Constraint Optimal Selection Techniques (COSTs) for Nonnegative Linear Programming Problems

Goh Saito[†‡§] H.W. Corley[†‡] Jay M. Rosenberger[†‡] Tai-Kuan Sung[†¶]

[‡]*IMSE Department, The University of Texas at Arlington, PO Box 19017, Arlington, Texas 76019, USA*; [¶]*Nan Kai University of Technology, Taiwan*

(*Updated 13 February 2012*)

We describe an active-set, cutting-plane approach called Constraint Optimal Selection Techniques (COSTs) and develop an efficient new COST for solving nonnegative linear programming problems. We give a geometric interpretation of the new selection rule and provide computational comparisons of the new COST with existing linear programming algorithms for some large-scale sample problems.

Keywords: linear programming; large-scale linear programming; cutting planes; active-set methods; constraint selection;

1. Introduction

1.1. *The Nonnegative Linear Programming Problem*

Linear programming represents a mathematical model for solving numerous practical problems such as the optimal allocation of resources. The general linear programming (LP) model [2] can be stated as the problem P

$$\text{maximize} \quad z = \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{1}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{2}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{3}$$

where $\mathbf{x}$ is an n-dimensional column vector of variables; $\mathbf{A}$ is an $m \times n$ matrix $[a_{ij}]$ with m rows of transposed n-dimensional column vectors $\mathbf{a}_i^{\mathrm{T}}$, $\forall i = 1, \ldots, m$; $\mathbf{b}$ is an m-dimensional column vector; $\mathbf{c}$ is an n-dimensional column vector; and $\mathbf{0}$ is a column vector of zeros of appropriate dimension according to context.

Simplex pivoting algorithms and polynomial interior-point barrier-function methods represent the two principal solution approaches to solve problem P [15]. Unfortunately there is no single best algorithm. For either method, we can always formulate an instance of P for which the method performs poorly [12]. Significantly, however, binary and integer models represent the principal use of LP in industrial applications. Since interior-point methods do not allow the efficient post-optimality analysis of pivoting algorithms in solving such problems, simplex methods remain

---

[†]All four authors contributed equally to this research.

[§]Corresponding author. Email: goh.saito@mavs.uta.edu

the dominant approach. However, current simplex algorithms are not adequate in many situations. In particular, emerging technologies require computer solutions in nearly real time for problems involving millions of constraints or variables.

In this paper we consider the special case of P with $\mathbf{a}_i \geq \mathbf{0}$ and $\mathbf{a}_i \neq \mathbf{0}$, $\forall i = 1, \ldots, m$; $\mathbf{b} > \mathbf{0}$; and $\mathbf{c} > \mathbf{0}$, where all inequalities between vectors are meant componentwise. Such a version of P is called a nonnegative linear program (NNLP) and provides a model for a significant portion of industrial LP applications. Examples include updating airline schedules as weather conditions and passenger loads change [e.g., 12] finding an optimal driving route using real-time traffic data from global positioning satellites [e.g., 6], and detecting problematic repeats in DNA sequences [e.g., 7]. The dual of an NNLP (maximization) problem is considered the standard minimization NNLP problem. We focus here on the maximization case.

NNLPs have the following two useful properties. They are always feasible at the origin $\mathbf{x} = \mathbf{0}$. Moreover, for each $j = 1, \ldots, n$,

$$x_j \leq \min_{i=1,\ldots,m} \left\{ \frac{b_i}{a_{ij}} \middle| a_{ij} > 0 \right\}.$$

Hence NNLPs have both an unbounded feasible region and unbounded objective function if and only if some column of $\mathbf{A}$ is a zero vector. Thus their boundedness is easily verifiable without computation.

1.2. *An Active-set Framework*

Our active-set framework for solving an NNLP P is described as follows. For P with a bounded feasible region as determined above, we begin with a relaxation of P. We next form an initial bounded NNLP P_0 with a single artificial bounding constraint $\mathbf{a}_0^{\mathrm{T}}\mathbf{x} \leq b_0$ in (2) with $\mathbf{a}_0 > \mathbf{0}$ and $b_0 > 0$, together with the nonnegativity constraints of (3). One well-known bounding constraint has the form $\mathbf{1}\mathbf{x} \leq M$ for M sufficiently large enough so as not to reduce the feasible region of P. However, a more efficient bounding constraint is developed for P_0 here.

We then solve P_0 and a series of relaxations P_r, $r = 1, 2, \ldots$, of P by adding constraints from set (2) to the previous relaxation. The constraints of P_r are called its *operative constraints*, while the rest of set (2) are called its *inoperative constraints*. Since the bounding constraint and nonnegativity restrictions form a bounded region, each problem P_r yields an optimal solution $\mathbf{x}_r^*$.

A relaxed problem P_{r+1} is obtained from P_r by adding one or more constraints chosen from the violated inoperative constraints of P_r, i.e., a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ not in P_r for which $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} - b_i > 0$. These constraints are selected in order as the violated inoperative constraints considered most likely to be binding at optimality for the original problem P according to a particular constraint selection criterion.

P_{r+1} is next solved by a dual simplex algorithm. By continuing in this manner, eventually a solution $\mathbf{x}_r^*$ is obtained that satisfies all inoperative constraints for P_r and yields a solution of P. The rationale for any such active-set approach is that a solution to P is determined by relatively few constraints satisfied as equalities – at most n such constraints for the n variables in constraint set (2). Therefore, the goal is to add only constraints likely to be binding at optimality.

1.3. *Background*

Active-set approaches for solving P have been studied by Stone [13], Thompson et al. [14], Adler et al. [1], Zeleny [18], Myers and Shih [8], and Curet [5], with the term

"constraint selection technique" used in Myers and Shih [8]. In such early work, however, the only two selection criteria used for selecting an inoperative constraint violated by the solution to the current relaxed problem were (a) to randomly select a constraint from the violated inoperative constraint and (b) to choose an inoperative constraint most violated by the current problem. For example, method (a) was used in Adler et al. [1], while method (b) was used in Zeleny [18]. In particular, assume that P_0 has a bounding constraint $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M$ and the remaining constraints of (2) are randomly ordered. Then method (a) is called SUB in this paper because it selects the violated inoperative constraint with the smallest SUBscript in (2). Method (b) is called VIOL because it selects the most VIOLated inoperative constraint. VIOL is identical to the Priority Constraint Method of Thompson et al. [14], which is a standard pricing method for delayed column generation in terms of the dual [2].

More recent work on constraint selection has considered the angle that the normal vector $\mathbf{a}_i$ of a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ in (2) forms with the normal vector $\mathbf{c}$ of the objective function (1) as measured by the cosine of the angle between $\mathbf{a}_i$ and $\mathbf{c}$ denoted by $\cos(\mathbf{a}_i, \mathbf{c}) = \frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{\|\mathbf{a}_i\|\|\mathbf{c}\|}$. Geometric considerations in Naylor and Sell [9, pp. 273–274], for example, suggest that a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ with a larger cosine value might more likely be binding at an optimal extreme point of P than one with a smaller value. We call this observation the cosine criterion, which applied to the dual of P gives new pivot rules for the simplex algorithm such as the "most-obtuse-angle" rules studied by Pan [10, 11]. In addition, Trigos et al. [16] and Vieira et al. [17] use the cosine criterion on P for both (2) and (3) to get an initial basis for the simplex algorithm and a reduction in the number of iterations. Finally, Corley et al. [3, 4] choose for P_{r+1} a single inoperative constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ of P_r violating $\mathbf{x}_r^*$ and having the largest $\cos(\mathbf{a}_i, \mathbf{c})$.

From the above work, two factors appear to influence the likelihood of a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ being binding at optimality in an NNLP:

Factor I — the angle of its normal vector $\mathbf{a}_i$ with the normal vector $\mathbf{c}$ of the objective function,

Factor II — the depth of the cut that $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ removes as a violated inoperative constraint from the feasible region of a non-terminal P_r.

In Corley et al. [3], Factor I alone is utilized, while in VIOL described above, only Factor II is employed. We formally define a Constraint Optimal Selection Technique (COST) as an instance of the above active-set framework for solving P in which one or more violated inoperative constraints are added to each P_r according to a constraint selection criterion with a constraint selection metric based on both Factor I and Factor II.

1.4. *Contributions*

The main contributions of this paper are

(i) a RAD constraint selection metric involving Factor I and Factor II that chooses violated inoperative constraints likely to be active at optimality,

(ii) a bounding technique for the initial problem P_0 in which multiple constraints are added according to RAD until each variable x_j is bounded (*multi-bound*), and

(iii) a method of adding multiple cuts which bound every variable x_j in each active-set iteration (*multi-cut*), to enhance RAD and efficiently solve the problem P.

These innovations result in a method superior to other approaches for NNLPs. Specifically, we develop the efficient COST RAD in Section 2 and give a geometric interpretation. In Section 3, we present computational results where RAD is significantly faster for NNLPs than the CPLEX barrier and simplex methods over all densities. In Section 4 we offer conclusions and discuss future research.

2. The COST RAD

2.1. *Constraint Selection Criterion*

An initial bounded problem P_0 is formed by adding a bounding constraint such as $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M$ or by adding multiple constraints as described below in Section 2.2. P_0 is then solved to obtain an initial solution $\mathbf{x}_0^*$. In general, RAD generates P_{r+1} by adding one or more inoperative constraints of P_r that maximize the constraint selection metric $\frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i}$ among all inoperative constraints of P_r violating its solution $\mathbf{x}_r^*$. Since $b_i > 0$, $\mathbf{c} > \mathbf{0}$, $\mathbf{a}_i \geq \mathbf{0}$, and $\mathbf{a}_i \neq \mathbf{0}$ for NNLPs, the value $\frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i} > 0$. Define

$$RAD\left(\mathbf{a}_i, b_i, \mathbf{c}\right) = \frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i}.$$

Thus RAD seeks $i^* \in \underset{i \notin OPERATIVE}{\arg\max} \left(RAD\left(\mathbf{a}_i, b_i, \mathbf{c}\right) \middle| \, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}_r^* > b_i \right)$, or equivalently

$$i^* \in \underset{i \notin OPERATIVE}{\arg\max} \left(\frac{\|\mathbf{a}_i\|}{b_i} \cos\left(\mathbf{a}_i, \mathbf{c}\right) \middle| \, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}_r^* > b_i \right)$$

for each P_r since $\|\mathbf{c}\|$ is constant for all elements of $RAD\left(\mathbf{a}_i, b_i, \mathbf{c}\right)$. Ties are broken arbitrarily. The term $\cos\left(\mathbf{a}_i, \mathbf{c}\right)$ in $RAD\left(\mathbf{a}_i, b_i, \mathbf{c}\right)$ invokes Factor I as in Corley et al. [3, 4], while $\frac{\|\mathbf{a}_i\|}{b_i}$ measures Factor II for the hyperplane $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i$. Although equality constraints are not considered in this paper, it should be noted that any equality constraints could be included in P_0.

2.2. *The COST RAD with Multi-bound and Multi-cut*

The initial bounded problem P_0 is formed by adding multiple constraints from (2) ordered by decreasing value of RAD until no column of $\mathbf{A}$ is a zero vector (Step 1). After an optimal solution to the initial bounded problem is obtained by the primal simplex method (Step 2), subsequent iterations are solved by the dual simplex method (Step 3). Moreover, after the solution of P_0 and each subsequent P_r, constraints are again added in groups. P_{r+1} is formed by selecting inoperative constraints in decreasing order of RAD until a positive coefficient is included for each variable x_j (Step 3, lines 8–15). The following pseudocode depicts the COST RAD.

Step 1 — Identify constraints to initially bound the problem.

1: $\mathbf{a}^* \leftarrow \mathbf{0}$
2: **while** $\mathbf{a}^* \not> \mathbf{0}$ **do**
3: Let $i^* \in \underset{i \notin BOUNDING}{\arg\max}$ RAD$\left(\mathbf{a}_i, b_i, \mathbf{c}\right)$
4: **if** $\exists j \mid a_{ij}^* = 0$ **and** $a_{i^*j} > 0$ **then**
5: $BOUNDING \leftarrow BOUNDING \cup \{i^*\}$

end if
$\mathbf{a}^* \leftarrow \mathbf{a}^* + \mathbf{a}_{i^*}$
end while

Step 2 — Using the primal simplex method, obtain an optimal solution $\mathbf{x}_0^*$ for the initial bounded problem P_0

$$\begin{aligned} \text{maximize} \quad & z = \mathbf{c}^{\mathrm{T}}\mathbf{x} && (4)\\ \text{subject to} \quad & \mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i \quad \forall i \in BOUNDING && (5)\\ & \mathbf{x} \ge \mathbf{0}. && (6) \end{aligned}$$

Step 3 — Perform the following iterations until an optimal solution to problem P is found.

$r \leftarrow 0$
$STOP \leftarrow$ **false**
$OPERATIVE \leftarrow BOUNDING$
while $STOP =$ **false do**
  if $\mathbf{a}_i^{\mathrm{T}}\mathbf{x}_r^* \le b_i \ \forall i \notin OPERATIVE$ **then**
    $STOP \leftarrow$ **true** // $\mathbf{x}_r^*$ is an optimal solution to P.
  else
    $\mathbf{a}^* \leftarrow \mathbf{0}$
    while $\mathbf{a}^* \not> \mathbf{0}$ **or** $OPERATIVE \subset \left\{ i \middle| \, \mathbf{a}_i^{\mathrm{T}}\mathbf{x}_r^* > b_i \right\}$ **do**
      Let $i^* \in \underset{i \notin OPERATIVE}{\arg\max} \ \mathrm{RAD}\left(\mathbf{a}_i, b_i, \mathbf{c} \middle| \, \mathbf{a}_i{}^{\mathrm{T}}\mathbf{x}_r^* > b_i\right)$
      if $\exists j \middle| \, a_{ij}^* = 0$ **and** $a_{i^*j} > 0$ **then**
        $OPERATIVE \leftarrow OPERATIVE \cup \{i^*\}$
      end if
      $\mathbf{a}^* \leftarrow \mathbf{a}^* + \mathbf{a}_{i^*}$
    end while
    $r \leftarrow r + 1$
    Solve the following P_r by the dual simplex method to obtain $\mathbf{x}_r^*$.

$$\begin{aligned} \text{maximize} \quad & z = \mathbf{c}^{\mathrm{T}}\mathbf{x} && (7)\\ \text{subject to} \quad & \mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i \quad \forall i \in OPERATIVE && (8)\\ & \mathbf{x} \ge \mathbf{0}. && (9) \end{aligned}$$

  end if
end while

2.3. *Geometric Interpretation of RAD*

Figure 1 depicts a two-dimensional example of a constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i$ and a current solution $\mathbf{x}_r^*$. Since $\mathbf{c} > \mathbf{0}$, $\mathbf{a}_i \ge \mathbf{0}$, and $\mathbf{a}_i \ne \mathbf{0}$ for each constraint i in set (2), the value $\mathbf{a}_i^{\mathrm{T}}\mathbf{c} > 0$. One might interpret RAD as approximating each constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \le b_i$ by a sphere about the origin with radius equal to the length along the vector $\mathbf{c}$ to its intersection with the hyperplane $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i$. This point is $\frac{b_i}{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}\mathbf{c}$. Since these spheres are nested, the answer to the problem $\hat{\mathrm{P}}$, obtained from P by replacing the set (2) with the spherical approximating constraints, is immediate. It is determined by the hyperplane $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i$ minimizing the norm of such points

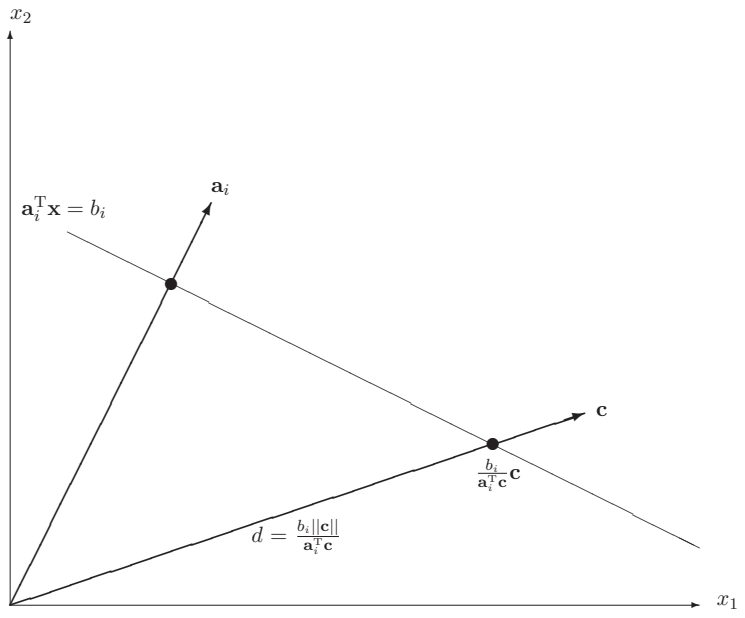

Figure 1. Geometric interpretation of RAD.

in the direction of the gradient $\mathbf{c}$. This minimization yields $\frac{b_{i^*}}{\mathbf{a}_{i^*}^{\mathrm{T}}\mathbf{c}}\mathbf{c}$ as the solution to $\hat{\mathrm{P}}$, where $i^* \in \underset{i=1,\ldots,m}{\arg\max}\left\{\frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i}\right\}$.

This interpretation gives RAD its name. But the constraints are linear, not spherical, so the solution to P is an extreme point. In Figure 1, the constraint selection criterion is to minimize distance d. Or since $\mathbf{c}$ is constant for all constraints (2), we seek to maximize $\frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}{b_i}$ among all inoperative constraints of P_r. Alternately, note that $\frac{b_i\|\mathbf{c}\|}{\mathbf{a}_i^{\mathrm{T}}\mathbf{c}}$ is the inverse of the length of the projection of the vector $\frac{\mathbf{a}_i}{b_i}$ in the direction of vector $\mathbf{c}$. In other words, RAD is directly proportional to the length of the projection of the vector $\frac{\mathbf{a}_i}{b_i}$ in the direction of vector $\mathbf{c}$.

However, the vector $\mathbf{a}_i$ may include some zeros in certain components. In this case, the hyperplane from a single constraint $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ will not bound every variable, even though the approximating spheres in $\hat{\mathrm{P}}$ bound every variable. Consequently, adding multiple constraints to P_r in which there is at least one positive coefficient for each variable x_j will create a "dome-like" convex polytope with a set of hyperplanes around the origin in the n-dimensional positive real space. These multiple constraints more accurately approximate the spheres of $\hat{\mathrm{P}}$. Our computational experiments indicate that RAD performs significantly better with this method of bounding all columns of $\mathbf{A}$ to form bounding constraints and to provide groups of constraints for subsequent iterations.

3. Computational Experiments

The COST RAD was compared with the CPLEX primal simplex method, the CPLEX dual simplex method, and the polynomial interior-point CPLEX barrier method, as well as with the previously defined constraint selection techniques SUB and VIOL used as controls. RAD, SUB, and VIOL utilized the CPLEX dual simplex solver to solve each new relaxed problem P_{r+1}.

Table 1. Randomly generated NNLP problem Set 1.

Number of Variables	1,000
Number of Constraints	200,000
Range of a_{ij}	$1 \leq$ Random Real ≤ 5
Range of b_i	$1 \leq$ Random Real ≤ 10
Range of c_j	$1 \leq$ Random Real ≤ 10

	Average of 5 instances of LPs at each density					
Problem Instance	Density	Minimum number of nonzero a_{ij} in a constraint	Maximum number of nonzero a_{ij} in a constraint	Average number of nonzero a_{ij} in a constraint	Total number of nonzero a_{ij}	Number of binding constraints at optimality
1 – 5	0.005050	2.0	17.8	5.1	1,009,980	726.2
6 – 10	0.006019	2.0	19.8	6.0	1,203,860	717.0
11 – 15	0.007005	2.0	22.4	7.0	1,401,005	701.6
16 – 20	0.008001	2.0	23.8	8.0	1,600,161	673.6
21 – 25	0.009004	2.0	27.0	9.0	1,800,698	668.4
26 – 30	0.009999	2.0	27.4	10.0	1,999,898	665.2
31 – 35	0.020001	3.6	43.8	20.0	4,000,270	571.4
36 – 40	0.029999	10.0	57.6	30.0	5,999,746	513.4
41 – 45	0.040007	15.4	71.8	40.0	8,001,314	488.0
46 – 50	0.049997	22.6	85.2	50.0	9,999,341	460.4
51 – 55	0.060002	29.4	100.0	60.0	12,000,361	433.2
56 – 60	0.069995	37.4	110.0	70.0	13,998,950	422.6
61 – 65	0.080009	44.4	122.4	80.0	16,001,832	396.6
66 – 70	0.089990	51.2	134.2	90.0	17,998,055	384.0
71 – 75	0.099997	61.2	146.6	100.0	19,999,422	372.4
76 – 80	0.199993	146.6	261.6	200.0	39,998,674	310.8
81 – 85	0.300012	235.0	369.6	300.0	60,002,460	262.2
86 – 90	0.399997	333.4	472.0	400.0	79,999,452	233.0
91 – 95	0.499983	429.0	572.6	500.0	99,996,701	202.6
96 – 100	0.750009	684.6	809.0	750.0	150,001,816	158.0
101 – 105	1.000000	1,000.0	1,000.0	1,000.0	200,000,000	111.0

3.1. *Problem Instances*

Three sets of randomly generated NNLPs were constructed. In the first set, NNLPs with 1,000 variables (n) and 200,000 constraints (m) were generated at various densities ranging from 0.005 to 1. Randomly generated real numbers between 1 and 5, 1 and 10, 1 and 10 were assigned to elements of $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ respectively. The number of nonzero a_{ij} in each constraint was binomially distributed $B(n, p =$ density). Additionally, we required each constraint to have at least two nonzero a_{ij}, so that a constraint would not become a simple upper bound on a variable. At each of the 21 densities, 5 random LPs were generated. Table 1 summarizes the NNLPs generated for Set 1.

Set 2 was prepared using the same method as in Set 1, but with different parameters. Here n and m were increased to 5,000 and 1,000,000, respectively; and randomly generated real numbers between 1 and 100 were assigned to elements of $\mathbf{b}$, and $\mathbf{c}$. Since computer memory was limited to approximately 3×10^6 nonzero a_{ij}, 0.06 was the maximum possible density allowed. Hence, much smaller density increments were chosen. Table 2 summarizes the NNLPs generated for Set 2.

The third set, in which the ratio m/n was varied from 200 to 1, is described below in Section 3.3.4.

3.2. *CPLEX Preprocessing*

Before presenting computational results, we discuss two CPLEX parameters. The first parameter is termed PREIND (preprocessing presolve indicator). With the parameter setting PREIND = 1 (ON), the proprietary preprocessing routine in CPLEX is enabled. This routine reduces both the number of constraints m and the number of variables n before the CPLEX primal simplex, dual simplex, or

Table 2. Randomly generated NNLP problem Set 2.

Number of Variables	5,000
Number of Constraints	1,000,000
Range of a_{ij}	$1 \leq$ Random Real ≤ 5
Range of b_i	$1 \leq$ Random Real ≤ 100
Range of c_j	$1 \leq$ Random Real ≤ 100

	Average of 5 instances of LPs at each density					
Problem Instance	Density	Minimum number of nonzero a_{ij} in a constraint	Maximum number of nonzero a_{ij} in a constraint	Average number of nonzero a_{ij} in a constraint	Total number of nonzero a_{ij}	Number of binding constraints at optimality
1 – 5	0.000508	2.0	12.4	2.5	2,540,864	3,641.0
6 – 10	0.000574	2.0	13.2	2.9	2,868,363	3,437.6
11 – 15	0.000650	2.0	14.4	3.3	3,249,509	3,244.0
16 – 20	0.000733	2.0	15.6	3.7	3,667,001	3,149.2
21 – 25	0.000822	2.0	17.0	4.1	4,108,983	3,054.2
26 – 30	0.000914	2.0	17.6	4.6	4,570,717	2,967.8
31 – 35	0.001009	2.0	18.8	5.1	5,047,736	2,891.2
36 – 40	0.002000	2.0	28.4	10.0	9,998,206	2,475.2
41 – 45	0.003000	2.0	38.4	15.0	15,001,505	2,227.6
46 – 50	0.004000	2.8	45.2	20.0	19,999,272	2,068.2
51 – 55	0.004999	5.4	52.6	25.0	24,996,828	1,936.4
56 – 60	0.006000	7.8	59.4	30.0	30,000,841	1,861.2
61 – 65	0.007001	10.0	68.6	35.0	35,003,724	1,766.2
66 – 70	0.008000	13.0	74.4	40.0	40,000,970	1,692.2
71 – 75	0.009001	17.6	79.8	45.0	45,002,765	1,630.8
76 – 80	0.009999	20.2	87.6	50.0	49,996,022	1,598.4
81 – 85	0.020000	56.8	150.6	100.0	99,999,172	1,288.2
86 – 90	0.030000	94.4	211.8	150.0	150,001,867	1,146.8
91 – 95	0.039999	137.2	271.6	200.0	199,995,504	1,028.6
96 – 100	0.050000	175.8	328.2	250.0	249,999,857	968.6
101 – 105	0.060002	224.4	388.4	300.0	300,008,499	901.6

barrier solver is used. Setting PREIND = 0 (OFF) disables the CPLEX presolve routine. With PREIND = 1, the total CPLEX solution time, including the time it takes to perform presolve, is much faster than with PREIND = 0. Indeed, most of the speed of CPLEX is due to its proprietary presolving routines, which reduce problem sizes.

A second preprocessing parameter PREDUAL (preprocessing dual) also affects the manner in which CPLEX processes LPs before it solves them by the primal simplex, dual simplex, or barrier solver. With the parameter setting PREDUAL = 0 (AUTO), CPLEX automatically chooses whether or not to take the dual of the original LP to be solved. With PREDUAL = −1 (OFF), CPLEX does not take the dual. With PREDUAL = 1 (ON), CPLEX always solves the dual LP problem.

The default settings for the two preprocessing parameters are PREIND = 1 (preprocessing presolve ON) and PREDUAL = 0 (preprocessing dual AUTO). Without explicitly setting these parameters to off, CPLEX automatically alters the LPs prior to starting its primal simplex, dual simplex, or barrier solvers. However, the CPLEX barrier solver, even with PREIND = 0 (OFF), always performs a "restricted presolve."

Although the proprietary presolve function is made available to the users of CPLEX, details of the routine are not disclosed. To illustrate its effect, however, problem sets were solved with and without preprocessing for the CPLEX primal simplex and dual simplex solvers. Only default preprocessing parameters were used for the CPLEX barrier solver, however, because its "restricted presolve" cannot be turned off. It is also important to note that no CPLEX preprocessing was implemented by the CPLEX primal simplex and dual simplex solvers as part of RAD, SUB, and VIOL.

Table 3. Comparison of computation times of CPLEX and COST RAD methods on Problem Set 1 (Random NNLP with 1,000 Variables and 200,000 Constraints, $a_{ij} = 1$ to $5, b_i = 1$ to $10, c_j = 1$ to 10).

	CPLEX Primal Simplex	CPLEX Dual Simplex	CPLEX Barrier	RAD
Presolve	On	On	On	Off
Predual	Auto	Auto	Auto	Off
Density	CPU TIME† (std. dev.), sec			
0.005056	6.8 (0.2)	50.0 (6.2)	2.7 (0.1)	2.1 (0.1)
0.006023	10.1 (0.8)	58.8 (5.5)	3.2 (0.1)	2.4 (0.1)
0.007014	12.6 (0.9)	87.0 (3.6)	4.2 (0.2)	2.7 (0.1)
0.007999	15.3 (0.8)	99.2 (6.1)	5.2 (0.4)	2.5 (0.1)
0.009007	18.6 (0.6)	113.4 (5.8)	7.3 (0.4)	2.8 (0.2)
0.010000	22.5 (1.1)	124.8 (7.1)	9.8 (0.5)	2.8 (0.2)
0.019991	40.8 (2.0)	195.5 (8.3)	36.9 (4.5)	3.1 (0.2)
0.029987	46.4 (2.3)	221.9 (6.7)	59.0 (6.4)	3.3 (0.4)
0.040024	50.9 (4.1)	238.1 (13.0)	82.6 (7.8)	3.4 (0.3)
0.050009	52.4 (3.9)	251.7 (25.8)	111.8 (15.7)	3.4 (0.3)
0.059995	58.1 (5.4)	237.0 (12.9)	134.4 (19.0)	3.2 (0.3)
0.069983	62.2 (3.4)	241.1 (15.8)	168.9 (17.5)	3.4 (0.2)
0.080011	64.9 (4.1)	255.0 (21.4)	207.7 (42.9)	3.3 (0.3)
0.089992	63.9 (4.0)	246.8 (15.8)	263.2 (62.2)	3.4 (0.4)
0.099989	68.3 (6.2)	277.4 (41.2)	308.3 (60.0)	3.3 (0.3)
0.199977	79.8 (5.9)	289.8 (5.5)	774.1 (83.1)	4.3 (0.4)
0.300018	87.1 (2.8)	339.0 (27.4)	1,547.2 (160.9)	4.9 (0.6)
0.399917	96.1 (3.6)	383.0 (27.0)	2,379.5 (246.4)	5.6 (0.5)
0.499921	97.4 (2.8)	427.9 (29.0)	3,551.3 (240.1)	6.7 (0.4)
0.750024	116.6 (4.6)	407.7 (25.1)	7,184.4 (486.0)	7.9 (1.0)
1.000000	132.4 (13.3)	315.6 (32.4)	10,628.8 (197.5)	8.1 (0.1)
Average (pooled standard deviation)	57.3 (4.5)	231.5 (19.5)	1,308.1 (144.2)	3.9 (0.4)

†Average of 5 instances of LPs at each density.

3.3. *Computational Results*

Comparisons of computational methods were performed with the IBM CPLEX 12.1 callable library on an Intel Core 2 Duo E8600 3.33GHz workstation with a Linux 64-bit operating system and 8 GB of RAM. Computational test results of Tables 3 through 8 were obtained by calling CPLEX commands from an application written in the programming language C. In these tables, each CPU time presented is an average computation time of solving five instances of randomly generated NNLPs.

Computational results for the CPLEX primal simplex, dual simplex, and barrier solvers for the test problem Set 1 are presented in Table 3. RAD times are also shown for comparison. For NNLPs in Set 1 ($n = 1,000$; $m = 200,000$), CPU times for COST RAD were faster than the CPLEX primal simplex, the CPLEX dual simplex, and the CPLEX barrier linear programming solvers at all densities (ranged from 0.005 to 1). On average, RAD was approximately 15 times faster than the fastest CPLEX solver – primal simplex.

In Table 4, computational results for the CPLEX primal simplex, dual simplex, and barrier solvers for the test problem Set 2 ($n = 5,000$; $m = 1,000,000$) are presented. A reporting limit of 2,400 seconds was used. This number represents the approximate sum of the RAD CPU times over all densities. Again, CPU times for the COST RAD were faster than any of the CPLEX linear programming solvers. On average, RAD was approximately 8 times faster than the fastest CPLEX solver – primal simplex. Again, RAD was both fast and robust for densities restricted to a maximum of 0.06 from memory limitations. Moreover, note that the average RAD times peak at density of roughly 0.01 and then start decreasing. This fact indicates that RAD increases its discriminating power for constraint selection with increasing density, which is further described in Section 3.3.2.

Table 4. Comparison of computation times of CPLEX and COST RAD methods on Problem Set 2 (Random NNLP with 5,000 Variables and 1,000,000 Constraints, $a_{ij} = 1$ to $5, b_i = 1$ to $100, c_j = 1$ to 100).

	CPLEX Primal Simplex	CPLEX Dual Simplex	CPLEX Barrier	RAD
Presolve	On	On	On	Off
Predual	Auto	Auto	Auto	Off
Density	CPU TIME† (std. dev.), sec			
0.000508	11.6 (0.6)	13.7 (1.6)	23.6 (1.2)	7.7 (0.2)
0.000574	28.9 (5.1)	30.4 (1.9)	37.3 (3.9)	12.3 (0.3)
0.000650	14.2 (0.9)	99.4 (1.6)	48.5 (0.5)	16.4 (0.6)
0.000733	21.3 (1.8)	169.3 (3.6)	62.6 (2.3)	22.9 (0.7)
0.000822	31.9 (3.6)	249.2 (13.6)	65.8 (3.3)	28.5 (3.2)
0.000914	40.4 (1.8)	335.8 (8.3)	74.5 (3.5)	35.2 (2.0)
0.001009	50.6 (1.6)	420.8 (13.2)	83.8 (3.8)	41.8 (4.3)
0.002000	179.2 (9.9)	1796.3 (100.0)	160.1 (8.5)	96.2 (3.2)
0.003000	249.9 (8.4)	‡	213.9 (14.6)	116.8 (7.9)
0.004000	305.3 (7.0)	‡	221.7 (38.3)	126.8 (8.1)
0.004999	363.2 (10.2)	‡	228.2 (27.7)	128.5 (11.7)
0.006000	423.1 (23.9)	‡	259.2 (35.2)	136.4 (13.3)
0.007001	469.8 (18.8)	‡	338.0 (21.3)	129.4 (7.8)
0.008000	523.2 (23.2)	‡	325.7 (22.6)	125.3 (10.8)
0.009001	560.8 (22.6)	‡	433.2 (47.9)	116.7 (4.2)
0.009999	619.2 (16.5)	‡	483.4 (51.8)	123.0 (12.2)
0.020000	1235.0 (89.4)	‡	1760.8 (409.4)	92.0 (8.4)
0.030000	1699.4 (73.3)	‡	‡	82.5 (3.0)
0.039999	1942.9 (76.7)	‡	‡	71.5 (4.8)
0.050000	1994.1 (197.7)	‡	‡	72.8 (2.0)
0.060002	2053.3 (60.9)	‡	out of memory	65.7 (5.5)
Average (pooled standard deviation)	610.3 (55.5)	n/a	n/a	78.5 (6.8)

†Average of 5 instances of LPs at each density.

‡Runs with CPU times $> 2,400$ seconds (total CPU time of RAD for all densities) are not reported.

3.3.1. Influences of the Active-set Approach, COST RAD, Multi-bound and Multi-cut

RAD with multi-bound and multi-cut was shown to achieve significantly fast CPU times in Tables 3 and 4. In order to examine the contribution to RAD of its selection metric, active-set, multi-bound, and multi-cut aspects, some relevant computational results are summarized in Tables 5 and 6.

In Table 5, the CPLEX primal simplex, the CPLEX dual simplex, and the CPLEX barrier linear programming solvers are presented, with both the default CPLEX preprocessing setting (preprocessing presolve = ON and preprocessing dual = AUTO) and with these preprocessing parameters turned off. The run times of CPLEX methods are compared to SUB, an essentially random active-set approach that utilizes a single bounding constraint (single-bound) and adds a single constraint per active-set iteration r (single cut). Results for RAD are also listed for reference. As noted earlier, the CPLEX barrier solver always applies its restricted presolve on problems, regardless of the setting specified for the preprocessing parameters. An active-set approach by itself was sufficient to exceed the performance of some CPLEX methods in high density problems. Although it was not competitive in lower density problems, the CPU times of SUB with single-bound and single-cut stayed at about the same level across all densities.

In Table 6, the effects of utilizing the COST RAD, multi-bound and multi-cut in an active-set framework are presented. On average, there is about seven to ten-fold reduction in CPU times, going from an active-set framework utilizing randomly ordered list of constraints (SUB) to one utilizing RAD ordered list of constraints.

Within each of the methods, introducing multi-cut reduced the CPU times about

Table 5. Comparison of computation times to illustrate the effect of applying an active-set method on Problem Set 1 (Random NNLP with 1,000 Variables and 200,000 Constraints, $a_{ij} = 1$ to $5, b_i = 1$ to $10, c_j = 1$ to 10).

	CPLEX Primal Simplex		CPLEX Dual Simplex		CPLEX Barrier†	SUB‡	RAD
Presolve	On	Off	On	Off	On	Off	Off
Predual	Auto	Off	Auto	Off	Auto	Off	Off
Density	CPU TIME§, sec						
0.005056	6.8	644.7	50.0	818.8	2.7	212.9	2.1
0.006023	10.1	627.0	58.8	1,005.2	3.2	236.2	2.4
0.007014	12.6	629.3	87.0	963.6	4.2	260.1	2.7
0.007999	15.3	647.7	99.2	999.8	5.2	257.3	2.5
0.009007	18.6	677.9	113.4	949.5	7.3	276.3	2.8
0.010000	22.5	660.1	124.8	986.4	9.8	282.6	2.8
0.019991	40.8	601.3	195.5	1,113.7	36.9	287.9	3.1
0.029987	46.4	553.6	221.9	1,126.1	59.0	267.6	3.3
0.040024	50.9	489.2	238.1	921.6	82.6	266.3	3.4
0.050009	52.4	263.5	251.7	504.9	111.8	246.4	3.4
0.059995	58.1	262.4	237.0	433.2	134.4	228.9	3.2
0.069983	62.2	272.7	241.1	386.2	168.9	236.6	3.4
0.080011	64.9	265.7	255.0	366.6	207.7	219.7	3.3
0.089992	63.9	265.5	246.8	366.5	263.2	217.9	3.4
0.099989	68.3	276.6	277.4	362.0	308.3	210.8	3.3
0.199977	79.8	304.6	289.8	346.5	774.1	227.7	4.3
0.300018	87.1	322.7	339.0	323.8	1,547.2	235.0	4.9
0.399917	96.1	373.9	383.0	471.3	2,379.5	260.8	5.6
0.499921	97.4	420.2	427.9	263.6	3,551.3	260.1	6.7
0.750024	116.6	596.7	407.7	202.7	7,184.4	264.3	7.9
1.000000	132.4	1,731.5	315.6	131.1	10,628.8	221.0	8.1
Average	57.3	518.4	231.5	621.1	1,308.1	246.5	3.9

†Ran only with presolve on, because it cannot be completely turned off for CPLEX Barrier method.

‡One constraint was added per iteration r. $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M = 10^{10}$ was used as the bounding constraint.

§Average of 5 instances of LPs at each density.

19% and 12% for SUB and RAD respectively, for lower density (≤ 0.03) problems. It makes sense that there is less reduction in high density problems, because fewer constraints are added per multi-cut (i.e. each iteration r) at higher density. The observation is described more in detail in Section 3.3.3. The fact that the performance of multi-cut SUB gets worse particularly at high density problems, whereas that of multi-cut RAD does not, directly reflects the effectiveness of COST RAD, which is further described in Section 3.3.2.

The effect of applying multi-bound is much smaller in magnitude than the other two factors, but the reduction in CPU times could be observed in multi-cut RAD at lower density problems.

3.3.2. Effective Sorting of Constraints by the COST RAD

In the COST RAD algorithm, constraints (2) in problem P are sorted by a constraint selection criterion RAD, as defined in section 2.1. In its implementation in the C language, all constraints can be sorted first *before* iteratively solving P_r for $\mathbf{x}_r^*$ (i.e. Prior COST as defined in [4]). This fact is a significant computational advantage of RAD, which does not depend on $\mathbf{x}_r^*$, over constraint selection metrics such as VIOL or Posterior COSTs [4], which do depend on $\mathbf{x}_r^*$. Once the optimal solutions to the test problems were found, the effectiveness of RAD in selecting constraints likely binding at optimality was evaluated by finding the distribution of constraints actually binding at optimality in the RAD-sorted list of constraints, as shown in Figure 2. For a problem with density of 1 from Set 1, all binding rows at optimality were found in the first 1.0% of the sorted constraints. Although the distribution became wider for NNLPs with lower density, all binding rows at optimality were found at worst within 6.5% of the sorted list. Figure 2 also shows

Table 6. Comparison of computation times to illustrate the effects of COST RAD, multi-bound and multi-cut on Problem Set 1 (Random NNLP with 1,000 Variables and 200,000 Constraints, $a_{ij} = 1$ to $5, b_i = 1$ to $10, c_j = 1$ to 10).

	SUB				RAD			
bound cut	single† single	multi single	single† multi	multi multi	single† single	multi single	single† multi	multi multi
Density	CPU TIME‡, sec							
0.005056	212.9	216.7	10.5	11.5	25.2	25.2	2.7	2.1
0.006023	236.2	240.3	12.0	11.9	29.0	30.0	3.0	2.4
0.007014	260.1	274.2	12.8	14.1	33.5	33.8	2.9	2.7
0.007999	257.3	269.8	13.0	13.5	37.1	32.5	2.8	2.5
0.009007	276.3	287.7	14.1	14.1	38.7	35.8	3.1	2.8
0.010000	282.6	294.9	14.8	15.0	36.5	36.9	3.1	2.8
0.019991	287.9	283.6	16.4	17.2	38.2	39.2	3.4	3.1
0.029987	267.6	272.5	16.3	18.2	37.9	40.5	3.5	3.3
0.040024	266.3	267.5	16.7	17.7	33.6	34.2	3.4	3.4
0.050009	246.4	246.0	15.9	16.8	32.4	33.4	3.5	3.4
0.059995	228.9	227.6	15.1	15.7	27.9	28.5	3.2	3.2
0.069983	236.6	231.1	15.5	15.8	27.0	27.1	3.4	3.4
0.080011	219.7	216.0	15.2	15.2	24.2	24.2	3.3	3.3
0.089992	217.9	215.2	15.0	15.0	22.4	22.2	3.4	3.4
0.099989	210.8	207.8	14.6	14.6	21.1	21.0	3.5	3.3
0.199977	227.7	229.9	18.0	17.8	15.0	15.1	4.2	4.3
0.300018	235.0	240.2	21.7	21.8	11.3	11.3	4.8	4.9
0.399917	260.8	268.3	28.3	28.6	10.3	10.4	5.2	5.6
0.499921	260.1	267.9	34.1	34.0	8.9	8.9	6.4	6.7
0.750024	264.3	272.0	57.7	57.6	8.5	8.5	8.1	7.9
1.000000	221.0	228.0	227.5	227.1	8.1	8.1	8.1	8.1
Average	246.5	250.3	28.8	29.2	25.1	25.1	4.0	3.9

†$\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M = 10^{10}$ was used as the bounding constraint.

‡Average of 5 instances of LPs at each density.

the steepness of the curves. 95% of binding rows at optimality were found in the top 1.8% of the sorted constraints, and 99% of binding rows at optimality were found in the top 3.0% of the sorted constraints, over all densities in Set 1.

3.3.3. Number of Constraints Added

In Table 7, CPU times and number of constraints added during computation of the test problems Set 1 by COST RAD are compared with the constraint selection methods SUB and VIOL of [18] and [2], respectively. "Number of Constraints Added" includes both constraints added in $BOUNDING$ and $OPERATIVE$ sets. To implement SUB and VIOL as in previous work, a single bounding constraint $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M = 10^{10}$ was used, whereas RAD added a set of multiple constraints described by $BOUNDING$ in Step 1 of the pseudocode.

In our implementations of RAD, we sorted arrays of pointers to the constraint sets using qsort in ANSI C. However, the blocks of memory for the original constraints are not in the same order as the sorted sets of pointers. Searching for new constraints in RAD can thus cause extra steps in accessing memory. Consequently, our implementations reallocate a new block of memory for the constraint set and then copy the content of the original block in the order of the sorted pointers. The maximum (and average in parentheses) CPU time required to calculate RAD criteria, sort the array and copy the block of memory were 1.6, 0.1 and 3.3 (0.3, 0.1 and 0.7) seconds, respectively, for problem Set 1.

An average computation time of 246.5 and 118.5 seconds for SUB and VIOL, respectively, were faster than the 518.4 and 621.1 seconds of CPLEX primal simplex and dual simplex, respectively, with the preprocessing parameters turned off. This fact supports the merit of active-set methods in general, where successive calls to the CPLEX dual simplex algorithm involved only a $r \times r$ basis matrix for the relaxed

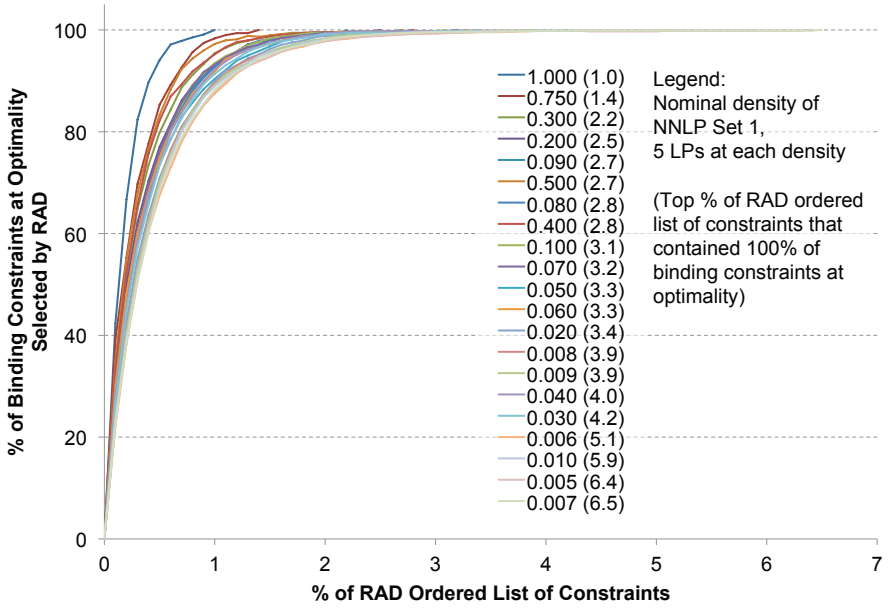

Figure 2. Binding constraints at optimality found in RAD-sorted list of constraints on problem Set 1 (random NNLP with 1,000 variables and 200,000 constraints, densities 0.005 to 1), average of 5 instances of LPs at each density.

problem P_r, even when constraints were added one at a time in SUB and VIOL. Furthermore, in the RAD implementation, a group of constraints was added per iteration r, thereby reducing the total number of calls to the CPLEX dual simplex algorithm. The number of iterations r for RAD was 33.2 on average.

Among the active-set methods, RAD with multi-bound and multi-cut had the shortest CPU time by an order of magnitude. VIOL has about the same number of added constraints as RAD on average, but it has higher percent of constraints added that are binding at optimality than RAD on average. Its CPU times are also slower. The reason is that VIOL is a function of $\mathbf{x}_r^*$. In other words, VIOL makes use of *posterior* information, while RAD does not depend on $\mathbf{x}_r^*$. Finally, in the column labeled "% Reduction in # of Original Constraints at Methods Optimal Solution," RAD eliminated an average of 99.5% of the original constraints in solving the test problems. In this measure, SUB, and VIOL were competitive. However, this fact stems from the active-set framework itself. Even the random SUB eliminated an average of 98.1% of the original constraints.

Comparing the results of single-bound, single-cut version of RAD from Table 6 to VIOL indicates the importance of considering both Factors I and II. In particular, Factor II alone in VIOL produced significantly worse computational times. This observation indicates that the selection rules of RAD might improve pricing methods for delayed column generation, of which VIOL is a standard approach.

3.3.4. Varying m/n Ratio

In Table 8, the third set of NNLPs with various dimensions for the matrix **A** were generated by varying the ratio m/n from 200 to 1. For LPs with a square or narrow ($m \geq n$) **A** matrix, CPU times for RAD were faster than any of the CPLEX methods except for both a ratio of m/n = 20 and a density less than or equal to 0.009. In that one anomalous instance, the CPLEX barrier solver had minimally faster times. The essence of Table 8 is shown in Figure 3. The superior overall performance of RAD is apparent across all densities and m/n ratios. It should be noted that problems P with $m < n$ can be optimized by solving the dual of P with

Table 7. Comparison of computation times of COST RAD and non-COST methods, SUB and VIOL on problem Set 1 (random NNLP with 1,000 variables and 200,000 constraints, $a_{ij} = 1$ to $5, b_i = 1$ to $10, c_j = 1$ to 10).

Density	CPU TIME†, sec			Number of Added Constraints (and number of iterations r for RAD)†			% of Constraints Added by Method and also Binding at Optimality†			% Reduction in # of Original Constraints at Method's Optimal Solution†		
	SUB‡	VIOL‡	RAD	SUB‡	VIOL‡	RAD	SUB‡	VIOL‡	RAD	SUB‡	VIOL‡	RAD
0.005056	212.9	43.3	2.1	6,542.2	1,685.6	1,588.4 (7.4)	11.1%	43.1%	45.7%	96.7%	99.2%	99.2%
0.006023	236.2	49.0	2.4	6,385.8	1,647.8	1,551.8 (7.4)	11.2%	43.3%	45.9%	96.8%	99.2%	99.2%
0.007014	260.1	52.4	2.7	6,322.6	1,588.4	1,508.6 (7.0)	10.9%	43.3%	45.6%	96.8%	99.2%	99.2%
0.007999	257.3	51.8	2.5	6,058.6	1,503.0	1,440.2 (7.4)	11.1%	44.8%	46.7%	97.0%	99.2%	99.3%
0.009007	276.3	56.9	2.8	6,111.2	1,498.8	1,424.4 (8.0)	10.9%	44.5%	46.8%	96.9%	99.3%	99.3%
0.010000	282.6	59.1	2.8	5,927.2	1,462.6	1,396.2 (7.8)	11.0%	44.6%	46.8%	97.0%	99.3%	99.3%
0.019991	287.9	67.3	3.1	5,014.4	1,196.8	1,181.8 (9.6)	11.5%	48.0%	48.7%	97.5%	99.4%	99.4%
0.029987	267.6	75.8	3.3	4,405.6	1,078.6	1,074.6 (10.6)	11.9%	48.7%	48.9%	97.8%	99.5%	99.5%
0.040024	266.3	81.2	3.4	4,166.0	983.2	1,010.6 (12.6)	11.7%	49.7%	48.4%	97.9%	99.5%	99.5%
0.050009	246.4	88.3	3.4	3,810.0	928.0	971.4 (13.4)	12.5%	51.2%	48.9%	98.1%	99.5%	99.5%
0.059995	228.9	90.9	3.2	3,558.2	859.4	910.6 (14.0)	12.1%	50.3%	47.4%	98.2%	99.6%	99.5%
0.069983	236.6	99.2	3.4	3,441.2	832.2	886.0 (15.8)	12.2%	50.6%	47.5%	98.3%	99.6%	99.6%
0.080011	219.7	101.4	3.3	3,221.2	780.4	857.8 (16.8)	12.1%	50.0%	45.5%	98.4%	99.6%	99.6%
0.089992	217.9	107.3	3.4	3,107.0	753.0	819.8 (17.6)	12.6%	51.9%	47.7%	98.4%	99.6%	99.6%
0.099989	210.8	110.4	3.3	3,003.8	713.4	802.8 (18.2)	13.1%	55.1%	49.0%	98.5%	99.6%	99.6%
0.199977	227.7	166.9	4.3	2,323.2	569.8	693.2 (27.8)	13.4%	54.8%	45.0%	98.8%	99.7%	99.7%
0.300018	235.0	199.9	4.9	1,925.4	475.4	603.6 (35.0)	13.8%	56.0%	44.1%	99.0%	99.8%	99.7%
0.399917	260.8	227.1	5.6	1,686.2	415.4	556.0 (43.4)	13.7%	55.6%	41.5%	99.2%	99.8%	99.7%
0.499921	260.1	238.6	6.7	1,468.4	354.2	498.4 (50.4)	14.2%	58.7%	41.7%	99.3%	99.8%	99.8%
0.750024	264.3	275.9	7.9	1,042.2	278.0	395.2 (71.8)	15.5%	58.3%	41.0%	99.5%	99.9%	99.8%
1.000000	221.0	245.3	8.1	670.6	185.0	295.6 (294.6)	16.6%	60.0%	37.6%	99.7%	99.9%	99.9%
Average	246.5	118.5	3.9	3,818.6	942.3	974.6 (33.2)	12.5%	50.6%	45.7%	98.1%	99.5%	99.5%

†Average of 5 instances of LPs at each density.

‡One constraint was added per iteration r. $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq M = 10^{10}$ was used as the bounding constraint.

a corresponding dual version of RAD

$$j^* \in \underset{j \notin OPERATIVE}{\arg\min} \left(\frac{\mathbf{a}^{j\mathrm{T}}\mathbf{b}}{c_j} \middle| \mathbf{a}^{j\mathrm{T}}\mathbf{y}_r^* < c_j \right),$$

where $\mathbf{a}^j$ is the j^{th} column of $\mathbf{A}$.

4. Conclusions

An efficient COST RAD with multi-bound and multi-cut was developed here. A geometric interpretation was given, and the new technique was tested on sets of large-scale randomly generated NNLPs with $m \gg n$. RAD substantially outperformed the CPLEX primal simplex, dual simplex and barrier solvers for NNLPs (maximization) with long-and-narrow $\mathbf{A}$ matrices with various densities, from very sparse to 1. Moreover, RAD maintains the attractive features of the dual simplex method such as a basis, shadow prices, reduced costs, and sensitivity analysis [13]. RAD also retains the post-optimality advantages of pivoting algorithms useful for integer programming. Furthermore, as a practical matter, a well-defined description of RAD is in this paper and hence in the public domain, unlike the CPLEX preprocessing routines.

Computational results obtained by the authors but not reported here suggest that the speed of current methods for large scale LP derives from preliminary problem reduction techniques. However, the COST framework takes the alternate approach of dynamically building up problems, and RAD is in the public domain unlike the CPLEX preprocessing routines. Further research should provide even more effective constraint selection metrics for this purpose. In particular, simultaneously addressing both the primal and dual problems could conceivably improve our approach by adding both constraints and variables.

Future work will be directed at improving the COST RAD by using the local information at each $\mathbf{x}_r^*$ as well as the global metric information. In addition, a forthcoming paper by the authors will generalize RAD to any LP problem, not just NNLPs.

5. Acknowledgements

We gratefully acknowledge the Texas Advanced Research Program for supporting this material under Grant No. 003656-0197-2003, and Lloyd Clarke for his kind assistance on the use of CPLEX.

References

[1] I. Adler, R. Karp, and R. Shamir, *A family of simplex variants solving an $m \times d$ linear program in expected number of pivots steps depending on d only*, Mathematics of Operations Research 11 (1986), pp. 570–590.

[2] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali, *Linear Programming and Network Flows, 3rd ed.*, John Wiley, New York, 2005.

[3] H.W. Corley, J.M. Rosenberger, W.-C. Yeh, and T.-K. Sung, *The cosine simplex algorithm*, International Journal of Advanced Manufacturing Technology 27 (2006), pp. 1047–1050.

[4] H.W. Corley and J.M. Rosenberger, *System, method and apparatus for allocating resources by constraint selection*, U.S. Patent 8,082,549, issued December 20, 2011. http://patft1.uspto.gov/netacgi/nph-Parser?patentnumber=8082549

[5] N. Curet, *A primal-dual simplex method for linear programs*, Operations Research Letters 13 (1993), pp. 223–237.

Table 8. Comparison of computation times of CPLEX and COST RAD methods on random NNLP, varying m/n ratio (random NNLP with $a_{ij} = 1$ to $5, b_i = 1$ to $10, c_j = 1$ to 10).

	RAD				CPLEX Primal Simplex†				CPLEX Dual Simplex†				CPLEX Barrier†			
n	1,000	3,163	10,000	14,143	1,000	3,163	10,000	14,143	1,000	3,163	10,000	14,143	1,000	3,163	10,000	14,143
m	200,000	63,246	20,000	14,143	200,000	63,246	20,000	14,143	200,000	63,246	20,000	14,143	200,000	63,246	20,000	14,143
m/n	200	20	2	1	200	20	2	1	200	20	2	1	200	20	2	1
Nominal Density	CPU Time (sec), average of 5 instances of LPs at each density															
0.005	2.1	32.3	124.0	133.9	6.8	73.4	517.5	566.1	50.0	794.5	1,625.3	1,635.3	2.7	24.9	626.3	1,693.9
0.006	2.4	32.1	107.0	132.6	10.1	79.5	477.6	541.2	58.8	878.1	1,572.8	2,077.5	3.2	27.1	631.2	1,771.8
0.007	2.7	33.0	107.1	117.7	12.6	83.1	484.0	548.3	87.0	890.8	1,469.0	1,308.9	4.2	28.2	634.3	1,653.3
0.008	2.5	32.5	98.5	99.4	15.3	80.8	483.4	483.6	99.2	881.0	1,401.7	2,147.7	5.2	32.3	621.2	1,695.7
0.009	2.8	33.5	93.8	91.9	18.6	82.9	463.4	472.8	113.4	932.1	1,336.8	2,031.4	7.3	33.0	625.6	1,679.4
0.01	2.8	32.0	85.0	91.1	22.5	85.0	440.2	478.9	124.8	913.7	1,283.9	2,003.2	9.8	33.8	628.6	1,640.5
0.02	3.1	25.2	54.3	50.9	40.8	84.7	407.5	410.8	195.5	930.4	777.3	763.9	36.9	45.2	687.9	1,780.4
0.03	3.3	21.3	38.7	39.5	46.4	78.0	408.9	437.1	221.9	1,219.0	615.1	609.0	59.0	67.4	730.4	1,802.6
0.04	3.4	19.6	32.6	34.1	50.9	72.2	534.2	499.4	238.1	1,504.6	453.2	978.7	82.6	91.8	817.3	1,922.4
0.05	3.4	17.3	29.3	30.3	52.4	73.4	631.4	594.7	251.7	1,978.1	423.8	577.7	111.8	123.7	897.4	2,065.0
0.06	3.2	16.7	27.0	27.2	58.1	71.5	711.8	692.0	237.0	2,018.3	362.4	534.0	134.4	162.0	971.7	2,294.0
0.07	3.4	14.5	25.7	26.4	62.2	73.2	814.2	882.2	241.1	2,481.6	319.3	439.2	168.9	195.9	1,144.9	2,442.9
0.08	3.3	14.7	22.8	25.1	64.9	72.4	919.9	954.7	255.0	2,855.7	302.4	682.6	207.7	237.0	1,244.6	2,641.7
0.09	3.4	13.0	24.0	22.9	63.9	72.4	1,135.1	814.9	246.8	2,619.2	277.4	348.9	263.2	304.6	1,359.9	2,733.1
0.1	3.3	13.5	21.6	22.4	68.3	75.8	1,163.3	775.8	277.4	2,226.2	252.7	395.1	308.3	343.6	1,470.7	‡
0.2	4.3	10.8	19.5	20.6	79.8	75.9	919.2	620.4	289.8	1,415.1	189.3	352.0	774.1	1,104.5	‡	‡
0.3	4.9	11.1	18.9	19.9	87.1	75.2	831.3	558.7	339.0	1,267.8	148.9	271.1	1,547.2	2,440.0	‡	‡
0.4	5.6	11.4	19.1	19.1	96.1	85.4	793.9	552.3	383.0	1,309.5	139.0	228.8	2,379.5	‡	‡	‡
0.5	6.7	10.8	18.6	19.7	97.4	84.9	772.6	507.9	427.9	1,106.9	142.1	192.1	‡	‡	‡	‡
0.75	7.9	10.7	17.1	18.2	116.6	102.2	654.2	492.9	407.7	1,070.8	141.5	196.2	‡	‡	‡	‡
1	8.1	12.8	17.9	19.9	132.4	111.5	440.7	349.7	315.6	717.1	169.0	195.9	‡	‡	‡	‡
Average	3.9	19.9	47.7	50.6	57.3	80.6	666.9	582.6	231.5	1,429.1	638.2	855.7	n/a	n/a	n/a	n/a

†Presolve = ON, Predual = Auto

‡Runs with CPU times $> 3,000$ seconds are not reported.

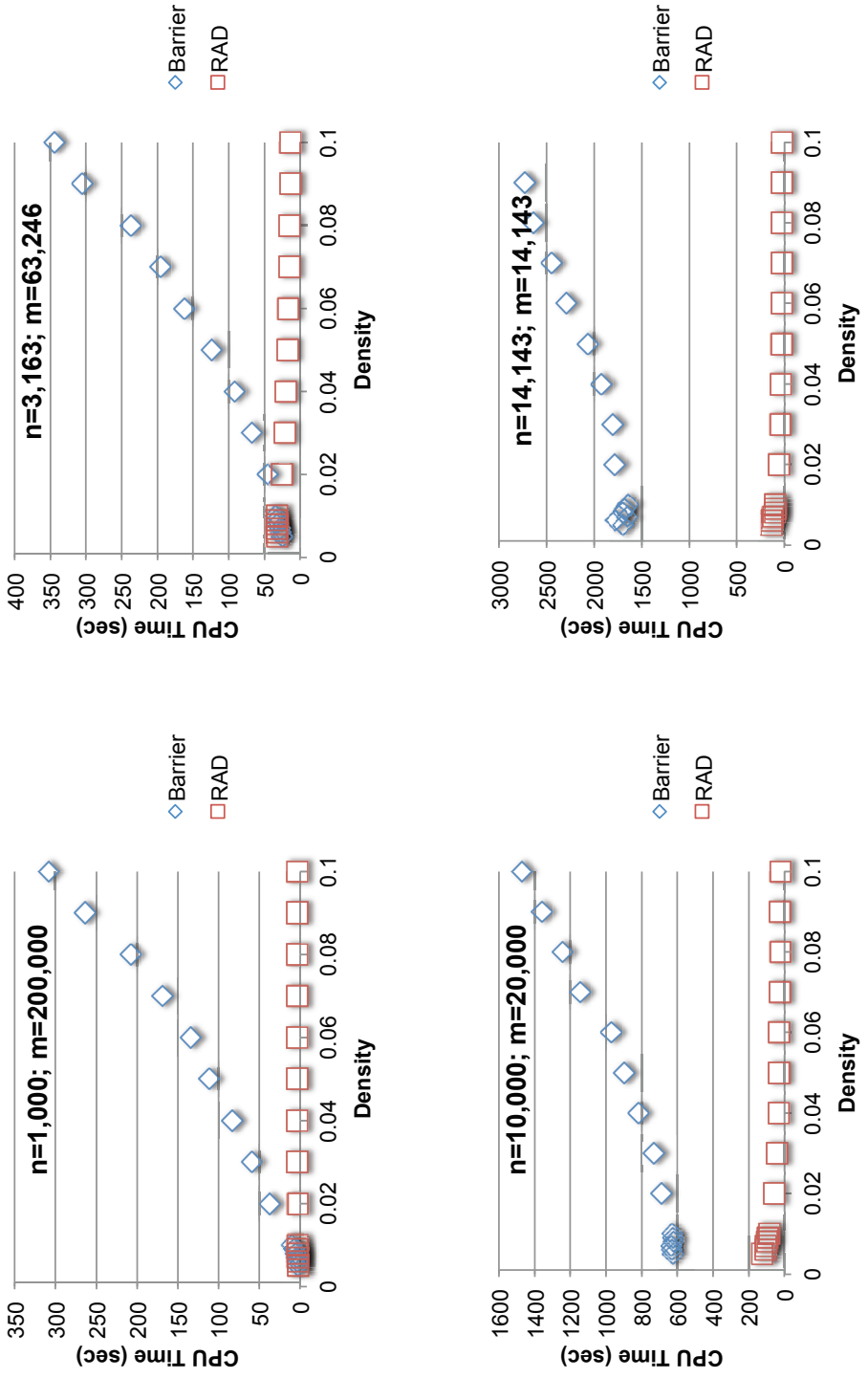

Figure 3. Graphical representation of data for CPLEX barrier and COST RAD presented in Table 8.

[6] P. Dare and H. Saleh, *GPS network design: logistics solution using optimal and near-optimal methods*, Journal of Geodesy 74 (2000), pp. 467–478.
[7] H.L. Li and C.J. Fu, *A linear programming approach for identifying a consensus sequence on DNA sequences*, Bioinformatics 21 (2005), pp. 1838–1845.
[8] D. Myers and W. Shih, *A constraint selection technique for a class of linear programs*, Operations Research Letters 7 (1988), pp. 191–195.
[9] A. Naylor and G. Sell, *Linear Operator Theory in Engineering and Science*, Springer-Verlag, New York, 1982.
[10] P.-Q. Pan, *Practical finite pivoting rules for the simplex method*, OR Spektrum 12 (1990), pp. 219–225.
[11] P.-Q. Pan, *A simplex-like method with bisection for linear programming*, Optimization 22 (1991), pp. 717–743.
[12] J.M. Rosenberger, E.L. Johnson, and G.L. Nemhauser, *Rerouting aircraft for aircraft recovery*, Transportation Science 37 (2003), pp. 408–421.
[13] J. Stone, *The cross-section method: an algorithm for linear programming*, in *Rand Corporation Memorandum P-1490*, Santa Monica, CA, 1958.
[14] G. Thompson, F. Tonge, and S. Zionts, *Techniques for removing nonbinding constraints and extraneous variables from linear programming problems*, Management Science 12 (1966), pp. 588–608.
[15] M.J. Todd, *The many facets of linear programming*, Mathematical Programming 91 (2002), pp. 417–436.
[16] F. Trigos, J. Frausto-Solis, and R.R. Rivera-Lopez, *A simplex cosine method for solving hard linear problems*, Advances in Simulation, System Theory and Systems Engineering, WSEAS Press, 70X (2002), pp. 27–32.
[17] H. Vieira and M.P. Estellita-Lins, *An improved initial basis for the simplex algorithm*, Computers and Operations Research 32 (2005), pp. 1983–1993.
[18] M. Zeleny, *An external reconstruction approach (ERA) to linear programming*, Computers and Operations Research 13 (1986), pp. 95–100.