the other methods we examined, both in terms of accuracy and stability of prediction. The preliminary tests in Section 5.4 suggest that these advantages may not necessarily carry over to larger data sets, where it appears that the random forest method, in addition to running more quickly, may sometimes be better able to exploit the availability of additional observations. However, we need to study the behavior of REPR-family methods more thoroughly on data sets of varying dimensions, both in terms of the number of observations m and number of attributes n, before reaching definitive conclusions. To obtain acceptable running times on larger data sets, it may be necessary to explore several additional avenues of parallel computing, such as using parallelism within each branch-and-bound node as well as across the tree and in the

Table 5. Run Time Comparison Between REPR and Our Earlier EREPR Implementation in Eckstein et al. (2017): The "Current" Runs Are REPR on 16 Cores with 150 Column-Generation Iterations

	Run time in seconds		
Method	Current REPR	EREPR from Eckstein et al. (2017)	Speedup
SERVO	0.7	11	16.9
CONCRETE	105.6	1,791	17.0
MACHINE	89.0	187	2.1
YACHT	3.0	36	11.8
MPG	12,017.1	47,398	3.9
COOL	18.7	288	15.4
HEAT	20.6	222	10.8
AIRFOIL	582.3	7,984	13.7

Note. The EREPR runs from Eckstein et al. (2017) also use 16 cores but have 100 column-generation iterations.

Figure 4. (Color online) Test MSE as a Function of Iteration Count for the Smaller Data Sets, Using Logarithmic Vertical Scales

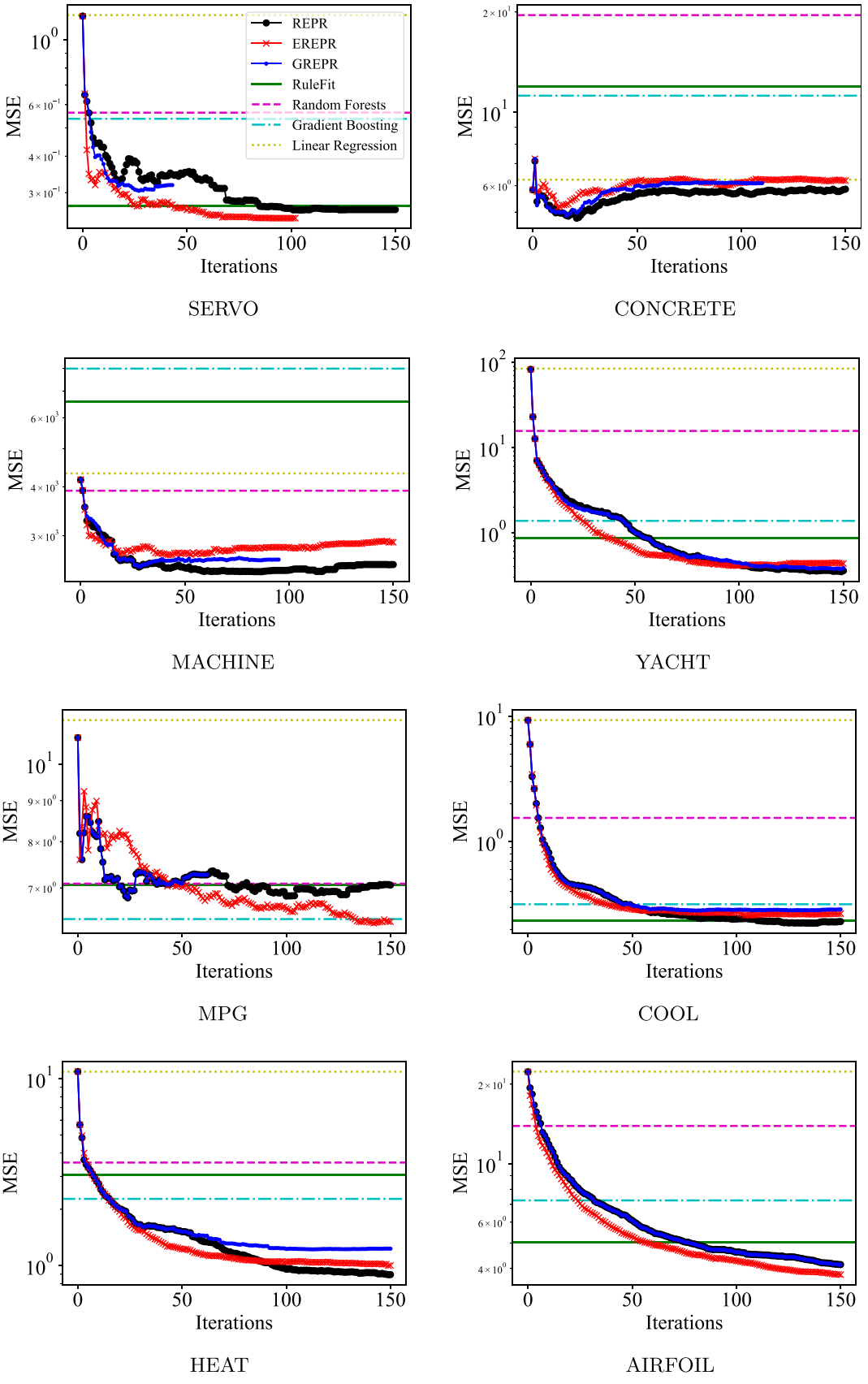


Table 6. Summary of Larger Data Sets

Data set	m	n	$K_0(X)$
SML_dining	4127	12	1.4×10^{67}
SML_room	4127	12	1.4×10^{67}
parkinsons_total	5875	20	2.0×10^{117}
parkinsons_motor	5875	20	2.0×10^{117}

solution of the restricted master problems. We also plan to study some extensions, such as introducing explanatory variables that are the product of a single box rule and single attribute from the base data. In our column-generation context, entering variables of this form could be identified by a modification to the pricing problem resembling $n+1$ invocations of the RMA algorithm or heuristic.

6.2. Relationship to "One-Pass" Mixed-Integer Methods

REPR is an example of applying optimization methods typically associated with operations research to a problem in machine learning. Another example of such an effort is Bertsimas and Dunn (2017), which proposes solving large mixed-integer linear programming (MILP) problems to create multilabel classification trees of some limited depth D; the method is not incremental or greedy but instead solves a single large MILP per problem instance. Because it uses a MILP solver, this method also draws on branch-and-bound techniques. One can build regression models in a similar manner as described in Dunn (2018). The challenge with such a one-pass approach is choosing the maximum tree depth D and the difficulty of the resulting linear or quadratic programming models, whose size grows exponentially with D. We are not aware of a publicly available implementation of these methods for regression, but if one were available it would be interesting to compare the performance of the resulting models with REPR. It appears more difficult to apply a one-pass approach like that of Bertsimas and Dunn (2017) to constructing models of the additive form (2) because it is unclear how to formulate a corresponding mixed-integer model whose continuous relaxation is convex. Furthermore, one would likely need to preselect some limit on the number of rule coefficients γ_k allowed to be nonzero, and the mixed-integer model would have a large number of symmetries that would need to be suppressed for any branch-and-bound-based method to be practical.

6.3. Overfitting Control and Relationship to Other Boosting Regression Methods

The end of Section 3.1 describes how our column-generation algorithm relates to forward selection methods in linear regression: it is similar to applying forward selection to the set of rules $\{r_k\}_{k\in K}$, except that entering variables are selected on the basis of covariance rather than correlation. It could be interesting to explore whether RMA could be used help create rule-based models through other kinds of regression-model building techniques. Some common procedures of this kind include least-squares boosting (see, for example, Friedman 2001, Bühlmann and Yu 2003, Bühlmann 2006, and Bühlmann and Hothorn 2007) and forward stagewise regression (see, for example, Hastie et al. 2009, section 3.3.3). Such methods do not fully reoptimize the regression coefficients at each step as in forward selection or REPR; instead, they simply add a new term to an existing model or adjust the value of one of its coefficients. Other related methods, such as in Efron et al. (2004) and Freund et al. (2017), make somewhat more complicated updates, but far short of full reoptimization. As described in Freund et al. (2017), these methods may be interpreted as a form of subgradient descent in a high-dimensional primal space, whereas our column-generation method may be understood as a cutting plane method in the dual—one iteratively optimizes a problem with an incomplete set of dual constraints (equivalent

Table 7. Average Test MSE over the Larger Data Sets

	Data set			
Method	SML_dining	SML_room	parkinsons_total	parkinsons_motor
GREPR	$\mathbf{1.763 \times 10^{-6}}$	$\mathbf{1.456 \times 10^{-3}}$	6.937×10^{-3}	9.429×10^{-3}
RuleFit	6.741×10^{-6}	3.675×10^{-3}	6.509×10^{-3}	8.906×10^{-3}
Random forests	2.269×10^{-6}	1.501×10^{-3}	$\mathbf{4.491 \times 10^{-3}}$	$\mathbf{5.425 \times 10^{-3}}$
Gradient boosting	1.228×10^{-5}	4.720×10^{-3}	2.595×10^{-2}	2.592×10^{-2}
Linear regression	1.606×10^{-5}	6.933×10^{-3}	1.023×10^{-1}	1.151×10^{-1}

Note. The smallest value in each column is bolded.

Table 8. Standard Deviation of Test MSE over the Larger Data Sets

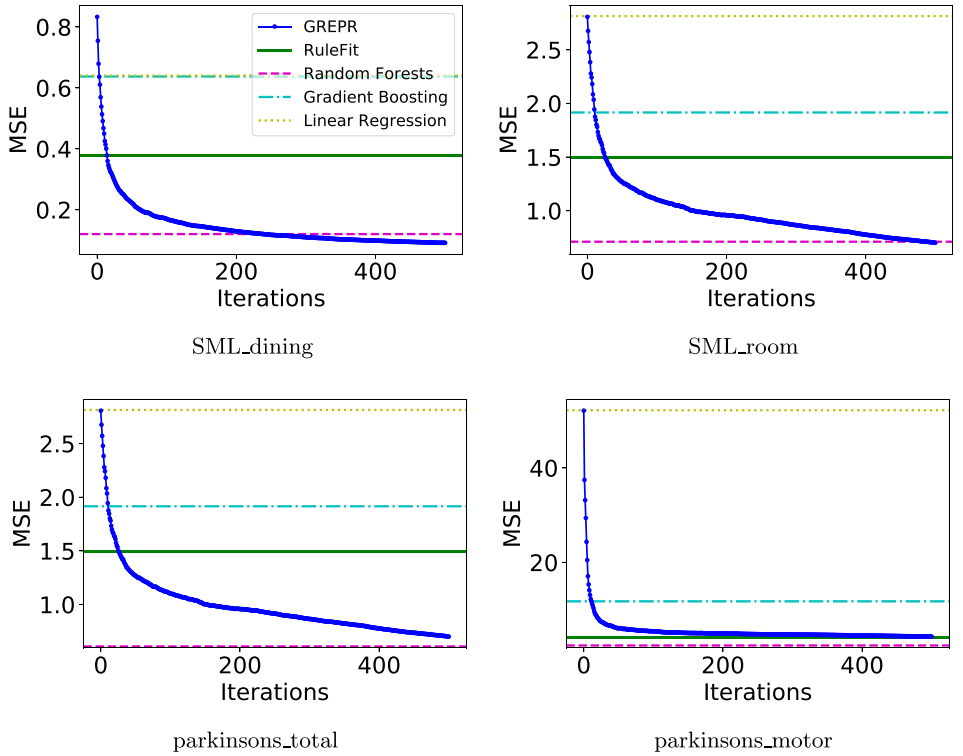
	Data set			
Method	SML_dining	SML_room	parkinsons_total	parkinsons_motor
GREPR	$\mathbf{3.586 \times 10^{-5}}$	$\mathbf{2.045 \times 10^{-4}}$	3.671×10^{-4}	1.269×10^{-3}
RuleFit	3.925×10^{-5}	2.882×10^{-4}	7.370×10^{-4}	4.775×10^{-4}
Random forests	6.511×10^{-5}	2.528×10^{-4}	$\mathbf{3.192 \times 10^{-4}}$	$\mathbf{3.375 \times 10^{-4}}$
Gradient boosting	1.519×10^{-4}	3.389×10^{-4}	1.709×10^{-3}	7.396×10^{-4}
Linear regression	1.976×10^{-4}	7.375×10^{-4}	3.673×10^{-3}	2.829×10^{-3}

Note. The smallest value in each column is bolded.

to primal variables) and then searches for a dual constraint violated by the current solution, which is then added to the model. It seems often stated, for example in Efron et al. (2004), that classical forward selection methods are prone to overfitting, and methods that make simpler, less "aggressive" model updates are less so. Curiously, our experiments show very limited overfitting by REPR despite its close relation to forward selection. Still, it could be interesting to try subgradient-related model-building methods, such as described in Friedman (2001), Bühlmann and Yu (2003), Bühlmann (2006), Bühlmann and Hothorn (2007), and Freund et al. (2017)[1] with box rules generated by RMA or our greedy RMA heuristic. One possible concern is that combining subgradient-related model-building with optimal RMA-based generation of new rules might lead to "unbalanced" algorithms, in which too much effort is spent generating new variables in relation to the work spent adjusting the model coefficients; such methods could run very slowly. Subgradient-related model-building methods may, thus, be better suited to working with static lists of covariates that may be searched relatively quickly. However, the potential mismatch between subgradient-related "master" methods and the subproblems might be less severe if only the greedy RMA heuristic were used.

One possible partial explanation why REPR has little apparent tendency toward overfitting is the presence of L_1 regularizing terms in the objective function in (3). By biasing the regression coefficients γ_k toward zero, they may tend to dampen the rate at which they are adjusted as new rules are added to the model. Thus, the L_1-regularizing terms may have a somewhat similar effect to the "learning rate" parameter in subgradient-related

Figure 5. (Color online) Test MSE as a Function of Iteration Count for the Larger Data Sets



boosting methods such as in Friedman (2001), Bühlmann and Yu (2003), Bühlmann (2006), and Bühlmann and Hothorn (2007). Within the biases introduced by the L_1 terms, however, REPR still fully reoptimizes the regression coefficients at each iteration, which may help it converge faster than techniques such as least-squares boosting.

Of the five values for E that we tested in our inner cross-validation procedure, $E = 0$ was selected about 5% of the time. Furthermore, inspecting iterates for runs with $E = 0$ showed severe overfitting for two data sets, SERVO and MACHINE: in these cases, the MSE on the testing set begins increasing significantly even as the MSE on the training data drops to near zero; the effect is particular strong for MACHINE, for which the testing error becomes many orders of magnitude larger than the training error. These observations accord with the hypothesis in the preceding paragraph, but it also appears that other factors may be at work because $E = 0$ is sometimes selected by cross-validation, and the other data sets do not show the same overfitting pattern as SERVO and MACHINE.

As pointed out in Freund et al. (2017), using a boosting-class method (in our case, based on column generation) and terminating after a certain number of iterations provides a degree of "implicit" regularization and sparsity promotion when constructing a model: if we adjoin only the "best" new covariate to our model at each iteration and stop after some fixed number of iterations S, the remaining covariates (in our case, rules drawn from the large set K) are implicitly suppressed from entering the model. In the case of REPR, the connection between regularization and stopping is apparent from the column-generation stopping criterion $z^* \leq E$ that arises naturally from (6) and (7): once the dual variables μ and ν are fixed, the only real effect of the regularization parameter E is to set the threshold at which rules are considered "good enough" to enter the model. Thus, one might question whether the parameter E is really necessary: perhaps setting the maximum number of iterations S could be sufficient to control the complexity of the resulting model, whose regression coefficients would not be biased toward zero if we effectively take $E = 0$. However, E also affects both the primal and dual solutions identified by each restricted master problem and, thus, the entire path of the algorithm. As hypothesized in the previous paragraph, the result of taking $E > 0$ may be that REPR adjusts the regression coefficents less aggressively than ordinary forward selection, thus helping to avoid overfitting.

Endnote

[1] The "least angle" model-building procedure of Efron et al. (2004) would be more challenging to apply with dynamically generated rules although it might be possible through enumerating multiple near-optimal solutions of the subproblems.

References

Aho T, Ženko B, Džeroski S, Elomaa T (2012) Multi-target regression with rule ensembles. *J. Machine Learn. Res.* 13:2367–2407.
Bentley J (1984) Programming pearls: Algorithm design techniques. *Comm. ACM* 27(9):865–873.
Bertsimas D, Dunn J (2017) Optimal classification trees. *Machine Learn.* 106(7):1039–1082.
Breiman L (2001) Random forests. *Machine Learn.* 45(1):5–32.
Bühlmann P (2006) Boosting for high-dimensional linear models. *Ann. Statist.* 34(2):559–583.
Bühlmann P, Hothorn T (2007) Boosting algorithms: Regularization, prediction and model fitting. *Statist. Sci.* 22(4):477–505.
Bühlmann P, Yu B (2003) Boosting with the L_2 loss: Regression and classification. *J. Amer. Statist. Assoc.* 98(462):324–339.
Cohen WW, Singer Y (1999) A simple, fast, and effective rule learner. Hendler J, Subramanian D, eds. *Proc. 16th Natl. Conf. Artificial Intelligence (AAAI-99)* (AAAI Press, Menlo Park, CA), 335–342.
Dembczyński K, Kotłowski W, Słowiński R (2008a) Maximum likelihood rule ensembles. McCallum A, Roweis S, eds. *Proc 25th Internat. Conf. Machine Learn. (ICML '08)* (ACM, New York), 224–231.
Dembczyński K, Kotłowski W, Słowiński R (2008b) Solving regression by learning an ensemble of decision rules. Rutkowski L, Tadeusiewicz R, Zadeh LA, Zurada JM, eds. *Artificial Intelligence and Soft Computing—ICAISC 2008*, Lecture Notes in Artificial Intelligence, vol. 5097 (Springer, Berlin, Heidelberg), 533–544.
Demiriz A, Bennett KP, Shawe-Taylor J (2002) Linear programming boosting via column generation. *Machine Learn.* 46(1–3):225–254.
Dunn J (2018) Optimal trees for prediction and prescription. Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge.
Eckstein J, Goldberg N (2012) An improved branch-and-bound method for maximum monomial agreement. *INFORMS J. Comput.* 24(2):328–341.
Eckstein J, Goldberg N, Kagawa A (2017) Rule-enhanced penalized regression by column generation using rectangular maximum agreement. *Proc. Machine Learn. Res.* 70:1059–1067.
Eckstein J, Hart WE, Phillips CA (2015) PEBBL: An object-oriented framework for scalable parallel branch and bound. *Math. Programming Comput.* 7(4):429–469.
Efron B, Hastie T, Johnstone I, Tibshirani R (2004) Least angle regression. *Ann. Statist.* 32(2):407–499.
Ford LR Jr, Fulkerson DR (1958) A suggested computation for maximal multi-commodity network flows. *Management Sci.* 5(1):97–101.
Freund RM, Grigas P, Mazumder R (2017) A new perspective on boosting in linear regression via subgradient optimization and relatives. *Ann. Statist.* 45(6):2328–2364.
Friedman JH (2001) Greedy function approximation: A gradient boosting machine. *Ann. Statist.* 29(5):1189–1232.
Friedman JH, Popescu BE (2008) Predictive learning via rule ensembles. *Ann. Appl. Statist.* 2(3):916–954.

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Oper. Res.* 9(6):849–859.
Griva I, Nash SG, Sofer A (2009) *Linear and Nonlinear Optimization*, 2nd ed. (SIAM, Philadelphia).
Gropp W, Lusk E, Skjellum A (1994) *Using MPI: Portable Parallel Programming with the Message-Passing Interface* (MIT Press, Cambridge, MA).
Gurobi Optimization (2016) Gurobi optimizer reference manual. Accessed April 5, 2019, http://www.gurobi.com/documentation/7.0/refman/index.html.
Hastie T, Tibshirani R, Friedman J (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, 2nd ed. (Springer, New York)
Kagawa A (2018) The rectangular maximum agreement problem: Applications and parallel solution. Unpublished doctoral dissertation, Rutgers University, New Brunswick, Newark, and Camden, NJ.
Kagawa A, Eckstein J (2019) "Boosting" GitHub repository. Accessed February 2019, https://github.com/aik7/Boosting.
Lichman M (2013) UCI machine learning repository. Accessed January 2017, http://archive.ics.uci.edu/ml.
Miller AJ (1990) *Subset Selection in Regression* (Chapman and Hall, Boca Raton, FL).
Ridgeway G (2007) Generalized boosted models: A guide to the gbm package. Accessed November 2017, http://www.saedsayad.com/docs/gbm2.pdf.
Tibshirani R (1996) Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. B.* 58(1):267–288.
Weisberg S (2005) *Applied Linear Regression*, 3rd ed. (Wiley, Minneapolis).
Weiss SM, Indurkhya N (1993) Optimized rule induction. *IEEE Expert* 8(6):61–69.