International conference “Mathematical Optimization Theory and Operations Research”
(**MOTOR 2021**)

A new approach to linear programs with many two-sided constraints

P. Stetsyuk, A. Fischer, O. Pichugina

V.M. Glushkov Institute of Cybernetics of NAS of Ukraine (Ukraine)
Technische Universität Dresden (Germany)
National Aerospace University «Kharkiv Aviation Institute» (Ukraine)

06.07.2021

LPs with much more constraints than variables: application area

- robust linear optimization problems with a large finite uncertainty set (RLP);
- Danzig-Wolfe and Benders decomposition schemes;
- minimax and maximin problems;
- approximation problems with Chebyshev's minimax criterion;
- Boolean estimation problems.

$$(RLP) \quad \min_{x \in \mathbb{R}^n} \left\{ c^\top x : \; B(\xi)\, x - b(\xi) \le 0 \;\forall \xi \in U \right\}.$$

$c \in \mathbb{R}^n$, ξ is the parameter vector, U is an uncertainty set, $B(\xi)$ is a matrix, $b(\xi)$ is a vector, $|U| \gg n$.

LPs with Two-Sided Constraints (LP1)

The general LP:

$$(LP) \quad \min_{x\in\mathbb{R}^n} \quad c^\top x \qquad \text{subject to} \qquad Ax \le u,$$

An LP we focus on is

$$(LP1) \quad \min_{x\in\mathbb{R}^n} \quad c^\top x \qquad \text{subject to} \qquad l \le Ax \le u,$$

Here, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $l, u \in \mathbb{R}^m$ are given, and $\langle x^*, c_* \rangle$, where $c_* = c^\top x^*$, is to be found.

If $c_* > -\infty$, LP is equivalent to LP1 with $l = -Me$, where $e \in \mathbb{R}^m$ is a vector of ones, $M > 0$ is large.

Formation of a non-smooth penalty function associated with LP1

The goal is to develop a penalty approach to solving LP1.
Let us introduce a penalty problem

$$(PP) \quad \min_{x\in\mathbb{R}^n} c_P(x)$$

of minimizing the penalty function

$$c_P(x) := c^\top x + P \cdot \max\left\{\|(Ax-u)_+\|_\infty,\ \|(l-Ax)_+\|_\infty\right\}, \qquad (1)$$

where $v_+ := (\max\{0, v_1,\}, \ldots, \max\{0, v_m\})^\top$ for any $v \in \mathbb{R}^m$ and $P > 0$ denotes a penalty parameter.

$c_P(x)$ **properties:**

- c_P is convex;
- c_P is piece-wise linear;
- $c_P(x) = c^\top x$ iff x is a feasible for LP1.

Conditions on LP1 and PP equivalence

Due to the above assumption that the LP1 has at least one solution x^*, there are Lagrange multiplier vectors $\lambda^* \in \mathbb{R}^m$ (associated to the constraints $l - Ax \leq 0$) and $\Lambda^* \in \mathbb{R}^m$ (associated to $Ax - u \leq 0$) so that $(x^*, \lambda^*, \Lambda^*)$ satisfy the Karush-Kuhn-Tucker conditions for LP1:

$$\begin{aligned} & c - A^\top \lambda + A^\top \Lambda = 0, \\ & (l - Ax)^\top \lambda = 0, \quad (Ax - u)^\top \Lambda = 0, \quad \lambda \geq 0, \\ & \Lambda \geq 0, \quad l \leq Ax \leq u. \end{aligned} \tag{2}$$

We use the following theorem obtained by applying Theorem 27 in [1] to LP1.

Theorem 1.1.

Let x^* be an LP1 solution and λ^*, Λ^* be Lagrange multiplier vectors so that $(x^*, \lambda^*, \Lambda^*)$ satisfies the Karush-Kuhn-Tucker conditions (2).

(a) *If $P \geq P_* := \|\lambda^*\|_1 + \|\Lambda^*\|_1$, then $\min\limits_{x \in \mathbb{R}^n} c_P(x) = c_*$ holds.*

(b) *If $P > P_*$, then the set X_P of minimizers of c_P coincides with X^*.*

Here, X^* and X_P are sets of minimizers of LP1 and PP, respectively, i.e.,

$$X^* = \left\{ x \in \mathbb{R}^n : \; c^\top x = c_* \right\}, \; X_P = \left\{ \hat{x} \in \mathbb{R}^n : \; c_P(\hat{x}) = \min_{x \in \mathbb{R}^n} c_P(x) \right\}.$$

Calculating the subgradient $g_{c_P}(x)$ of $c_P(x)$

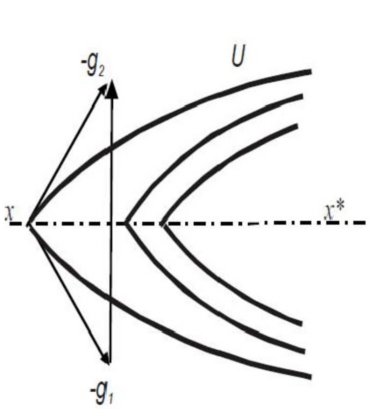
Given $x \in \mathbb{R}^n$, we use the following formula for selecting subgradient $g_{c_P}(x) \in \partial c_P(x)$:

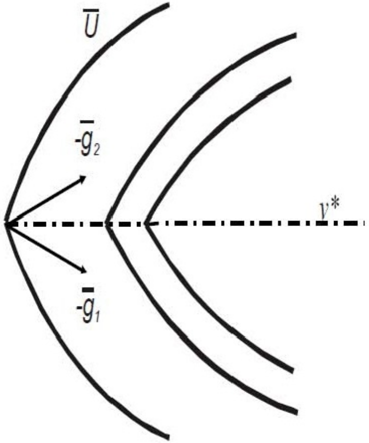
$$g_{c_P}(x) = c + P \cdot \begin{cases} 0 \in \mathbb{R}^n, & \text{if} \quad t_1 \le 0 \text{ and } t_2 \le 0, \\ (a_{i_1 1}, \dots, a_{i_1 n})^\top, & \text{if} \quad t_1 > 0 \text{ and } t_1 \ge t_2, \\ -(a_{i_2 1}, \dots, a_{i_2 n})^\top, & \text{if} \quad t_2 > 0 \text{ and } t_2 > t_1, \end{cases} \qquad (3)$$

where $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, the numbers t_1, t_2 and indices i_1, i_2 are defined by

$$\begin{array}{rcccll} t_1 & := & (Ax - u)_{i_1} & \ge & (Ax - u)_i & \forall i = 1, \dots, m, \\ t_2 & := & (l - Ax)_{i_2} & \ge & (l - Ax)_i & \forall i = 1, \dots, m. \end{array}$$

The main idea of the Shor's $r(\alpha)$-algorithm for non-smooth optimization





The idea allows designing an algorithm that utilizes a space dilation and is monotone or almost monotone with respect to the objective function.

The Shor's $r(\alpha)$-algorithm

Definition 1.2.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function and $g_f(x)$ be an element of the subdifferential $\partial f(x)$ for $x \in \mathbb{R}^n$. Moreover, let $x_0 \in \mathbb{R}^n$, $h_0 > 0$, $B_0 := I \in \mathbb{R}^{n \times n}$, and $\alpha > 1$ be given. Then, the $r(\alpha)$-algorithm is an iterative procedure that constructs sequences $\{x_k\}, \{\xi_k\}, \{\eta_k\} \subset \mathbb{R}^n$, $\{h_k\} \subset \mathbb{R}^1_{>0}$ according to the following scheme:

$$x_{k+1} := x_k - h_k B_k \xi_k, \quad B_{k+1} := B_k R_\beta(\eta_k), \quad k = 0, 1, 2, \ldots,$$

where $\|B_k^\top g_f(x_k)\| \neq 0$, $\|B_k^\top r_k\| \neq 0$,

$$\xi_k := \frac{B_k^\top g_f(x_k)}{\left\|B_k^\top g_f(x_k)\right\|}, \quad h_k \geq h_k^* := \operatorname*{argmin}_{h \geq 0} f(x_k - hB_k\xi_k),$$

$$\eta_k := \frac{B_k^\top r_k}{\left\|B_k^\top r_k\right\|}, \quad r_k := g_f(x_{k+1}) - g_f(x_k),$$

$$R_\beta(\eta_k) := I + (\beta - 1)\eta_k \eta_k^\top, \quad \beta := \frac{1}{\alpha} < 0.$$

Here, α is a parameter for space dilation, x_0 is a starting point, h_k is a step length in the normalized direction $-B_k\xi_k$, h_k^* is the step size to the minimum of the function in this direction, ξ_k is a normalized subgradient in transformed subgradient space, η_k is a normalized difference of consecutive subgradients in the transformed space, r_k is a difference of consecutive subgradients in the original space, $R_\beta(\eta_k)$ is an operator of a compression of the transformed space in the normalized direction η_k with the coefficient $\beta = 1/\alpha < 1$ on iteration k.

The Shor's $r(\alpha)$-algorithm (cont.)

To **stop** the $r(\alpha)$-algorithm after x_k is obtained, the following termination criteria are used:

- $\|x_k - x_{k-1}\| \le \varepsilon_x$ (common for non-smooth problems),
- $\|g_f(x_k)\| \le \varepsilon_g$ (common for smooth problems),
- $k \ge maxitn$ (common for general problems)

The parameters $\varepsilon_x > 0$, $\varepsilon_g > 0$, and a large natural number *maxitn* are predefined values.

Variants of the $r(\alpha)$-**algorithm** are characterized by a way to choose h_k:

- $h_k = h_k^*$ ($h_k \approx h_k^*$) meaning an exact (approximate) one-dimensional search for the minimum of $g_f(x)$ in the direction of its descent;
- the value h_k is tuned (adapted) during the one-dimensional descent (the process is characterized by a step reduction coefficient $q_1 \le 1$). These are $r(\alpha)$-algorithms with the adaptive step adjustment.

Computational Experiments 1, 2

The Linear Programming by r-algorithm (**LPralg**) is an approach to solve LP1 by switching to an equivalent non-smooth optimization problem PP, which is solved by the variant of r(α)-algorithm supporting the adaptive step adjustment.

LPralg was implemented in GNU Octave (GNU Octave version 5.1.0).

The **first experiment** is done with this LPralg implementation. Also, we apply the GLP_PRIMAL solver from the GNU Linear Programming Kit (GLPK) to directly solve these test problems. Both algorithms were run on the same machine, an Intel Core i5-9400f processor with 2.9 GHz and 16GB RAM.

For the **second experiment**, we run test problems on the NEOS server with solvers from CPLEX and Gurobi.

The entries of the vector c and the matrix $A = (a_{ij})$ were generated randomly as the following:

$$c_j \sim U(0,1), \quad a_{ij} \sim U(0,1) \qquad \text{for } i, \dots, m \quad \text{and} \quad j = 1, \dots, n.$$

The lower and upper bound vectors l and u were defined by

$$l_i := 0.9 \sum_{j=1}^{n} a_{ij}, \quad u_i := 1.1 \sum_{j=1}^{n} a_{ij} \qquad \text{for } i, \dots, m.$$

The first experiment details

Octave functions [2]:

- **ralgb5a** – is an Octave function implementing the $r(\alpha)$-algorithm;
- **fgLP** – is an Octave function, which implements formula (3) for evaluating value of $c(P)$ and the subgradient $g_{c_P}(x)$ in ralgb5a.

The **ralgb5a** parameters are:

$$\texttt{x} := 0 \in \mathbb{R}^n, \quad \texttt{alpha} := 4, \quad \texttt{q1} := 1.0, \quad \texttt{h0} := 1; \tag{4}$$

the **fgLP** parameter is:

$$\texttt{P} := 10;$$

the termination parameters are given by

$$\texttt{epsx} := 10^{-10}, \quad \texttt{epsg} := 10^{-8}, \quad \texttt{maxitn} := 10.000.$$

All generated test problems are characterized by a small number $n \in \{100, 200, 300\}$ of variables and a much larger number m of two-sided constraints with

$$m \in \{20.000, 50.000, 100.000, 200.000\}.$$

First Experiment: Comparison of GLPK and LPralg for LP1

n	m	t_{GLPK}	t_{LPralg}	$t_{\text{GLPK}}/t_{\text{LPralg}}$	Δ
100	20.000	23.7	3.4	7.0	2.7e-08
	50.000	81.0	6.5	**12.5**	2.0e-08
	100.000	107	17.6	6.1	8.5e-09
	200.000	271	35.5	7.6	2.8e-08
200	20.000	58.6	13.2	4.4	4.4e-08
	50.000	155	30.5	5.1	2.3e-08
	100.000	325	61.6	5.3	4.2e-08
	200.000	708	139	5.1	5.1e-08
300	20.000	125	27.6	4.5	3.3e-07
	50.000	323	65.5	4.9	4.1e-08
	100.000	672	172	**3.9**	1.4e-07
	200.000	2 025	388	5.2	4.4e-07

* Run time is given in seconds

The second experiment details

In the second experiment, run times spent by LPralg and commercial solvers CPLEX and Gurobi are compared. For that, we run the solvers on the NEOS server. To compare the performance of CPLEX and Gurobi on NEOS and on our machine, we performed auxiliary computational tests confirming that the NEOS run times are close to those produced on our machine. The table shows the times for the NEOS server for the same test problems.

For LPralg, the same parameters as in (4) are used. The termination parameters are now $\texttt{epsx} := 10^{-9}$, $\texttt{epsg} := 10^{-8}$, $\texttt{maxitn} := 20.000$, $\texttt{intp} := 1.000$. The table shows the run times of LPralg not only for $\texttt{q1} := 1.0$ as in (4) but also for $\texttt{q1} := 0.95$.

Table: Second Experiment: Run times for LPralg (in seconds)

n	m	$\texttt{q1} = 0.95$	$\texttt{q1} = 1.0$
100	300.000	164	58.3
50	600.000	59.7	46.4
20	1.500.000	33.5	24.3

The second experiment results

To finally apply CPLEX and Gurobi to the same test problems as used in previous table, this table shows the results obtained with the NEOS server. To reduce time differences, which result from concrete situations at NEOS, we have performed 4 runs for the same test problem for each of the two solvers. The table presents the average run time.

If we compare the results in the table with those in previous one, we see that LPralg is comparable and often seems to be faster by a factor of $2-4$ than CPLEX or Gurobi.

Table: Second Experiment: Averaged run times (in seconds) on the NEOS server

CPLEX		
n	m	averaged run time
100	300.000	202
50	600.000	151
20	1.500.000	142
Gurobi		
n	m	averaged run time
100	300.000	108
50	600.000	148
20	1.500.000	84

Conclusion

We presents the algorithm LPralg, intended to solve linear programs with many two-sided constraints, and its software implementation in the Octave programming language. LPralg applies a penalty approach to reformulate a linear program equivalently as an unconstrained problem of minimizing a non-smooth convex penalty function. The later is tackled by a version of Shor's *r*-algorithm supporting an adaptive step length adjustment.

The results for cases with much more two-sided constraints than variables and comparisons with free and commercial linear programming tools are quite promising.

This approach can be directly generalized to convex nonlinear optimization problems, in particular, to quadratic programming ones and problems with convex quadratic objective function and constraints.

References

Shor, N.Z.: Nondifferentiable Optimization and Polynomial Problems. Kluwer Academic Publishers, Dordrecht (1998). https://doi.org/10.1007/978-1-4757-6015-6

Stetsyuk, P., Fischer, F., Pichugina O. A Penalty Approach to Linear Programs with Many Two-Sided Constraints. In: Lecture Notes in Computer Science book series (LNCS, volume 12755). Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-77876-7_14

Thank you!
Questions?