American Journal of Operations Research, 2015, 5, 526-535
Published Online November 2015 in SciRes. http://www.scirp.org/journal/ajor
http://dx.doi.org/10.4236/ajor.2015.56041

A Dynamic Active-Set Method for Linear Programming

Alireza Noroziroshan*, H. W. Corley, Jay M. Rosenberger

IMSE Department, The University of Texas at Arlington, Arlington, USA
Email: *alireza.norozi.en@gmail.com

Received 25 October 2015; accepted 15 November 2015; published 18 November 2015

Abstract

An efficient active-set approach is presented for both nonnegative and general linear programming by adding varying numbers of constraints at each iteration. Computational experiments demonstrate that the proposed approach is significantly faster than previous active-set and standard linear programming algorithms.

Keywords

Constraint Optimal Selection Techniques, Dynamic Active-Set Methods, Large-Scale Linear Programming, Linear Programming

1. Introduction

Consider the linear programming problem

$$\text{(P)} \quad \text{Max } z = \boldsymbol{c}^T \boldsymbol{x} \tag{1}$$

$$s.t.$$

$$\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} \tag{2}$$

$$\boldsymbol{x} \ge \boldsymbol{0}, \tag{3}$$

where $\boldsymbol{x}$ and $\boldsymbol{c}$ are the n-dimensional column vectors of variables and objective coefficients, respectively, and z represents the objective function. The matrix $\boldsymbol{A}$ is an $m \times n$ matrix $\left[a_{ij}\right]$ with row vectors $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m$; $\boldsymbol{b}$ is an m-dimensional column vector; and $\boldsymbol{0}$ is an n-dimensional column vector of zeros. The non-polynomial simplex methods and polynomial interior-point barrier-function algorithms illustrate the two different approaches to solve problem P. There is no single best algorithm [1]. For any existing approach, there is a problem instance for

*Corresponding author.

How to cite this paper: Noroziroshan, A., Corley, H.W. and Rosenberger, J.M. (2015) A Dynamic Active-Set Method for Linear Programming. *American Journal of Operations Research*, **5**, 526-535. http://dx.doi.org/10.4236/ajor.2015.56041

which the developed method performs poorly [2] [3]. However, interior point methods do not provide efficient post-optimality analysis, so the simplex algorithm is the most frequently used approach [2], even for sparse large scale linear programming problems where barrier methods perform extremely well. In fact, the simplex method has been called "the algorithm that runs the world" [4]; yet it often cannot efficiently solve the large scale LPs required in many applications.

In this paper we consider both the general linear program (LP) and the special case with $\boldsymbol{a}_i \geq \boldsymbol{0}$ and $\boldsymbol{a}_i \neq \boldsymbol{0}$, $\forall i = 1, \cdots, m$; $\boldsymbol{b} > \boldsymbol{0}$; and $\boldsymbol{c} > \boldsymbol{0}$, which is called a nonnegative linear program (NNLP). NNLPs have some useful properties that simplify their solution, and they model various practical applications such as determining an optimal driving route using global positioning data [5] and updating airline schedules [6], for example. We propose active-set methods for LPs and NNLPs. Our approach divides the constraints of problem P into operative and inoperative constraints at each iteration. Operative constraints are those active in a current relaxed subproblem P_r, $r = 1, 2, \cdots,$ of P, while the inoperative ones are constraints of the problem P not active in P_r. In our active-set method we iteratively solve P_r, $r = 1, 2, \cdots,$ of P after adding one or more violated inoperative constraints from (2) to P_{r-1} until the solution $\boldsymbol{x}_r^*$ to P_r is a solution to P.

Active-set methods have been studied by Stone [7], Thompson *et al*. [8], Myers and Shih [9], and Curet [10], for example. More explicitly, Adler *et al*. [11] suggested a random constraint selection rule, in which a violated constraint was randomly added to the operative set. Bixby *et al*. [1] developed a sifting method for problems having wide and narrow structure. In effect, the sifting method is an active-set method for the dual. Zeleny [12] used a constraint selection rule called VIOL here, which added the constraint most violated at each iteration. Mitchell [13] used a multi-cut version of the VIOL for an interior point cutting plane algorithm. Corley *et al.* [14] developed a cosine simplex algorithm where a single violated constraint maximizing $\cos(\boldsymbol{a}_i, \boldsymbol{c}) = \dfrac{\boldsymbol{a}_i \boldsymbol{c}}{\|\boldsymbol{a}_i\| \|\boldsymbol{c}\|}$ was added to the operative set. Junior and Lins [15] used a similar cosine criterion to determine an improved initial basis for the simplex algorithm.

References [6] [16] and [17] are the most directly related to the current work. The constraint optimal selection technique (COST) RAD was introduced in [6] and [16] as a constraint selection metric for NNLPs, and then generalized in [17] to GRAD for LPs. In RAD and GRAD, an initial problem P_0 is formulated from P such that all variables are bounded by at least one constraint, which may be an artificial bounding constraint $\boldsymbol{c}^T \boldsymbol{x} \leq M$ for sufficiently large M. Multiple violated constraints are then added to problems P_r, $r = 0, 1, \cdots,$ according to the constraint selection metric RAD or GRAD. The contribution here is to improve significantly the speed of RAD and GRAD by varying the number of added constraints at each P_r according to an empirically derived function estimating the effectiveness of that iteration.

The paper is organized as follows. The constraint selection metric and dynamic active-set conjunction with RAD and GRAD is given in Section 2. In Section 3, we describe the problem instances and CPLEX preprocessing settings. Section 4 contains computational experiments for both NNLPs and LPs. Then the performance of the new methods is compared to the previous ones as well as the CPLEX simplex, dual simplex, and barrier method. In Section 5, we present conclusions.

2. A Dynamic Active-Set Approach

The purpose of our dynamic active-set method is to add violated constraints to problem P_r more effectively than in [6] and [17]. In COST RAD of [6] for NNLPs we use the constraint selection metric

$$\mathrm{RAD}(\boldsymbol{a}_i, b_i, \boldsymbol{c}) = \frac{\boldsymbol{a}_i^T \boldsymbol{c}}{b_i} \tag{4}$$

to order constraints from highest to lowest value of RAD as a geometric heuristic for determining the constraints most likely to be binding at optimality. Moreover, at each iteration in [6] we add violated constraints in order of decreasing RAD until the added constraints contain non-zero coefficients for all variables. In similar fashion for COST GRAD of [17], we use the constraint selection metric

$$\mathrm{GRAD}(\boldsymbol{a}_i, b_i, \boldsymbol{c}) = \sum_{j=1, c_j > 0}^{n} \frac{a_{ij} c_j}{b_i^+} - \sum_{j=1, c_j < 0}^{n} \frac{-a_{ij}}{b_i^+}, \tag{5}$$

where

$$b_i^+ = \begin{cases} b_i - \min\limits_{k=1,\cdots,m}\left[b_k\right] + \varepsilon & \text{if } \min\limits_{k=1,\cdots,m}\left[b_k\right] < 0 \\ b_i & \text{Otherwise} \end{cases} \tag{6}$$

for a small positive constant ε and $i^* \in \operatorname{argmax}\left(\text{GRAD}\left(\boldsymbol{a}_i, b_i, \boldsymbol{c}\right) : \boldsymbol{a}_i^T \boldsymbol{x}_r^* > b_i,\ i \notin \text{OPERATIVE}\right)$. However, in COST GRAD, violated constraints are added until every column of $\boldsymbol{A}$ has at least one positive and one negative value.

In this paper we propose a dynamic method that adds a varying number of constraints to P_r that depends on the progress made at P_{r-1}. No equality constraints are considered here, but any equality constraints can be included in P_0. An active-set function is defined to compensate for the lack of progress in P_{r-1} by adding more violated constraints at P_r. The algorithm stops when the solution $\boldsymbol{x}_r^*$ to P_r is the optimal solution to P. Of course, one could simply add all violated constraints at any one iteration. However, the proposed dynamic method attempts to balance the number of iterations required to eliminate all constraints violation, while still maintaining an efficient active-set method.

2.1. Dynamic Active-Set Approach for NNLP

The dynamic active-set approach developed for solving NNLPs is as follows. Constraints are initially ordered by the RAD constraint selection metric (4). To construct P_0, we choose constraints from (2) in descending order of RAD until each variables x_j has an $a_{ij} > 0$ in the coefficient matrix of P_0. We say the variables are covered by the constraints of the initial problem P_0. P_0 is then solved by the primal simplex to achieve an initial solution $\boldsymbol{x}_0^*$. Now let γ_0 be the number of constraints in the original problem P and, in general, let γ_r be the number of constraints of problem P violated by $\boldsymbol{x}_r^*$. At each iteration r, the total number of violated constraints γ_r is computed, and the improvement percentage is calculated by

$$\omega_r = \max\left\{0, \left(\frac{\gamma_{r-1} - \gamma_r}{\gamma_{r-1}}\right)\right\} \times 100,\ \forall r = 1, 2, \cdots, \tag{7}$$

where $\omega_r > 0$ represents the percent of improvement made in reducing the total number of violated constraints at iteration r. Next, with [.] denoting the greatest integer function, let

$$\varphi_{r+1} = \left[\varphi_r * \left(1 + \log\left(101 - \omega_r\right)\right)\right],\ 0 \le \omega_r \le 100,\ \forall r = 1, 2, \cdots, \tag{8}$$

where $\varphi_1 = 100$. The value φ_r is an upper bound on the possible number of non-operative violated constraints that can be added at active-set iteration $r = 1, 2, \cdots$. The actual number added is $\min\left\{\varphi_{r+1}, \gamma_r\right\}$. The active-set iterations terminate when $\gamma_r = 0$ and therefore $\omega_r = 100$. Equation (8) was developed from the results of computational experiments.

Pseudocode for the dynamic active-set NNLPs is as follows.

Step 1—Identify constraints to initially bound the problem.

1: $\boldsymbol{a}^* \leftarrow 0$
2: *while* $\boldsymbol{a}^* \not> \boldsymbol{0}$ *do*
3: *Let* $i^* \in \operatorname{argmax}_{i \notin \text{BOUNDING}} \text{RAD}\left(\boldsymbol{a}_i, b_i, \boldsymbol{c}\right)$
4: *if* $\exists\, j \,|\, a_j^* = 0$ and $a_{i^* j} > 0$ *then*
5: BOUNDING $\leftarrow$ BOUNDING $\cup \left\{i^*\right\}$
6: *end if*
7: $\boldsymbol{a}^* \leftarrow \boldsymbol{a}^* + \boldsymbol{a}_{i^*}$
8: *Optimized* $\leftarrow$ *false*
9: *end while*

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

$$\text{maximize } z = \boldsymbol{c}^T \boldsymbol{x}$$

$$\text{subject to } \boldsymbol{a}_i^T \boldsymbol{x} \le b_i \ \forall i \in \text{BOUNDING}$$

$$\boldsymbol{x} \ge \boldsymbol{0}$$

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

1: $\varphi_1 \leftarrow \#100$
2: $r \leftarrow 0$
3: $\gamma_0 \leftarrow \#rows$
4: *while* $(Optimized = false)$ *do*
5: $r \rightarrow r+1$
6: *if* $\{\boldsymbol{a}_i^T \boldsymbol{x}_r^* > b_i, \forall i = 0, \cdots, rows\}$ *then* calculate γ_r
7: Calculate $\omega_r = \max\left\{0, \left(\frac{\gamma_{r-1} - \gamma_r}{\gamma_{r-1}}\right)\right\} * 100$
8: *if* $0 \le \omega_r < 100$ *then* $\varphi_{r+1} = \left[\varphi_r * \left(1 + \log\left(101 - \omega_r\right)\right)\right]$
9: *Let* $i^* \in \text{argmax}_{i \notin \text{OPERATIVE}} \left\{\text{RAD}\left(\boldsymbol{a}_i, b_i, \boldsymbol{c}\right) : \boldsymbol{a}_i^T \boldsymbol{x}_r^* > b_i\right\}$
10: *for* (i = 0 to $\min\{\varphi_{r+1}, \gamma_r\}$) OPERATIVE $\leftarrow$ OPERATIVE $\cup \{i^*\}$ *end*
11: Solve the following P_r by the dual simplex method to obtain $\boldsymbol{x}_r^*$
12: *else if* (ω_r = 100) *then Optimized* $\leftarrow$ *true*// $\boldsymbol{x}_r^*$ is an optimal solution to P.
13: *end if*
14: *else Optimized* $\leftarrow$ *true*// $\boldsymbol{x}_r^*$ is an optimal solution to P.
15: *end if*
16: *end while*

2.2. Dynamic Active-Set Approach for LP

The dynamic active-set approach for solving LP similar to the one for NNLPs. We construct P_0 by choosing a number of constraints ρ_1 from (2) in descending order of GRAD from (2) until no variable x_j is left without at least either a positive or a negative coefficient. In addition, we include an artificial bounding constraint $\mathbf{1}^T \boldsymbol{x} \le M$. If $\rho_1 < 100$, then set $\rho_1 = 100$. Then P_0 is solved to obtain an initial solution $\boldsymbol{x}_0^*$. As in Section 2.1, it is initially assumed that all constraints are violated $(\gamma_0 = m)$. Then the relative improvement percent ω_r is calculated by (7) for P_r and P_{r+1}. Now let

$$\rho_{r+1} = \left[\rho_r * \log\left(101 - \omega_r\right)\right], 0 \le \omega_r \le 100, \forall r = 1, 2, \cdots, \tag{9}$$

where the value ρ_r is an upper bound on the possible number of non-operative violated constraints that can be added at active-set iteration $r = 1, 2, \cdots$. The actual number added is $\min\{\rho_{r+1}, \gamma_r\}$. As ω decreases, ρ_{r+1} increases in (9) to add more violated constraints to P_{r+1}. The algorithm stops at 100% reduction in the number of violated constraints.

Pseudocode for the dynamic active-set for LPs is as follows.

Step 1—Identify constraints to initially bound the problem.

1: $\boldsymbol{a}^* \leftarrow 0$
2: *while* $\boldsymbol{a}^* \not> 0$ *do*
3: *Let* $i^* \in \text{argmax}_{i \notin \text{BOUNDING}} \text{GRAD}\left(\boldsymbol{a}_i, b_i, \boldsymbol{c}\right)$
4: *if* $\exists j \left| a_j^* = 0 \right.$ *and* $a_{i^* j} > 0$ *or* $a_{i^* j} < 0$ *then*
5: BOUNDING $\leftarrow$ BOUNDING $\cup \{i^*\}$
6: *end if*
7: $\boldsymbol{a}^* \leftarrow \boldsymbol{a}^* + \boldsymbol{a}_{i^*}$
8: *Optimized* $\leftarrow$ *false*
9: *end while*

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

$$\text{maximize } z = \boldsymbol{c}^T \boldsymbol{x}$$

$$\text{subject to } \boldsymbol{a}_i^T \boldsymbol{x} \le b_i \ \forall i \in BOUNDING$$

$$\mathbf{1}^T \boldsymbol{x} \le M$$

$$\boldsymbol{x} \ge \mathbf{0}$$

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

1: $\rho_1 \leftarrow \text{Max}\{\#\text{BOUNDING}, 100\}$
2: $r \leftarrow 1$
3: $\gamma_0 \leftarrow \#rows$
4: $while(Optimized = false)\, do$
5: $\quad r \leftarrow r+1$
6: $if\ \{\boldsymbol{a}_i^T \boldsymbol{x}_r^* > b_i, \forall i = 0, \cdots, rows\}\ then$ calculate γ_r
7: $\quad Calculate\ \omega_r = \max\left\{0, \left(\frac{\gamma_{r-1} - \gamma_r}{\gamma_{r-1}}\right)\right\} * 100$
8: $\quad if\ 0 \le \omega_r < 100\ then\ \rho_{r+1} = \left[\rho_r * \log(101 - \omega_r)\right]$
9: $\quad Let\ i^* \in \text{argmax}_{i \notin \text{OPERATIVE}} \{\text{GRAD}(\boldsymbol{a}_i, b_i, \boldsymbol{c}) : \boldsymbol{a}_i^T \boldsymbol{x}_r^* > b_i\}$
10: $\quad for\ (i = 0\ \text{to}\ \min\{\rho_{r+1}, \gamma_r\})$ OPERATIVE $\leftarrow$ OPERATIVE $\cup \{i^*\}\ end$
11: $\quad$ Solve the following P_r by the dual simplex method to obtain $\boldsymbol{x}_r^*$
12: $end\ if$
13: $else\ Optimized \leftarrow true$// $\boldsymbol{x}_r^*$ is an optimal solution to P.
14: $end\ if$
15: $end\ while$

3. Problem Instances and CPLEX Preprocessing

Four sets of NNLPs used in [6] are considered to evaluate the performance of the developed algorithm. Each problem set contains five problem instances for 21 different density levels and for varying ratios of (m constraints)/(n variables) from 200 to 1. Each set contains 105 randomly generated NNLPs with various densities p ranging from 0.005 to 1. Randomly generated real numbers between 1 and 5, 1 and 10, 1 and 10 were assigned to the elements of $\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ respectively. To avoid having a constraint in the form of an upper bound on a variable, each constraint is required to have at least two non-zero a_{ij}. For general LP, a problem set containing 105 randomly generated by Saito *et al.* [17] is compared with the dynamic approach of this paper. These LP problems contain 1000 variables (n) and 200,000 constraints (m), with various densities ranging from 0.005 to 1 and the randomly generated a_{ij} ranging between −1 and −5 or between 1 and 5.

Two parameters that CPLEX uses for solving linear programming are PREIND (preprocessing pre-solve indicator) and PREDUAL (preprocessing dual). As described in [17] and [6], when parameter setting PREIND = 1 (ON), the preprocessing pre-solver is enabled and both the number of variables and the number of constraints is reduced before any type of algorithm is used. By setting PREIND = 0 (OFF) the pre-solver routine in CPLEX is disabled. PREDUAL is the second preprocessing parameter in CPLEX. By setting parameter PREDUAL = 0 (ON) or −1 (OFF), CPLEX automatically selects whether to solve the dual of the original LP or not. Both are used with the default settings for the CPLEX primal simplex method, the CPLEX dual simplex method, and the CPLEX barrier method. Neither CPLEX pre-solver nor PREDUAL parameters were used in any part of the developed dynamic active-set methods for NNLPs and LPs.

4. Computational Experiments

The computations were performed on an Intel Core (TM) 2 Duo X9650 3.00 GHz with a Linux 64-bit operating system and 8 GB of RAM. The developed methods use IBM CPLEX 12.5 callable library to solve linear programming problems. The dynamic RAD and dynamic GRAD are compared with the previously developed COST RAD and COST GRAD, respectively, as well as VIOL, the CPLEX primal simplex method, the CPLEX dual simplex method, and the CPLEX barrier method.

4.1. Computational Results for NNLP

Table 1 illustrates the performance comparison between dynamic RAD method and the previously defined constraint selection technique COST RAD on Set 1 to Set 4 for various dimensions of the matrix $\boldsymbol{A}$ used in [6]. Both methods are compared with the CPLEX barrier method (interior point), the CPLEX primal simplex method, and the CPLEX dual simplex method. The worst performance occurs at m/n ratio of 200, where on average, dynamic

Table 1. Results from dynamic RAD and COST RAD for set 1-set 4, (random, NNLP $a_{ij} = 1 - 5$, $b_i = 1 - 10$, $c_j = 1 - 10$).

		Dynamic RAD[+]				COST RAD[+]			
	n	1000	3163	10,000	14,143	1000	3163	10,000	14,143
	m	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
Density	No	CPU time, sec[++]							
0.005	1	2.02	30.88	108.52	126.93	2.10	30.82	108.70	127.55
0.006	2	2.47	32.19	106.48	113.68	2.42	31.48	104.87	114.03
0.007	3	2.63	30.39	96.61	104.34	2.65	29.41	92.45	104.18
0.008	4	2.46	31.03	90.14	89.24	2.54	30.63	88.20	90.73
0.009	5	2.67	28.89	82.66	86.57	2.78	30.10	83.53	85.21
0.01	6	2.73	28.22	75.46	83.66	2.79	27.81	77.90	80.43
0.02	7	2.88	23.17	45.55	49.82	3.09	24.69	47.63	49.95
0.03	8	2.83	17.97	33.85	37.35	3.22	20.49	36.68	38.33
0.04	9	2.92	15.24	29.23	28.98	3.33	19.06	32.74	32.53
0.05	10	2.97	14.10	24.83	26.37	3.34	16.97	28.23	28.59
0.06	11	2.86	11.93	23.38	24.45	3.20	14.94	27.58	27.27
0.07	12	2.94	11.21	20.38	21.08	3.41	14.88	23.59	23.79
0.08	13	2.87	10.25	19.47	21.43	3.32	13.57	23.44	24.19
0.09	14	3.05	9.33	19.43	20.49	3.38	12.67	23.09	23.80
0.1	15	3.20	9.33	18.03	18.78	3.39	12.92	22.93	20.85
0.2	16	4.39	8.07	14.86	16.50	4.30	11.09	18.87	20.31
0.3	17	5.26	8.19	13.77	15.27	4.97	10.58	18.11	19.46
0.4	18	6.40	9.19	14.32	15.60	5.76	12.31	18.55	18.88
0.5	19	7.80	9.84	14.33	15.97	6.98	11.92	18.00	19.89
0.75	20	10.86	11.91	14.55	16.26	8.26	12.01	17.19	18.06
1	21	12.93	12.01	12.61	14.58	8.39	12.20	17.71	18.50
Average		4.24	17.30	41.83	45.11	3.98	19.07	44.28	46.98

[+]Used CPLEX preprocessing parameters of presolve = off and predual = off. [++]Average of 5 instances of LPs at each density.

RAD is 8% faster than COST RAD for densities less than 0.2 and 18% slower for densities above 0.2. When the density increases, dynamic RAD shows an increase in computation time more than that of COST RAD. On the other hand, for an *m*/*n* ratio of 20 the CPU times decrease with an increase in density. For higher densities above 0.01, dynamic RAD is more efficient and takes less computation times than COST RAD. On average, dynamic RAD is 10% more efficient than COST RAD. For an *m*/*n* ratio of 2 at densities higher than 0.009, the data show that COST RAD starts taking significantly more time than dynamic RAD. Dynamic RAD was 5.5% faster than COST RAD over all densities and 21% faster on average for densities above 0.5. For an *m*/*n* ratio of 1 with densities greater than 0.01, dynamic RAD is about 8% more efficient than COST RAD. On average, dynamic RAD is superior performance to COST RAD for problem sets 2, 3, and 4.

Table 2 from [6] is presented to provide an immediate comparison of the developed dynamic RAD method with the standard CPLEX solvers. A reporting limit of 3000 seconds was used. On average, the CPU times for dynamic RAD were faster than any of the CPLEX solvers across all densities and ratios. However, CPLEX barrier methods show smaller CPU times when ratio *m*/*n* = 20 and the density is less than or equal to 0.01.

Table 2. Results from the CPLEX primal, the dual simplex, and the barrier method for set 1-set 4, (random, NNLP $a_{ij} = 1 - 5$, $b_i = 1 - 10$, $c_j = 1 - 10$) [6].

		Primal[¯¯]				Dual[¯¯]				Barrier[¯¯]			
	n	1000	3163	10,000	14,143	1000	3163	10,000	14,143	1000	3163	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
	m/n	200	20	2	1	200	20	2	1	200	20	2	1
Density	No	CPU time, sec[++]											
0.005	1	7.01	71.02	228.51	309.83	54.84	762.62	1597.24	1169.04	2.36	14.52	240.17	650.83
0.006	2	10.36	77.28	245.60	291.07	60.29	803.97	1607.16	2413.42	2.39	16.30	224.08	666.54
0.007	3	12.98	75.84	219.72	265.09	91.39	876.85	1483.20	1702.47	3.04	18.34	233.55	671.56
0.008	4	15.72	82.01	206.45	239.30	100.06	912.75	1445.54	1236.76	3.90	20.70	232.38	668.82
0.009	5	19.25	80.35	196.72	216.23	114.95	898.99	1375.73	427.95	4.76	22.66	232.23	649.26
0.01	6	21.92	78.50	182.47	216.60	123.49	912.63	1252.05	436.31	5.53	24.29	228.76	650.30
0.02	7	39.90	78.80	118.28	127.59	203.08	963.66	807.29	362.34	17.13	32.08	242.54	711.26
0.03	8	45.42	79.75	98.02	108.60	217.18	1207.76	545.91	723.98	28.79	45.03	266.90	727.61
0.04	9	50.30	78.78	89.75	88.32	248.75	1489.40	450.08	539.92	41.50	62.28	292.15	806.80
0.05	10	55.16	78.92	81.09	82.14	256.49	1746.46	418.69	519.50	53.72	81.32	327.01	837.67
0.06	11	60.34	77.49	77.28	78.27	251.39	2124.31	378.71	409.47	67.58	100.48	359.53	897.58
0.07	12	62.07	78.93	70.44	70.37	251.74	2446.69	310.89	544.15	84.70	125.49	401.72	948.01
0.08	13	62.92	76.96	70.21	69.81	264.48	2799.62	307.25	388.94	99.51	149.37	454.01	1038.86
0.09	14	66.57	79.07	71.46	72.37	258.14	2523.03	718.04	427.95	119.26	186.06	495.28	1153.31
0.1	15	71.00	74.57	67.43	62.64	287.36	2251.10	267.14	436.31	138.67	207.54	539.64	1194.56
0.2	16	87.49	83.12	64.38	62.99	294.39	1450.82	201.73	362.34	379.68	691.77	1298.76	2529.97
0.3	17	94.57	77.91	67.14	66.61	341.44	1280.71	175.16	267.16	657.45	1333.29	2418.75	[b]
0.4	18	99.33	78.46	73.58	71.48	384.10	1236.30	146.09	233.39	985.86	2076.09	[b]	[b]
0.5	19	111.30	84.30	86.50	75.62	427.16	1173.49	133.49	208.65	1350.82	[b]	[b]	[b]
0.75	20	128.26	99.35	115.00	102.51	410.98	1056.18	132.25	181.95	[b]	[b]	[b]	[b]
1	21	207.55	94.09	393.54	145.96	375.89	411.19	148.90	165.45	[b]	[b]	[b]	[b]
Average		63.30	80.26	134.46	134.45	238.93	1396.60	662.03	626.55	n/a	n/a	n/a	n/a

[¯¯]Used CPLEX preprocessing parameters of presolve = ON and predual = Auto; [++]Average of 5 instances of LPs at each density; [b]Runs with CPU times > 3000 s are not reported.

4.2. Computational Results for LP

Table 3 shows computational results for the CPLEX primal simplex method, the dual simplex method, and the interior point barrier method for the general LP problem set used in [17]. CPU times for COST GRAD and VIOL using both the multi-cut technique and dynamic approaches are presented for comparison. Dynamic GRAD is stable over the range of densities. In addition, its performance is superior to multi-cut GRAD for every problem instance. Average CPU times for GRAD using multi-cut method and dynamic approach are 43.87 and 24.57 seconds, respectively, a 42% improvement in computation time. Average computation times for GRAD and VIOL using dynamic approach are 24.57 seconds vs. 33.82 seconds, respectively.

It should be noted that GRAD captures more information than VIOL in higher densities to discriminate between constraints. Interestingly, when the dynamic active-set is used for both GRAD and VIOL, their CPU

Table 3. Comparison of computation times of CPLEX solvers, GRAD, and VIOL using both dynamic active-set and multi-cut method on general LP problem set (random LP with 1000 variables and 200,000 constraints [17]).

No	Density	Constraint selection metric[+]				CPLEX[¯¯]		
		GRAD	VIOL	GRAD	VIOL	Primal	Dual	Barrier
		Multi-cut method		Dynamic active-set				
		CPU time, sec[++]						
1	0.005	9.85	12.31	7.96	9.26	40.99	23.05	2.39
2	0.006	11.48	14.50	9.44	11.11	84.56	35.52	2.62
3	0.007	13.36	14.21	10.60	12.58	128.65	48.62	3.79
4	0.008	14.24	14.67	12.09	13.00	183.70	61.56	4.93
5	0.009	15.41	15.32	12.25	14.57	212.79	75.34	6.06
6	0.01	16.55	17.09	14.10	15.33	256.70	92.11	7.33
7	0.02	24.74	22.24	20.93	21.79	396.55	205.25	15.86
8	0.03	27.84	24.30	22.91	26.21	460.01	295.18	26.63
9	0.04	30.55	24.47	23.87	29.52	602.73	350.86	35.26
10	0.05	37.59	28.72	28.52	33.57	617.29	396.10	46.76
11	0.06	34.29	26.58	26.86	33.80	656.22	438.92	59.55
12	0.07	37.46	28.05	26.91	34.34	729.43	465.61	71.65
13	0.08	36.28	26.29	25.54	33.46	739.21	510.10	82.98
14	0.09	37.97	27.74	24.60	33.21	823.11	521.89	94.01
15	0.10	39.50	28.30	25.99	35.61	956.17	554.29	108.03
16	0.20	56.26	36.64	27.97	41.28	1456.41	759.66	280.09
17	0.30	60.93	42.40	28.41	40.68	1664.83	900.12	527.05
18	0.40	74.58	56.97	33.39	52.19	2033.10	1057.27	760.07
19	0.50	85.02	71.35	36.85	54.68	1925.32	1334.80	1076.40
20	0.75	113.02	116.78	39.44	59.53	2232.88	1571.28	2132.53
21	1.00	144.35	173.02	57.22	104.58	2301.76	1717.25	3267.10
Average		43.87	39.14	24.57	33.82	881.07	543.56	410.05

[+]Used CPLEX preprocessing parameters of presolve = off and predual = off. $\mathbf{1}^T\boldsymbol{x} \le M = 10^{10}$ was used as the bounding constraint; [++]Average of 5 instances of LPs at each density; [¯¯]Used CPLEX preprocessing parameters of presolve = ON and predual = Auto.

times are significantly faster than the same metrics with the multi-cut method. GRAD using the multi-cut technique takes the longest computation time in comparison to others at higher densities. Unlike the proposed dynamic approach, the LP algorithm COST GRAD requires checking the signs of the nonzero a_{ij} and therefore more computation time for higher densities. The efficiency of VIOL decreases significantly with increasing density. On average, dynamic GRAD is approximately 35 times faster than the CPLEX primal simplex, 21 times faster than the CPLEX dual simplex, and 17 times faster the CPLEX barrier linear programming solvers without preprocessing. The superior overall performance of GRAD using dynamic approach is apparent across all densities in general LP set.

For comparison purposes, **Table 4** shows GRAD and VIOL computation times when a fixed number of violated constraints is added at each iteration. Adding a fixed number of constraints is examined for both GRAD and VIOL. At densities below 0.03, dynamic GRAD takes less CPU time than the fixed-cut approach. GRAD

Table 4. Comparison of computation times of GRAD using dynamic active-set and fixed cut method on general LP problem set (random LP with 1000 variables and 200,000 constraints [17]).

Constraint selection metric[+]									
No	Density	GRAD	VIOL	GRAD			VIOL		
		Dynamic active-set		Fixed number of constraints					
				100	500	1000	100	500	1000
				CPU times, sec[++]					
1	0.005	7.96	9.26	14.58	10.04	8.05	10.49	8.82	11.56
2	0.006	9.44	11.11	18.75	13.14	9.48	12.61	10.57	14.45
3	0.007	10.60	12.58	20.32	13.24	10.67	13.87	12.22	15.01
4	0.008	12.09	13.00	23.57	12.63	12.05	14.97	12.82	16.22
5	0.009	12.25	14.57	23.16	13.60	12.32	15.88	14.00	18.35
6	0.01	14.10	15.33	26.04	14.83	13.59	17.72	15.35	19.26
7	0.02	20.93	21.79	36.49	21.27	20.38	23.55	22.70	28.35
8	0.03	22.91	26.21	38.40	22.33	22.30	25.40	26.07	34.22
9	0.04	23.87	29.52	38.48	22.68	23.19	25.63	27.51	36.21
10	0.05	28.52	33.57	46.34	27.77	28.69	29.67	32.25	41.66
11	0.06	26.86	33.80	40.35	24.47	26.26	27.12	30.36	40.53
12	0.07	26.91	34.34	41.91	26.05	27.92	28.88	32.69	42.09
13	0.08	25.54	33.46	37.80	24.61	26.36	26.64	31.62	42.58
14	0.09	24.60	33.21	37.71	25.01	28.18	27.38	32.19	43.69
15	0.1	25.99	35.61	39.30	25.54	28.00	29.12	33.81	46.08
16	0.2	27.97	41.28	41.66	29.36	33.48	33.54	40.20	57.02
17	0.3	28.41	40.68	38.05	28.25	34.05	32.88	41.53	59.39
18	0.4	33.39	52.19	41.45	33.58	41.14	39.98	50.68	74.25
19	0.5	36.85	54.68	42.40	36.86	46.14	44.68	56.76	81.71
20	0.75	39.44	59.53	45.88	40.36	50.28	52.67	69.07	101.71
21	1	57.22	104.58	48.44	46.14	57.55	61.59	78.23	114.15
Average		24.57	33.82	35.29	24.37	26.67	28.30	32.36	44.69

[+]Used CPLEX preprocessing parameters of presolve = off and predual = off. $\mathbf{1}^T\mathbf{x} \le M = 10^{10}$ was used as the bounding constraint; [++]Average of 5 instances of LPs at each density.

with 500 cuts per iteration shows a faster solution time than 100 or 1000 cuts. VIOL performs best for a 100-constraint cut. On the other hand, GRAD performs best for a 500-constraint cut. In fact, the 500-constraint cut for GRAD performs as well as the GRAD dynamic active-set approach. However, determining an optimum number of cuts for a given problem is not possible.

5. Conclusion

In this paper, dynamic active-set methods have been proposed for both NNLPs and LPs. In particular, these new approaches were compared to existing methods for problems with various sizes and densities. On average, dynamic RAD shows superior performance over COST RAD for the NNLP problem sets 2, 3, and 4. In the LP

problem set, dynamic GRAD significantly outperformed the COST GRAD as well as the CPLEX primal simplex and the dual simplex. In this LP problem set, however, the barrier solver did outperform all methods for densities up to 0.03. In addition, dynamic GRAD outperformed a dynamic version of VIOL, which was a standard method in column generation and decomposition methods.

References

[1] Bixby, R.E., Gregory, J.W., Lustig, I.J., Marsten, R.E. and Shanno, D.F. (1992) Very Large-Scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods. *Operations Research*, **40**, 885-897. http://dx.doi.org/10.1287/opre.40.5.885

[2] Rosenberger, J.M., Johnson, E.L. and Nemhauser, G.L. (2003) Rerouting Aircraft for Airline Recovery. *Transportation Science*, **37**, 408-421. http://dx.doi.org/10.1287/trsc.37.4.408.23271

[3] Todd, M.J. (2002) The Many Facets of Linear Programming. *Mathematical Programming*, **91**, 417-436. http://dx.doi.org/10.1007/s101070100261

[4] Elwes, R. (2012) The Algorithm That Runs the World. *New Scientist*, **215**, 32-37. http://dx.doi.org/10.1016/S0262-4079(12)62078-8

[5] Dare, P. and Saleh, H. (2000) GPS Network Design: Logistics Solution Using Optimal and Near-Optimal Methods. *Journal of Geodesy*, **74**, 467-478. http://dx.doi.org/10.1007/s001900000104

[6] Saito, G., Corley, H.W., Rosenberger, J.M., Sung, T.-K. and Noroziroshan, A. (2015) Constraint Optimal Selection Techniques (COSTs) for Nonnegative Linear Programming Problems. *Applied Mathematics and Computation*, **251**, 586-598. http://dx.doi.org/10.1016/j.amc.2014.11.080

[7] Stone, J.J. (1958) The Cross-Section Method, an Algorithm for Linear Programming. DTIC Document, P-1490, 24.

[8] Thompson, G.L., Tonge, F.M. and Zionts, S. (1996) Techniques for Removing Nonbinding Constraints and Extraneous Variables from Linear Programming Problems. *Management Science*, **12**, 588-608. http://dx.doi.org/10.1287/mnsc.12.7.588

[9] Myers, D.C. and Shih, W. (1988) A Constraint Selection Technique for a Class of Linear Programs. *Operations Research Letters*, **7**, 191-195. http://dx.doi.org/10.1016/0167-6377(88)90027-2

[10] Curet, N.D. (1993) A Primal-Dual Simplex Method for Linear Programs. *Operations Research Letters*, **13**, 233-237. http://dx.doi.org/10.1016/0167-6377(93)90045-I

[11] Adler, I., Karp, R. and Shamir, R. (1986) A Family of Simplex Variants Solving an m× d Linear Program in Expected Number of Pivot Steps Depending on d Only. *Mathematics of Operations Research*, **11**, 570-590. http://dx.doi.org/10.1287/moor.11.4.570

[12] Zeleny, M. (1986) An External Reconstruction Approach (ERA) to Linear Programming. *Computers & Operations Research*, **13**, 95-100. http://dx.doi.org/10.1016/0305-0548(86)90067-5

[13] Mitchell, J.E. (2000) Computational Experience with an Interior Point Cutting Plane Algorithm. *SIAM Journal on Optimization*, **10**, 1212-1227. http://dx.doi.org/10.1137/S1052623497324242

[14] Corley, H.W., Rosenberger, J., Yeh, W.-C. and Sung, T.K. (2006) The Cosine Simplex Algorithm. *The International Journal of Advanced Manufacturing Technology*, **27**, 1047-1050. http://dx.doi.org/10.1007/s00170-004-2278-1

[15] Junior, H.V. and Lins, M.P.E. (2005) An Improved Initial Basis for the Simplex Algorithm. *Computers & Operations Research*, **32**, 1983-1993. http://dx.doi.org/10.1016/j.cor.2004.01.002

[16] Corley, H.W. and Rosenberger, J.M. (2011) System, Method and Apparatus for Allocating Resources by Constraint Selection. US Patent No. 8082549.

[17] Saito, G., Corley, H.W. and Rosenberger, J. (2012) Constraint Optimal Selection Techniques (COSTs) for Linear Programming. *American Journal of Operations Research*, **3**, 53-64. http://dx.doi.org/10.4236/ajor.2013.31004